FreeMat v4.1 Documentation

Samit Basu

July 10, 2011

Contents

Chapter 1

Introduction and Getting Started

1.1 INSTALL Installing FreeMat

1.1.1 General Instructions

Here are the general instructions for installing FreeMat. First, follow the instructions listed below for the platform of interest. Then, run the

```
-->pathtool
```

which brings up the path setup tool. More documentation on the GUI elements (and how to use them) will be forthcoming.

1.1.2 Linux

For Linux, FreeMat is now provided as a binary installation. To install it simply download the binary using your web browser, and then unpack it

```
tar xvfz FreeMat-<VERSION_NUMBER>-Linux-Binary.tar.gz
```

You can then run FreeMat directly without any additional effort

```
FreeMat-<VERSION_NUMBER>-Linux-Binary/Contents/bin/FreeMat
```

will start up FreeMat as an X application. If you want to run it as a command line application (to run from within an xterm), use the `nogui` flag

```
FreeMat-<VERSION_NUMBER>-Linux-Binary/Contents/bin/FreeMat -nogui
```

If you do not want FreeMat to use X at all (no graphics at all), use the `noX` flag

```
FreeMat-<VERSION_NUMBER>-Linux-Binary/Contents/bin/FreeMat -noX
```

For convenience, you may want to add FreeMat to your path. The exact mechanism for doing this depends on your shell. Assume that you have unpacked `FreeMat-<VERSION\_NUMBER>-Linux-Binary.tar.gz` into the directory `/home/myname`. Then if you use `csh` or its derivatives (like `tcsh`) you should add the following line to your `.cshrc` file:

```
set path=($path /home/myname/FreeMat-<VERSION_NUMBER>-Linux/Binary/Contents/bin)
```

If you use `bash`, then add the following line to your `.bash\_profile`

```
PATH=$PATH:/home/myname/FreeMat-<VERSION_NUMBER>-Linux/Binary/Contents/bin
```

If the prebuilt binary package does not work for your Linux distribution, you will need to build FreeMat from source (see the source section below). When you have FreeMat running, you can setup your path using the `pathtool`. Note that the `FREEMAT\_PATH` is no longer used by FreeMat. You must use the `pathtool` to adjust the path.

1.1.3 Windows

For Windows, FreeMat is installed via a binary installer program. To use it, simply download the setup program `FreeMat-<VERSION\_NUMBER>-Setup.exe`, and double click it. Follow the instructions to do the installation, then setup your path using `pathtool`.

1.1.4 Mac OS X

For Mac OS X, FreeMat is distributed as an application bundle. To install it, simply download the compressed disk image file `FreeMat-<VERSION\_NUMBER>.dmg`, double click to mount the disk image, and then copy the application `FreeMat-<VERSION\_NUMBER>` to some convenient place. To run FreeMat, simply double click on the application. Run `pathtool` to setup your FreeMat path.

1.1.5 Source Code

The source code build is a little more complicated than previous versions of FreeMat. Here are the current build instructions for all platforms.

1. Build and install Qt 4.3 or later - `http://trolltech.com/developer/downloads/opensource`
2. Install g77 or gfortran (use fink for Mac OS X, use `gcc-g77` package for MinGW)
3. Download the source code `FreeMat-<VERSION\_NUMBER>-src.tar.gz`.
4. Unpack the source code: `tar xvfz FreeMat-<VERSION\_NUMBER>-src.tar.gz`.
5. For Windows, you will need to install MSYS as well as MINGW to build FreeMat. You will also need unzip to unpack the enclosed matio.zip archive. Alternately, you can cross-build the WIndows version of FreeMat under Linux (this is how I build it now).
6. If you are extraordinarily lucky (or prepared), you can issue the usual ./configure, then the make and make install. This is not likely to work because of the somewhat esoteric dependencies of FreeMat. The configure step will probably fail and indicate what external dependencies are still needed.
7. I assume that you are familiar with the process of installing dependencies if you are trying to build FreeMat from source.

To build a binary distributable (app bundle on the Mac, setup installer on win32, and a binary distribution on Linux), you will need to run `make package` instead of `make install`.

Chapter 2

Variables and Arrays

2.1 CELL Cell Array Definitions

2.1.1 Usage

The cell array is a fairly powerful array type that is available in FreeMat. Generally speaking, a cell array is a heterogenous array type, meaning that different elements in the array can contain variables of different type (including other cell arrays). For those of you familiar with `C`, it is the equivalent to the `void *` array. The general syntax for their construction is

```
A = {row_def1;row_def2;...;row_defN}
```

where each row consists of one or more elements, seperated by commas

```
row_defi = element_i1,element_i2,...,element_iM
```

Each element can be any type of FreeMat variable, including matrices, arrays, cell-arrays, structures, strings, etc. The restriction on the definition is that each row must have the same number of elements in it.

2.1.2 Examples

Here is an example of a cell-array that contains a number, a string, and an array

```
--> A = {14,'hello',[1:10]}

A =
 [14] [hello] [1x10 double array]
```

Note that in the output, the number and string are explicitly printed, but the array is summarized. We can create a 2-dimensional cell-array by adding another row definition

```
--> B = {pi,i;e,-1}

B =
 [3.14159] [0+1i]
 [2.71828] [-1]
```

Finally, we create a new cell array by placing `A` and `B` together

```
--> C = {A,B}

C =
 [1x3 cell array] [2x2 cell array]
```

2.2 FUNCTIONHANDLES Function Handles

2.2.1 Usage

Starting with version 1.11, FreeMat now supports `function handles`, or `function pointers`. A `function handle` is an alias for a function or script that is stored in a variable. First, the way to assign a function handle is to use the notation

```
handle = @func
```

where `func` is the name to point to. The function `func` must exist at the time we make the call. It can be a local function (i.e., a subfunction). To use the `handle`, we can either pass it to `feval` via

```
[x,y] = feval(handle,arg1,arg2).
```

Alternately, you can the function directly using the notation

```
[x,y] = handle(arg1,arg2)
```

2.3 GLOBAL Global Variables

2.3.1 Usage

Global variables are shared variables that can be seen and modified from any function or script that declares them. The syntax for the `global` statement is

```
global variable_1 variable_2 ...
```

The `global` statement must occur before the variables appear.

2.3.2 Example

Here is an example of two functions that use a global variable to communicate an array between them. The first function sets the global variable.

```
    set_global.m
function set_global(x)
  global common_array
  common_array = x;
```

The second function retrieves the value from the global variable

```
    get_global.m
function x = get_global
  global common_array
  x = common_array;
```

Here we exercise the two functions

```
--> set_global('Hello')
--> get_global

ans =
Hello
```

2.4 INDEXING Indexing Expressions

2.4.1 Usage

There are three classes of indexing expressions available in FreeMat: `()`, `{}`, and `.` Each is explained below in some detail, and with its own example section.

2.4.2 Array Indexing

We start with array indexing `()`, which is the most general indexing expression, and can be used on any array. There are two general forms for the indexing expression - the N-dimensional form, for which the general syntax is

```
variable(index_1,index_2,...,index_n)
```

and the vector form, for which the general syntax is

```
variable(index)
```

Here each index expression is either a scalar, a range of integer values, or the special token `:`, which is shorthand for `1:end`. The keyword `end`, when included in an indexing expression, is assigned the length of the array in that dimension. The concept is easier to demonstrate than explain. Consider the following examples:

```
--> A = zeros(4)

A =
 0 0 0 0
 0 0 0 0
 0 0 0 0
 0 0 0 0

--> B = float(randn(2))

B =
   -0.1688    0.5183
    0.9485   -0.6864

--> A(2:3,2:3) = B

A =
         0         0         0         0
         0   -0.1688    0.5183         0
         0    0.9485   -0.6864         0
         0         0         0         0
```

Here the array indexing was used on the left hand side only. It can also be used for right hand side indexing, as in

```
--> C = A(2:3,1:end)

C =
         0   -0.1688    0.5183         0
         0    0.9485   -0.6864         0
```

Note that we used the `end` keyword to avoid having to know that `A` has 4 columns. Of course, we could also use the `:` token instead:

```
--> C = A(2:3,:)

C =
         0   -0.1688    0.5183         0
         0    0.9485   -0.6864         0
```

An extremely useful example of `:` with array indexing is for slicing. Suppose we have a 3-D array, that is `2 x 2 x 3`, and we want to set the middle slice:

```
--> D = zeros(2,2,3)

D =

(:,:,1) =
 0 0
 0 0

(:,:,2) =
 0 0
 0 0

(:,:,3) =
 0 0
 0 0

--> D(:,:,2) = int32(10*rand(2,2))

D =

(:,:,1) =
  0  0
  0  0

(:,:,2) =
  9 10
  5  8

(:,:,3) =
  0  0
  0  0
```

In another level of nuance, the assignment expression will automatically fill in the indexed rectangle on the left using data from the right hand side, as long as the lengths match. So we can take a vector and roll it into a matrix using this approach:

```
--> A = zeros(4)

A =
 0 0 0 0
 0 0 0 0
 0 0 0 0
 0 0 0 0

--> v = [1;2;3;4]

v =
 1
 2
 3
 4

--> A(2:3,2:3) = v

A =
```

```
 0 0 0 0
 0 1 3 0
 0 2 4 0
 0 0 0 0
```

The N-dimensional form of the variable index is limited to accessing only (hyper-) rectangular regions of the array. You cannot, for example, use it to access only the diagonal elements of the array. To do that, you use the second form of the array access (or a loop). The vector form treats an arbitrary N-dimensional array as though it were a column vector. You can then access arbitrary subsets of the arrays elements (for example, through a `find` expression) efficiently. Note that in vector form, the `end` keyword takes the meaning of the total length of the array (defined as the product of its dimensions), as opposed to the size along the first dimension.

2.4.3 Cell Indexing

The second form of indexing operates, to a large extent, in the same manner as the array indexing, but it is by no means interchangable. As the name implies, `cell`-indexing applies only to `cell` arrays. For those familiar with `C`, cell- indexing is equivalent to pointer derefencing in `C`. First, the syntax:

```
  variable{index_1,index_2,...,index_n}
```

and the vector form, for which the general syntax is

```
  variable{index}
```

The rules and interpretation for N-dimensional and vector indexing are identical to `()`, so we will describe only the differences. In simple terms, applying `()` to a cell-array returns another cell array that is a subset of the original array. On the other hand, applying `{}` to a cell-array returns the contents of that cell array. A simple example makes the difference quite clear:

```
--> A = {1, 'hello', [1:4]}

A =
 [1] [hello] [1x4 double array]

--> A(1:2)

ans =
 [1] [hello]

--> A{1:2}

ans =

1 of 2:
 1


2 of 2:
hello
```

You may be surprised by the response to the last line. The output is multiple assignments to `ans`!. The output of a cell-array dereference can be used anywhere a list of expressions is required. This includes arguments and returns for function calls, matrix construction, etc. Here is an example of using cell-arrays to pass parameters to a function:

```
--> A = {[1,3,0],[5,2,7]}

A =
 [1x3 double array] [1x3 double array]

--> max(A{1:end})

ans =
 5 3 7
```

And here, cell-arrays are used to capture the return.

```
--> [K{1:2}] = max(randn(1,4))
K =
 [0.779398] [4]
```

Here, cell-arrays are used in the matrix construction process:

```
--> C = [A{1};A{2}]

C =
 1 3 0
 5 2 7
```

Note that this form of indexing is used to implement variable length arguments to function. See `varargin` and `varargout` for more details.

2.4.4 Structure Indexing

The third form of indexing is structure indexing. It can only be applied to structure arrays, and has the general syntax

```
variable.fieldname
```

where `fieldname` is one of the fields on the structure. Note that in FreeMat, fields are allocated dynamically, so if you reference a field that does not exist in an assignment, it is created automatically for you. If variable is an array, then the result of the `.` reference is an expression list, exactly like the `{}` operator. Hence, we can use structure indexing in a simple fashion:

```
--> clear A
--> A.color = 'blue'

A =
    color: blue
--> B = A.color

B =
blue
```

Or in more complicated ways using expression lists for function arguments

```
--> clear A
--> A(1).maxargs = [1,6,7,3]

A =
    maxargs: 1x4 double array
--> A(2).maxargs = [5,2,9,0]
```

```
A =
1x2 struct array with fields:
    maxargs
--> max(A.maxargs)

ans =
 5 6 9 3
```

or to store function outputs

```
--> clear A
--> A(1).maxreturn = [];
--> A(2).maxreturn = [];
--> [A.maxreturn] = max(randn(1,4))
A =
1x2 struct array with fields:
    maxreturn
```

FreeMat now also supports the so called dynamic-field indexing expressions. In this mode, the fieldname is supplied through an expression instead of being explicitly provided. For example, suppose we have a set of structure indexed by color,

```
--> x.red = 430;
--> x.green = 240;
--> x.blue = 53;
--> x.yello = 105

x =
    red: 430
    green: 240
    blue: 53
    yello: 105
```

Then we can index into the structure `x` using a dynamic field reference:

```
--> y = 'green'

y =
green
--> a = x.(y)

a =
 240
```

Note that the indexing expression has to resolve to a string for dynamic field indexing to work.

2.4.5 Complex Indexing

The indexing expressions described above can be freely combined to affect complicated indexing expressions. Here is an example that exercises all three indexing expressions in one assignment.

```
--> Z{3}.foo(2) = pi

Z =
 [0] [0] [1x1 struct array]
```

From this statement, FreeMat infers that Z is a cell-array of length 3, that the third element is a structure array (with one element), and that this structure array contains a field named 'foo' with two double elements, the second of which is assigned a value of pi.

2.5 MATRIX Matrix Definitions

2.5.1 Usage

The matrix is the basic datatype of FreeMat. Matrices can be defined using the following syntax

```
A = [row_def1;row_def2;...,row_defN]
```

where each row consists of one or more elements, seperated by commas

```
row_defi = element_i1,element_i2,...,element_iM
```

Each element can either be a scalar value or another matrix, provided that the resulting matrix definition makes sense. In general this means that all of the elements belonging to a row have the same number of rows themselves, and that all of the row definitions have the same number of columns. Matrices are actually special cases of N-dimensional arrays where `N<=2`. Higher dimensional arrays cannot be constructed using the bracket notation described above. The type of a matrix defined in this way (using the bracket notation) is determined by examining the types of the elements. The resulting type is chosen so no information is lost on any of the elements (or equivalently, by choosing the highest order type from those present in the elements).

2.5.2 Examples

Here is an example of a matrix of `int32` elements (note that untyped integer constants default to type `int32`).

```
--> A = [1,2;5,8]

A =
 1 2
 5 8
```

Now we define a new matrix by adding a column to the right of `A`, and using float constants.

```
--> B = [A,[3.2f;5.1f]]

B =
    1.0000    2.0000    3.2000
    5.0000    8.0000    5.1000
```

Next, we add extend `B` by adding a row at the bottom. Note how the use of an untyped floating point constant forces the result to be of type `double`

```
--> C = [B;5.2,1.0,0.0]

C =
    1.0000    2.0000    3.2000
    5.0000    8.0000    5.1000
    5.2000    1.0000         0
```

If we instead add a row of `complex` values (recall that `i` is a `complex` constant, not a `dcomplex` constant)

```
--> D = [B;2.0f+3.0f*i,i,0.0f]

D =
   1.0000 +  0.0000i   2.0000 +  0.0000i   3.2000 +  0.0000i
   5.0000 +  0.0000i   8.0000 +  0.0000i   5.1000 +  0.0000i
   2.0000 +  3.0000i   0.0000 +  1.0000i        0
```

Likewise, but using `dcomplex` constants

```
--> E = [B;2.0+3.0*i,i,0.0]

E =
   1.0000 +  0.0000i   2.0000 +  0.0000i   3.2000 +  0.0000i
   5.0000 +  0.0000i   8.0000 +  0.0000i   5.1000 +  0.0000i
   2.0000 +  3.0000i   0.0000 +  1.0000i        0
```

Finally, in FreeMat, you can construct matrices with strings as contents, but you have to make sure that if the matrix has more than one row, that all the strings have the same length.

```
--> F = ['hello';'there']

F =
hello
there
```

2.6 PERSISTENT Persistent Variables

2.6.1 Usage

Persistent variables are variables whose value persists between calls to a function or script. The general syntax for its use is

```
persistent variable1 variable2 ... variableN
```

The `persistent` statement must occur before the variable is the tagged as persistent. Per the MATLAB API documentation an empty variable is created when the `persistent` statement is called.

2.6.2 Example

Here is an example of a function that counts how many times it has been called.

```
    count_calls.m
function count_calls
  persistent ccount
  if isempty(ccount); ccount = 0; end;
  ccount = ccount + 1;
  printf('Function has been called %d times\n',ccount);
```

We now call the function several times:

```
--> for i=1:10; count_calls; end
Function has been called 1 times
Function has been called 2 times
Function has been called 3 times
Function has been called 4 times
Function has been called 5 times
Function has been called 6 times
Function has been called 7 times
Function has been called 8 times
Function has been called 9 times
Function has been called 10 times
```

2.7 STRUCT Structure Array Constructor

2.7.1 Usage

Creates an array of structures from a set of field, value pairs. The syntax is

```
y = struct(n_1,v_1,n_2,v_2,...)
```

where `n\_i` are the names of the fields in the structure array, and `v\_i` are the values. The values `v\_i` must either all be scalars, or be cell-arrays of all the same dimensions. In the latter case, the output structure array will have dimensions dictated by this common size. Scalar entries for the `v\_i` are replicated to fill out their dimensions. An error is raised if the inputs are not properly matched (i.e., are not pairs of field names and values), or if the size of any two non-scalar values cell-arrays are different.

Another use of the `struct` function is to convert a class into a structure. This allows you to access the members of the class, directly but removes the class information from the object.

2.7.2 Example

This example creates a 3-element structure array with three fields, `foo` `bar` and `key`, where the contents of `foo` and `bar` are provided explicitly as cell arrays of the same size, and the contents of `bar` are replicated from a scalar.

```
--> y = struct('foo',{1,3,4},'bar',{'cheese','cola','beer'},'key',508)

y =
1x3 struct array with fields:
    foo
    bar
    key
--> y(1)

ans =
    foo: 1
    bar: cheese
    key: 508
--> y(2)

ans =
    foo: 3
    bar: cola
    key: 508
--> y(3)

ans =
    foo: 4
    bar: beer
    key: 508
```

An alternate way to create a structure array is to initialize the last element of each field of the structure

```
--> Test(2,3).Type = 'Beer';
--> Test(2,3).Ounces = 12;
--> Test(2,3).Container = 'Can';
--> Test(2,3)

ans =
    Type: Beer
```

```
    Ounces: 12
    Container: Can
--> Test(1,1)

ans =
    Type: 0
    Ounces: 0
    Container: 0
```

Chapter 3

Functions and Scripts

3.1 ANONYMOUS Anonymous Functions

3.1.1 Usage

Anonymous functions are simple, nameless functions that can be defined anywhere (in a script, function, or at the prompt). They are intended to supplant `inline` functions. The syntax for an anonymous function is simple:

```
y = @(arg1,arg2,...,argn) expression
```

where `arg1,arg2,...,argn` is a list of valid identifiers that define the arguments to the function, and `expression` is the expression to compute in the function. The returned value `y` is a function handle for the anonymous function that can then be used to evaluate the expression. Note that `y` will capture the value of variables that are not indicated in the argument list from the current scope or workspace at the time it is defined. So, for example, consider the simple anonymous function definition

```
y = @(x) a*(x+b)
```

In order for this definition to work, the variables `a` and `b` need to be defined in the current workspace. Whatever value they have is captured in the function handle `y`. To change the values of `a` and `b` in the anonymous function, you must recreate the handle using another call. See the examples section for more information. In order to use the anonymous function, you can use it just like any other function handle. For example,

```
p = y(3)
p = y()
p = feval(y,3)
```

are all examples of using the `y` anonymous function to perform a calculation.

3.1.2 Examples

Here are some examples of using an anonymous function

```
--> a = 2; b = 4;    % define a and b (slope and intercept)
--> y = @(x) a*x+b   % create the anonymous function

y =
 @(x)   a*x+b   % create the anonymous function
--> y(2)             % evaluate it for x = 2

ans =
```

```
 8

--> a = 5; b = 7;    % change a and b
--> y(2)             % the value did not change!  because a=2,b=4 are captured in y

ans =
 8

--> y = @(x) a*x+b   % recreate the function

y =
 @(x)   a*x+b   % recreate the function
--> y(2)             % now the new values are used

ans =
 17
```

3.2 FUNC2STR Function to String conversion

3.2.1 Usage

The `func2str` function converts a function pointer into a string. The syntax is

```
y = func2str(funcptr)
```

where `funcptr` is a function pointer. If `funcptr` is a pointer to a function, then `y` is the name of the function. On the other hand, if `funcptr` is an anonymous function then `func2str` returns the definition of the anonymous function.

3.2.2 Example

Here is a simple example of using `func2str`

```
--> y = @sin

y =
 @sin
--> x = func2str(y)

x =
sin
```

If we use an anonymous function, then `func2str` returns the definition of the anonymous function

```
--> y = @(x) x.^2

y =
 @(x)   x.^2
--> x = func2str(y)

x =
 @(x)   x.^2
```

3.3 FUNCTION Function Declarations

3.3.1 Usage

There are several forms for function declarations in FreeMat. The most general syntax for a function declaration is the following:

```
function [out_1,...,out_M,varargout] = fname(in_1,...,in_N,varargin)
```

where `out\_i` are the output parameters, `in\_i` are the input parameters, and `varargout` and `varargin` are special keywords used for functions that have variable inputs or outputs. For functions with a fixed number of input or output parameters, the syntax is somewhat simpler:

```
function [out_1,...,out_M] = fname(in_1,...,in_N)
```

Note that functions that have no return arguments can omit the return argument list (of `out\_i`) and the equals sign:

```
function fname(in_1,...,in_N)
```

Likewise, a function with no arguments can eliminate the list of parameters in the declaration:

```
function [out_1,...,out_M] = fname
```

Functions that return only a single value can omit the brackets

```
function out_1 = fname(in_1,...,in_N)
```

In the body of the function `in\_i` are initialized with the values passed when the function is called. Also, the function must assign values for `out\_i` to pass values to the caller. Note that by default, FreeMat passes arguments by value, meaning that if we modify the contents of `in\_i` inside the function, it has no effect on any variables used by the caller. Arguments can be passed by reference by prepending an ampersand `\&` before the name of the input, e.g.

```
function [out1,...,out_M] = fname(in_1,&in_2,in_3,...,in_N)
```

in which case `in\_2` is passed by reference and not by value. Also, FreeMat works like `C` in that the caller does not have to supply the full list of arguments. Also, when `keywords` (see help `keywords`) are used, an arbitrary subset of the parameters may be unspecified. To assist in deciphering the exact parameters that were passed, FreeMat also defines two variables inside the function context: `nargin` and `nargout`, which provide the number of input and output parameters of the caller, respectively. See help for `nargin` and `nargout` for more details. In some circumstances, it is necessary to have functions that take a variable number of arguments, or that return a variable number of results. In these cases, the last argument to the parameter list is the special argument `varargin`. Inside the function, `varargin` is a cell-array that contains all arguments passed to the function that have not already been accounted for. Similarly, the function can create a cell array named `varargout` for variable length output lists. See help `varargin` and `varargout` for more details.

The function name `fname` can be any legal FreeMat identifier. Functions are stored in files with the `.m` extension. Note that the name of the file (and not the function name `fname` used in the declaration) is how the function appears in FreeMat. So, for example, if the file is named `foo.m`, but the declaration uses `bar` for the name of the function, in FreeMat, it will still appear as function `foo`. Note that this is only true for the first function that appears in a `.m` file. Additional functions that appear after the first function are known as `helper functions` or `local` functions. These are functions that can only be called by other functions in the same `.m` file. Furthermore the names of these helper functions are determined by their declaration and not by the name of the `.m` file. An example of using helper functions is included in the examples.

Another important feature of functions, as opposed to, say `scripts`, is that they have their own `scope`. That means that variables defined or modified inside a function do not affect the scope of the caller. That means that a function can freely define and use variables without unintentionally using a variable name reserved elsewhere. The flip side of this fact is that functions are harder to debug than scripts without using the `keyboard` function, because the intermediate calculations used in the function are not available once the function exits.

3.3.2 Examples

Here is an example of a trivial function that adds its first argument to twice its second argument:

```
    addtest.m
function c = addtest(a,b)
  c = a + 2*b;

--> addtest(1,3)

ans =
 7

--> addtest(3,0)

ans =
 3
```

Suppose, however, we want to replace the value of the first argument by the computed sum. A first attempt at doing so has no effect:

```
    addtest2.m
function addtest2(a,b)
  a = a + 2*b;

--> arg1 = 1

arg1 =
 1

--> arg2 = 3

arg2 =
 3

--> addtest2(arg1,arg2)
--> arg1

ans =
 1

--> arg2

ans =
 3
```

The values of `arg1` and `arg2` are unchanged, because they are passed by value, so that any changes to `a` and `b` inside the function do not affect `arg1` and `arg2`. We can change that by passing the first argument by reference:

```
    addtest3.m
function addtest3(&a,b)
  a = a + 2*b
```

Note that it is now illegal to pass a literal value for `a` when calling `addtest3`:

```
--> addtest3(3,4)

a =
 11

Error: Must have lvalue in argument passed by reference
--> addtest3(arg1,arg2)

a =
 7

--> arg1

ans =
 7

--> arg2

ans =
 3
```

The first example fails because we cannot pass a literal like the number `3` by reference. However, the second call succeeds, and note that `arg1` has now changed. Note: please be careful when passing by reference - this feature is not available in MATLAB and you must be clear that you are using it.

As variable argument and return functions are covered elsewhere, as are keywords, we include one final example that demonstrates the use of helper functions, or local functions, where multiple function declarations occur in the same file.

```
    euclidlength.m
function y = foo(x,y)
  square_me(x);
  square_me(y);
  y = sqrt(x+y);

function square_me(&t)
  t = t^2;

--> euclidlength(3,4)

ans =
 5

--> euclidlength(2,0)

ans =
 2
```

3.4 KEYWORDS Function Keywords

3.4.1 Usage

A feature of IDL that FreeMat has adopted is a modified form of `keywords`. The purpose of `keywords` is to allow you to call a function with the arguments to the function specified in an arbitrary order. To specify the syntax of `keywords`, suppose there is a function with prototype

```
function [out_1,...,out_M] = foo(in_1,...,in_N)
```

Then the general syntax for calling function `foo` using keywords is

```
foo(val_1, val_2, /in_k=3)
```

which is exactly equivalent to

```
foo(val_1, val_2, [], [], ..., [], 3),
```

where the 3 is passed as the k-th argument, or alternately,

```
foo(val_1, val_2, /in_k)
```

which is exactly equivalent to

```
foo(val_1, val_2, [], [], ..., [], logical(1)),
```

Note that you can even pass reference arguments using keywords.

3.4.2 Example

The most common use of keywords is in controlling options for functions. For example, the following function takes a number of binary options that control its behavior. For example, consider the following function with two arguments and two options. The function has been written to properly use and handle keywords. The result is much cleaner than the MATLAB approach involving testing all possible values of `nargin`, and forcing explicit empty brackets for don't care parameters.

```
    keyfunc.m
function c = keyfunc(a,b,operation,printit)
  if (~isset('a') | ~isset('b'))
    error('keyfunc requires at least the first two 2 arguments');
  end;
  if (~isset('operation'))
    % user did not define the operation, default to '+'
    operation = '+';
  end
  if (~isset('printit'))
    % user did not specify the printit flag, default is false
    printit = 0;
  end
  % simple operation...
  eval(['c = a ' operation ' b;']);
  if (printit)
    printf('%f %s %f = %f\n',a,operation,b,c);
  end
```

Now some examples of how this function can be called using `keywords`.

```
--> keyfunc(1,3)               % specify a and b, defaults for the others

ans =
 4

--> keyfunc(1,3,/printit)      % specify printit is true
1.000000 + 3.000000 = 4.000000

ans =
 4
```

```
--> keyfunc(/operation='-',2,3) % assigns a=2, b=3

ans =
 -1

--> keyfunc(4,/operation='*',/printit) % error as b is unspecified
In /home/sbasu/Devel/FreeMat/help/tmp/keyfunc.m(keyfunc) at line 3
    In scratch() at line 1
    In base(base)
    In base()
    In global()
Error: keyfunc requires at least the first two 2 arguments
```

3.5 NARGIN Number of Input Arguments

3.5.1 Usage

The `nargin` function returns the number of arguments passed to a function when it was called. The general syntax for its use is

```
  y = nargin
```

FreeMat allows for fewer arguments to be passed to a function than were declared, and `nargin`, along with `isset` can be used to determine exactly what subset of the arguments were defined.

You can also use `nargin` on a function handle to return the number of input arguments expected by the function

```
  y = nargin(fun)
```

where `fun` is the name of the function (e.g. `'sin'`) or a function handle.

3.5.2 Example

Here is a function that is declared to take five arguments, and that simply prints the value of `nargin` each time it is called.

```
    nargintest.m
function nargintest(a1,a2,a3,a4,a5)
  printf('nargin = %d\n',nargin);

--> nargintest(3);
nargin = 1
--> nargintest(3,'h');
nargin = 2
--> nargintest(3,'h',1.34);
nargin = 3
--> nargintest(3,'h',1.34,pi,e);
nargin = 5
--> nargin('sin')

ans =
 1

--> y = @sin
```

```
y =
 @sin
--> nargin(y)

ans =
 1
```

3.6 NARGOUT Number of Output Arguments

3.6.1 Usage

The `nargout` function computes the number of return values requested from a function when it was called. The general syntax for its use

```
y = nargout
```

FreeMat allows for fewer return values to be requested from a function than were declared, and `nargout` can be used to determine exactly what subset of the functions outputs are required.

You can also use `nargout` on a function handle to return the number of input arguments expected by the function

```
y = nargout(fun)
```

where `fun` is the name of the function (e.g. `'sin'`) or a function handle.

3.6.2 Example

Here is a function that is declared to return five values, and that simply prints the value of `nargout` each time it is called.

```
    nargouttest.m
function [a1,a2,a3,a4,a5] = nargouttest
  printf('nargout = %d\n',nargout);
  a1 = 1; a2 = 2; a3 = 3; a4 = 4; a5 = 5;

--> a1 = nargouttest
nargout = 1

a1 =
 1

--> [a1,a2] = nargouttest
nargout = 2
a1 =
 1

a2 =
 2

--> [a1,a2,a3] = nargouttest
nargout = 3
a1 =
 1

a2 =
 2
```

```
a3 =
 3

--> [a1,a2,a3,a4,a5] = nargouttest
nargout = 5
a1 =
 1

a2 =
 2

a3 =
 3

a4 =
 4

a5 =
 5

--> nargout('sin')

ans =
 1

--> y = @sin

y =
 @sin
--> nargout(y)

ans =
 1
```

3.7 SCRIPT Script Files

3.7.1 Usage

A script is a sequence of FreeMat commands contained in a `.m` file. When the script is called (via the name of the file), the effect is the same as if the commands inside the script file were issued one at a time from the keyboard. Unlike `function` files (which have the same extension, but have a `function` declaration), script files share the same environment as their callers. Hence, assignments, etc, made inside a script are visible to the caller (which is not the case for functions.

3.7.2 Example

Here is an example of a script that makes some simple assignments and `printf` statements.

```
    tscript.m
a = 13;
printf('a is %d\n',a);
b = a + 32
```

If we execute the script and then look at the defined variables

```
--> tscript
a is 13

b =
 45

--> who
  Variable Name       Type   Flags             Size
              a    double                    [1x1]
            ans    double                    [0x0]
              b    double                    [1x1]
```

we see that `a` and `b` are defined appropriately.

3.8 SPECIAL Special Calling Syntax

3.8.1 Usage

To reduce the effort to call certain functions, FreeMat supports a special calling syntax for functions that take string arguments. In particular, the three following syntaxes are equivalent, with one caveat:

```
functionname('arg1','arg2',...,'argn')
```

or the parenthesis and commas can be removed

```
functionname 'arg1' 'arg2' ... 'argn'
```

The quotes are also optional (providing, of course, that the argument strings have no spaces in them)

```
functionname arg1 arg2 ... argn
```

This special syntax enables you to type `hold on` instead of the more cumbersome `hold('on')`. The caveat is that FreeMat currently only recognizes the special calling syntax as the first statement on a line of input. Thus, the following construction

```
for i=1:10; plot(vec(i)); hold on; end
```

would not work. This limitation may be removed in a future version.

3.8.2 Example

Here is a function that takes two string arguments and returns the concatenation of them.

```
    strcattest.m
function strcattest(str1,str2)
  str3 = [str1,str2];
  printf('str1 = %s, str2 = %s, str3 = %s\n',str1,str2,str3);
```

We call `strcattest` using all three syntaxes.

```
--> strcattest('hi','ho')
str1 = hi, str2 = ho, str3 = hiho
--> strcattest 'hi' 'ho'
str1 = hi, str2 = ho, str3 = hiho
--> strcattest hi ho
str1 = hi, str2 = ho, str3 = hiho
```

3.9 STR2FUNC String to Function conversion

3.9.1 Usage

The `str2func` function converts a function name into a function pointer. The syntax is

```
y = str2func('funcname')
```

where `funcname` is the name of the function. The return variable `y` is a function handle that points to the given function.

An alternate syntax is used to construct an anonymous function given an expression. They syntax is

```
y = str2func('anonymous def')
```

where `anonymous def` is an expression that defines an anonymous function, for example `'@(x) x.\^2'`.

3.9.2 Example

Here is a simple example of using `str2func`.

```
--> sin(.5)              % Calling the function directly

ans =
    0.4794

--> y = str2func('sin')  % Convert it into a function handle

y =
 @sin
--> y(.5)                % Calling 'sin' via the function handle

ans =
    0.4794
```

Here we use `str2func` to define an anonymous function

```
--> y = str2func('@(x) x.^2')

y =
 @(x)   x.^2
--> y(2)

ans =
 4
```

3.10 VARARGIN Variable Input Arguments

3.10.1 Usage

FreeMat functions can take a variable number of input arguments by setting the last argument in the argument list to `varargin`. This special keyword indicates that all arguments to the function (beyond the last non-`varargin` keyword) are assigned to a cell array named `varargin` available to the function. Variable argument functions are usually used when writing driver functions, i.e., functions that need to pass arguments to another function. The general syntax for a function that takes a variable number of arguments is

```
function [out_1,...,out_M] = fname(in_1,..,in_M,varargin)
```

Inside the function body, `varargin` collects the arguments to `fname` that are not assigned to the `in\_k`.

3.10.2 Example

Here is a simple wrapper to `feval` that demonstrates the use of variable arguments functions.

```
    wrapcall.m
function wrapcall(fname,varargin)
  feval(fname,varargin{:});
```

Now we show a call of the `wrapcall` function with a number of arguments

```
--> wrapcall('printf','%f...%f\n',pi,e)
3.141593...2.718282
```

A more serious driver routine could, for example, optimize a one dimensional function that takes a number of auxilliary parameters that are passed through `varargin`.

3.11 VARARGOUT Variable Output Arguments

3.11.1 Usage

FreeMat functions can return a variable number of output arguments by setting the last argument in the argument list to `varargout`. This special keyword indicates that the number of return values is variable. The general syntax for a function that returns a variable number of outputs is

```
  function [out_1,...,out_M,varargout] = fname(in_1,...,in_M)
```

The function is responsible for ensuring that `varargout` is a cell array that contains the values to assign to the outputs beyond `out\_M`. Generally, variable output functions use `nargout` to figure out how many outputs have been requested.

3.11.2 Example

This is a function that returns a varying number of values depending on the value of the argument.

```
    varoutfunc.m
function [varargout] = varoutfunc
  switch(nargout)
    case 1
      varargout = {'one of one'};
    case 2
      varargout = {'one of two','two of two'};
    case 3
      varargout = {'one of three','two of three','three of three'};
  end
```

Here are some examples of exercising `varoutfunc`:

```
--> [c1] = varoutfunc
c1 =
one of one
--> [c1,c2] = varoutfunc
c1 =
one of two
c2 =
two of two
--> [c1,c2,c3] = varoutfunc
c1 =
one of three
```

```
c2 =
two of three
c3 =
three of three
```

Chapter 4

Mathematical Operators

4.1 COLON Index Generation Operator

4.1.1 Usage

There are two distinct syntaxes for the colon `:` operator - the two argument form

```
y = a : c
```

and the three argument form

```
y = a : b : c
```

The two argument form is exactly equivalent to `a:1:c`. The output `y` is the vector

$$y = [a, a+b, a+2b, \ldots, a+nb]$$

where `a+nb <= c`. There is a third form of the colon operator, the no-argument form used in indexing (see `indexing` for more details).

4.1.2 Function Internals

The colon operator turns out to be trickier to implement than one might believe at first, primarily because the floating point versions should do the right thing, which is not the obvious behavior. For example, suppose the user issues a three point colon command

```
y = a : b : c
```

The first question that one might need to answer is: how many points in this vector? If you answered

$$n = \frac{c-a}{b} + 1$$

then you would be doing the straighforward, but not correct thing. because a, b, and c are all floating point values, there are errors associated with each of the quantities that can lead to n not being an integer. A better way (and the way FreeMat currently does the calculation) is to compute the bounding values (for b positive)

$$n \in \left[\frac{(c-a) \rightarrow 0}{b \rightarrow \infty}, \frac{(c-a) \rightarrow \infty}{b \rightarrow 0} \right] + 1$$

where

$$x \rightarrow y$$

means we replace x by the floating point number that is closest to it in the direction of y. Once we have determined the number of points we have to compute the intermediate values

$$[a, a+b, a+2*b, \ldots, a+n*b]$$

but one can readily verify for themselves that this may *not* be the same as the vector

$$\text{fliplr}[c, c-b, c-2*b, \ldots, c-n*b]$$

even for the case where

$$c = a + n*b$$

for some n. The reason is that the roundoff in the calculations may be different depending on the nature of the sum. FreeMat uses the following strategy to compute the double-colon vector:

1. The value `n` is computed by taking the floor of the larger value in the interval defined above.
2. If `n` falls inside the interval defined above, then it is assumed that the user intended `c = a + n*b`, and the symmetric algorithm is used. Otherwise, the nonsymmetric algorithm is used.
3. The symmetric algorithm computes the vector via

   $$[a, a+b, a+2b, \ldots, c-2b, c-b, c]$$

   working symmetrically from both ends of the vector (hence the nomenclature), while the nonsymmetric algorithm computes

   $$[a, a+b, a+2b, \ldots, a+nb]$$

   In practice, the entries are computed by repeated accumulation instead of multiplying the step size by an integer.
4. The real interval calculation is modified so that we get the exact same result with `a:b:c` and `c:-b:a` (which basically means that instead of moving towards infinity, we move towards the signed infinity where the sign is inherited from `b`).

If you think this is all very obscure, it is. But without it, you will be confronted by mysterious vectors where the last entry is dropped, or where the values show progressively larger amounts of accumulated roundoff error.

4.1.3 Examples

Some simple examples of index generation.

```
--> y = 1:4

y =
 1 2 3 4
```

Now by half-steps:

```
--> y = 1:.5:4

y =
    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    4.0000
```

Now going backwards (negative steps)

```
--> y = 4:-.5:1

y =
    4.0000    3.5000    3.0000    2.5000    2.0000    1.5000    1.0000
```

If the endpoints are the same, one point is generated, regardless of the step size (middle argument)

```
--> y = 4:1:4

y =
 4
```

If the endpoints define an empty interval, the output is an empty matrix:

```
--> y = 5:4

y =
  Empty array 1x0
```

4.2 COMPARISONOPS Array Comparison Operators

4.2.1 Usage

There are a total of six comparison operators available in FreeMat, all of which are binary operators with the following syntax

```
y = a < b
y = a <= b
y = a > b
y = a >= b
y = a ~= b
y = a == b
```

where `a` and `b` are numerical arrays or scalars, and `y` is a `logical` array of the appropriate size. Each of the operators has three modes of operation, summarized in the following list:

1. `a` is a scalar, `b` is an n-dimensional array - the output is then the same size as `b`, and contains the result of comparing each element in `b` to the scalar `a`.
2. `a` is an n-dimensional array, `b` is a scalar - the output is the same size as `a`, and contains the result of comparing each element in `a` to the scalar `b`.
3. `a` and `b` are both n-dimensional arrays of the same size - the output is then the same size as both `a` and `b`, and contains the result of an element-wise comparison between `a` and `b`.

The operators behave the same way as in `C`, with unequal types being promoted using the standard type promotion rules prior to comparisons. The only difference is that in FreeMat, the not-equals operator is `~=` instead of `!=`.

4.2.2 Examples

Some simple examples of comparison operations. First a comparison with a scalar:

```
--> a = randn(1,5)

a =
   -0.0454   -0.1876    1.5987   -0.9136   -0.2120

--> a>0

ans =
 0 0 1 0 0
```

Next, we construct two vectors, and test for equality:

```
--> a = [1,2,5,7,3]

a =
 1 2 5 7 3

--> b = [2,2,5,9,4]

b =
 2 2 5 9 4

--> c = a == b

c =
 0 1 1 0 0
```

4.3 DOTLEFTDIVIDE Element-wise Left-Division Operator

4.3.1 Usage

Divides two numerical arrays (elementwise) - gets its name from the fact that the divisor is on the left. There are two forms for its use, both with the same general syntax:

```
y = a .\ b
```

where `a` and `b` are `n`-dimensional arrays of numerical type. In the first case, the two arguments are the same size, in which case, the output `y` is the same size as the inputs, and is the element-wise division of `b` by `a`. In the second case, either `a` or `b` is a scalar, in which case `y` is the same size as the larger argument, and is the division of the scalar with each element of the other argument.

The rules for manipulating types has changed in FreeMat 4.0. See `typerules` for more details.

4.3.2 Function Internals

There are three formulae for the dot-left-divide operator, depending on the sizes of the three arguments. In the most general case, in which the two arguments are the same size, the output is computed via:

$$y(m_1,\ldots,m_d) = \frac{b(m_1,\ldots,m_d)}{a(m_1,\ldots,m_d)}$$

If `a` is a scalar, then the output is computed via

$$y(m_1,\ldots,m_d) = \frac{b(m_1,\ldots,m_d)}{a}$$

On the other hand, if `b` is a scalar, then the output is computed via

$$y(m_1,\ldots,m_d) = \frac{b}{a(m_1,\ldots,m_d)}.$$

4.3.3 Examples

Here are some examples of using the dot-left-divide operator. First, a straight-forward usage of the `.\\` operator. The first example is straightforward:

```
--> 3 .\ 8

ans =
    2.6667
```

We can also divide complex arguments:

```
--> a = 3 + 4*i

a =
   3.0000 +  4.0000i

--> b = 5 + 8*i

b =
   5.0000 +  8.0000i

--> c = b .\ a

c =
   0.5281 -  0.0449i
```

We can also demonstrate the three forms of the dot-left-divide operator. First the element-wise version:

```
--> a = [1,2;3,4]

a =
 1 2
 3 4

--> b = [2,3;6,7]

b =
 2 3
 6 7

--> c = a .\ b

c =
    2.0000    1.5000
    2.0000    1.7500
```

Then the scalar versions

```
--> c = a .\ 3

c =
    3.0000    1.5000
    1.0000    0.7500

--> c = 3 .\ a

c =
    0.3333    0.6667
    1.0000    1.3333
```

4.4 DOTPOWER Element-wise Power Operator

4.4.1 Usage

Raises one numerical array to another array (elementwise). There are three operators all with the same general syntax:

```
y = a .^ b
```

The result `y` depends on which of the following three situations applies to the arguments `a` and `b`:

1. `a` is a scalar, `b` is an arbitrary `n`-dimensional numerical array, in which case the output is `a` raised to the power of each element of `b`, and the output is the same size as `b`.

2. `a` is an `n`-dimensional numerical array, and `b` is a scalar, then the output is the same size as `a`, and is defined by each element of `a` raised to the power `b`.

3. `a` and `b` are both `n`-dimensional numerical arrays of *the same size*. In this case, each element of the output is the corresponding element of `a` raised to the power defined by the corresponding element of `b`.

The rules for manipulating types has changed in FreeMat 4.0. See `typerules` for more details.

4.4.2 Function Internals

There are three formulae for this operator. For the first form

$$y(m_1, \ldots, m_d) = a^{b(m_1,\ldots,m_d)},$$

and the second form

$$y(m_1, \ldots, m_d) = a(m_1, \ldots, m_d)^b,$$

and in the third form

$$y(m_1, \ldots, m_d) = a(m_1, \ldots, m_d)^{b(m_1,\ldots,m_d)}.$$

4.4.3 Examples

We demonstrate the three forms of the dot-power operator using some simple examples. First, the case of a scalar raised to a series of values.

```
--> a = 2

a =
 2

--> b = 1:4

b =
 1 2 3 4

--> c = a.^b

c =
  2  4  8 16
```

The second case shows a vector raised to a scalar.

```
--> c = b.^a

c =
  1  4  9 16
```

The third case shows the most general use of the dot-power operator.

```
--> A = [1,2;3,2]

A =
 1 2
 3 2

--> B = [2,1.5;0.5,0.6]

B =
    2.0000    1.5000
    0.5000    0.6000

--> C = A.^B

C =
    1.0000    2.8284
    1.7321    1.5157
```

4.5 DOTRIGHTDIVIDE Element-wise Right-Division Operator

4.5.1 Usage

Divides two numerical arrays (elementwise). There are two forms for its use, both with the same general syntax:

```
y = a ./ b
```

where `a` and `b` are `n`-dimensional arrays of numerical type. In the first case, the two arguments are the same size, in which case, the output `y` is the same size as the inputs, and is the element-wise division of `b` by `a`. In the second case, either `a` or `b` is a scalar, in which case `y` is the same size as the larger argument, and is the division of the scalar with each element of the other argument.

The rules for manipulating types has changed in FreeMat 4.0. See `typerules` for more details.

4.5.2 Function Internals

There are three formulae for the dot-right-divide operator, depending on the sizes of the three arguments. In the most general case, in which the two arguments are the same size, the output is computed via:

$$y(m_1,\ldots,m_d) = \frac{a(m_1,\ldots,m_d)}{b(m_1,\ldots,m_d)}$$

If `a` is a scalar, then the output is computed via

$$y(m_1,\ldots,m_d) = \frac{a}{b(m_1,\ldots,m_d)}$$

On the other hand, if `b` is a scalar, then the output is computed via

$$y(m_1,\ldots,m_d) = \frac{a(m_1,\ldots,m_d)}{b}.$$

4.5.3 Examples

Here are some examples of using the dot-right-divide operator. First, a straight-forward usage of the `./` operator. The first example is straightforward:

```
--> 3 ./ 8

ans =
    0.3750
```

We can also divide complex arguments:

```
--> a = 3 + 4*i

a =
   3.0000 +  4.0000i

--> b = 5 + 8*i

b =
   5.0000 +  8.0000i

--> c = a ./ b

c =
   0.5281 -  0.0449i
```

We can also demonstrate the three forms of the dot-right-divide operator. First the element-wise version:

```
--> a = [1,2;3,4]

a =
 1 2
 3 4

--> b = [2,3;6,7]

b =
 2 3
 6 7

--> c = a ./ b

c =
    0.5000    0.6667
    0.5000    0.5714
```

Then the scalar versions

```
--> c = a ./ 3

c =
    0.3333    0.6667
    1.0000    1.3333

--> c = 3 ./ a

c =
    3.0000    1.5000
    1.0000    0.7500
```

4.6 DOTTIMES Element-wise Multiplication Operator

4.6.1 Usage

Multiplies two numerical arrays (elementwise). There are two forms for its use, both with the same general syntax:

```
y = a .* b
```

where `a` and `b` are `n`-dimensional arrays of numerical type. In the first case, the two arguments are the same size, in which case, the output `y` is the same size as the inputs, and is the element-wise product of `a` and `b`. In the second case, either `a` or `b` is a scalar, in which case `y` is the same size as the larger argument, and is the product of the scalar with each element of the other argument.

The rules for manipulating types has changed in FreeMat 4.0. See `typerules` for more details.

4.6.2 Function Internals

There are three formulae for the dot-times operator, depending on the sizes of the three arguments. In the most general case, in which the two arguments are the same size, the output is computed via:

$$y(m_1, \ldots, m_d) = a(m_1, \ldots, m_d) \times b(m_1, \ldots, m_d)$$

If `a` is a scalar, then the output is computed via

$$y(m_1, \ldots, m_d) = a \times b(m_1, \ldots, m_d).$$

On the other hand, if `b` is a scalar, then the output is computed via

$$y(m_1, \ldots, m_d) = a(m_1, \ldots, m_d) \times b.$$

4.6.3 Examples

Here are some examples of using the dottimes operator. First, a straight-forward usage of the `.*` operator. The first example is straightforward:

```
--> 3 .* 8

ans =
 24
```

Next, we multiply a scalar by a vector of values:

```
--> 3.1 .* [2,4,5,6,7]

ans =
    6.2000   12.4000   15.5000   18.6000   21.7000
```

With complex values

```
--> a = 3 + 4*i

a =
   3.0000 +  4.0000i

--> b = a .* 2

b =
   6.0000 +  8.0000i
```

Finally, the element-wise version:

```
--> a = [1,2;3,4]

a =
 1 2
 3 4

--> b = [2,3;6,7]

b =
 2 3
 6 7

--> c = a .* b

c =
  2  6
 18 28
```

4.7 HERMITIAN Matrix Hermitian (Conjugate Transpose) Operator

4.7.1 Usage

Computes the Hermitian of the argument (a 2D matrix). The syntax for its use is

```
y = a';
```

where `a` is a `M x N` numerical matrix. The output `y` is a numerical matrix of the same type of size `N x M`. This operator is the conjugating transpose, which is different from the transpose operator `.'` (which does not conjugate complex values).

4.7.2 Function Internals

The Hermitian operator is defined simply as

$$y_{i,j} = \overline{a_{j,i}}$$

where `y\_ij` is the element in the `i`th row and `j`th column of the output matrix `y`.

4.7.3 Examples

A simple transpose example:

```
--> A = [1,2,0;4,1,-1]

A =
  1  2  0
  4  1 -1

--> A'

ans =
  1  4
  2  1
  0 -1
```

Here, we use a complex matrix to demonstrate how the Hermitian operator conjugates the entries.

```
--> A = [1+i,2-i]

A =
   1.0000 +  1.0000i   2.0000 -  1.0000i

--> A.'

ans =
   1.0000 +  1.0000i
   2.0000 -  1.0000i
```

4.8 LEFTDIVIDE Matrix Equation Solver/Divide Operator

4.8.1 Usage

The divide operator \ is really a combination of three operators, all of which have the same general syntax:

```
Y = A \ B
```

where `A` and `B` are arrays of numerical type. The result `Y` depends on which of the following three situations applies to the arguments `A` and `B`:

1. `A` is a scalar, `B` is an arbitrary `n`-dimensional numerical array, in which case the output is each element of `B` divided by the scalar `A`.
2. `A,B` are matrices with the same number of rows, i.e., `A` is of size `M x K`, and `B` is of size `M x L`, in which case the output is of size `K x L`.

The output follows the standard type promotion rules, although in the first two cases, if `A` and `B` are integers, the output is an integer also, while in the third case if `A` and `B` are integers, the output is of type `double`.

A few additional words about the third version, in which `A` and `B` are matrices. Very loosely speaking, `Y` is the matrix that satisfies `A * Y = B`. In cases where such a matrix exists. If such a matrix does not exist, then a matrix `Y` is returned that approximates `A * Y pprox B`.

4.8.2 Function Internals

There are three formulae for the times operator. For the first form

$$Y(m_1,\ldots,m_d) = \frac{B(m_1,\ldots,m_d)}{A}.$$

In the second form, the calculation of the output depends on the size of `A`. Because each column of `B` is treated independantly, we can rewrite the equation `A Y = B` as

$$A[y_1, y_2, \ldots, y_l] = [b_1, b_2, \ldots, b_l]$$

where `y\_i` are the columns of `Y`, and `b\_i` are the columns of the matrix `B`. If `A` is a square matrix, then the LAPACK routine `*gesvx` (where the `*` is replaced with `sdcz` depending on the type of the arguments) is used, which uses an LU decomposition of `A` to solve the sequence of equations sequentially. If `A` is singular, then a warning is emitted.

On the other hand, if `A` is rectangular, then the LAPACK routine `*gelsy` is used. Note that these routines are designed to work with matrices `A` that are full rank - either full column rank or full row rank. If `A` fails to satisfy this assumption, a warning is emitted. If `A` has full column rank (and thus necessarily has more rows than columns), then theoretically, this operator finds the columns `y\_i` that satisfy:

$$y_i = \arg\min_y \|Ay - b_i\|_2$$

and each column is thus the Least Squares solution of `A y = b\_i`. On the other hand, if `A` has full row rank (and thus necessarily has more columns than rows), then theoretically, this operator finds the columns `y\_i` that satisfy

$$y_i = \arg \min_{Ay=b_i} \|y\|_2$$

and each column is thus the Minimum Norm vector `y\_i` that satisfies `A y\_i = b\_i`. In the event that the matrix `A` is neither full row rank nor full column rank, a solution is returned, that is the minimum norm least squares solution. The solution is computed using an orthogonal factorization technique that is documented in the LAPACK User's Guide (see the References section for details).

4.8.3 Examples

Here are some simple examples of the divide operator. We start with a simple example of a full rank, square matrix:

```
--> A = [1,1;0,1]

A =
 1 1
 0 1
```

Suppose we wish to solve

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

(which by inspection has the solution `y\_1 = 1`, `y\_2 = 2`). Thus we compute:

```
--> B = [3;2]

B =
 3
 2

--> Y = A\B

Y =
 1
 2
```

Suppose we wish to solve a trivial Least Squares (LS) problem. We want to find a simple scaling of the vector `[1;1]` that is closest to the point `[2,1]`. This is equivalent to solving

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} y = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

in a least squares sense. For fun, we can calculate the solution using calculus by hand. The error we wish to minimize is

$$\varepsilon(y) = (y-2)^2 + (y-1)^2.$$

Taking a derivative with respect to `y`, and setting to zero (which we must have for an extrema when `y` is unconstrained)

$$2(y-2) + 2(y-1) = 0$$

which we can simplify to `4y = 6` or `y = 3/2` (we must, technically, check to make sure this is a minimum, and not a maximum or an inflection point). Here is the same calculation performed using FreeMat:

```
--> A = [1;1]

A =
 1
 1

--> B = [2;1]

B =
 2
 1

--> A\B

ans =
    1.5000
```

which is the same solution.

4.9 LOGICALOPS Logical Array Operators

4.9.1 Usage

There are three Boolean operators available in FreeMat. The syntax for their use is:

```
y = ~x
y = a & b
y = a | b
```

where `x`, `a` and `b` are `logical` arrays. The operators are

- NOT (`~`) - output `y` is true if the corresponding element of `x` is false, and ouput `y` is false if the corresponding element of `x` is true.
- OR (—) - output `y` is true if corresponding element of `a` is true or if corresponding element of `b` is true (or if both are true).
- AND (`\&`) - output `y` is true only if both the corresponding elements of `a` and `b` are both true.

The binary operators AND and OR can take scalar arguments as well as vector arguments, in which case, the scalar is operated on with each element of the vector. As of version 1.10, FreeMat supports `shortcut` evaluation. This means that if we have two expressions

```
if (expr1 & expr2)
```

then if `expr1` evaluates to `false`, then `expr2` is not evaluated at all. Similarly, for the expression

```
if (expr1 | expr2)
```

then if `expr1` evaluates to `true`, then `expr2` is not evaluated at all. Shortcut evaluation is useful for doing a sequence of tests, each of which is not valid unless the prior test is successful. For example,

```
if isa(p,'string') & strcmp(p,'fro')
```

is not valid without shortcut evaluation (if `p` is an integer, for example, the first test returns false, and an attempt to evaluate the second expression would lead to an error). Note that shortcut evaluation only works with scalar expressions.

4.9.2 Examples

Some simple examples of logical operators. Suppose we want to calculate the exclusive-or (XOR) of two vectors of logical variables. First, we create a pair of vectors to perform the XOR operation on:

```
--> a = (randn(1,6)>0)

a =
 0 0 0 0 1 0

--> b = (randn(1,6)>0)

b =
 1 1 0 1 0 1
```

Next, we can compute the OR of `a` and `b`:

```
--> c = a | b

c =
 1 1 0 1 1 1
```

However, the XOR and OR operations differ on the fifth entry - the XOR would be false, since it is true if and only if exactly one of the two inputs is true. To isolate this case, we can AND the two vectors, to find exactly those entries that appear as true in both `a` and `b`:

```
--> d = a & b

d =
 0 0 0 0 0 0
```

At this point, we can modify the contents of `c` in two ways – the Boolean way is to AND `\sim d` with `c`, like so

```
--> xor = c & (~d)

xor =
 1 1 0 1 1 1
```

The other way to do this is simply force `c(d) = 0`, which uses the logical indexing mode of FreeMat (see the chapter on indexing for more details). This, however, will cause `c` to become an `int32` type, as opposed to a logical type.

```
--> c(d) = 0

c =
 1 1 0 1 1 1
```

4.10 MATRIXPOWER Matrix Power Operator

4.10.1 Usage

The power operator for scalars and square matrices. This operator is really a combination of two operators, both of which have the same general syntax:

```
  y = a ^ b
```

The exact action taken by this operator, and the size and type of the output, depends on which of the two configurations of `a` and `b` is present:

1. a is a scalar, b is a square matrix
2. a is a square matrix, b is a scalar

4.10.2 Function Internals

In the first case that a is a scalar, and b is a square matrix, the matrix power is defined in terms of the eigenvalue decomposition of b. Let b have the following eigen-decomposition (problems arise with non-symmetric matrices b, so let us assume that b is symmetric):

$$b = E \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \lambda_n \end{bmatrix} E^{-1}$$

Then a raised to the power b is defined as

$$a^b = E \begin{bmatrix} a^{\lambda_1} & 0 & \cdots & 0 \\ 0 & a^{\lambda_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a^{\lambda_n} \end{bmatrix} E^{-1}$$

Similarly, if a is a square matrix, then a has the following eigen-decomposition (again, suppose a is symmetric):

$$a = E \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \lambda_n \end{bmatrix} E^{-1}$$

Then a raised to the power b is defined as

$$a^b = E \begin{bmatrix} \lambda_1^b & 0 & \cdots & 0 \\ 0 & \lambda_2^b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \lambda_n^b \end{bmatrix} E^{-1}$$

4.10.3 Examples

We first define a simple 2 x 2 symmetric matrix

```
--> A = 1.5

A =
    1.5000

--> B = [1,.2;.2,1]

B =
    1.0000    0.2000
    0.2000    1.0000
```

First, we raise B to the (scalar power) A:

```
--> C = B^A

C =
    1.0150    0.2995
    0.2995    1.0150
```

Next, we raise `A` to the matrix power `B`:

```
--> C = A^B

C =
    1.5049    0.1218
    0.1218    1.5049
```

4.11 MINUS Subtraction Operator

4.11.1 Usage

Subtracts two numerical arrays (elementwise). There are two forms for its use, both with the same general syntax:

```
y = a - b
```

where `a` and `b` are `n`-dimensional arrays of numerical type. In the first case, the two arguments are the same size, in which case, the output `y` is the same size as the inputs, and is the element-wise difference of `a` and `b`. In the second case, either `a` or `b` is a scalar, in which case `y` is the same size as the larger argument, and is the difference of the scalar to each element of the other argument.

The rules for manipulating types has changed in FreeMat 4.0. See `typerules` for more details.

4.11.2 Function Internals

There are three formulae for the subtraction operator, depending on the sizes of the three arguments. In the most general case, in which the two arguments are the same size, the output is computed via:

$$y(m_1, \ldots, m_d) = a(m_1, \ldots, m_d) - b(m_1, \ldots, m_d)$$

If `a` is a scalar, then the output is computed via

$$y(m_1, \ldots, m_d) = a - b(m_1, \ldots, m_d).$$

On the other hand, if `b` is a scalar, then the output is computed via

$$y(m_1, \ldots, m_d) = a(m_1, \ldots, m_d) - b.$$

4.11.3 Examples

Here are some examples of using the subtraction operator. First, a straight-forward usage of the minus operator. The first example is straightforward:

```
--> 3 - 8

ans =
 -5
```

Next, we subtract a vector of values from a scalar:

```
--> 3.1 - [2,4,5,6,7]

ans =
    1.1000   -0.9000   -1.9000   -2.9000   -3.9000
```

With complex values

```
--> a = 3 - 4*i

a =
   3.0000 -  4.0000i

--> b = a - 2

b =
   1.0000 -  4.0000i
```

Finally, the element-wise version:

```
--> a = [1,2;3,4]

a =
 1 2
 3 4

--> b = [2,3;6,7]

b =
 2 3
 6 7

--> c = a - b

c =
 -1 -1
 -3 -3
```

4.12 PLUS Addition Operator

4.12.1 Usage

Adds two numerical arrays (elementwise) together. There are two forms for its use, both with the same general syntax:

```
y = a + b
```

where `a` and `b` are `n`-dimensional arrays of numerical type. In the first case, the two arguments are the same size, in which case, the output `y` is the same size as the inputs, and is the element-wise the sum of `a` and `b`. In the second case, either `a` or `b` is a scalar, in which case `y` is the same size as the larger argument, and is the sum of the scalar added to each element of the other argument.

The rules for manipulating types has changed in FreeMat 4.0. See `typerules` for more details.

4.12.2 Function Internals

There are three formulae for the addition operator, depending on the sizes of the three arguments. In the most general case, in which the two arguments are the same size, the output is computed via:

$$y(m_1,\ldots,m_d) = a(m_1,\ldots,m_d) + b(m_1,\ldots,m_d)$$

If `a` is a scalar, then the output is computed via

$$y(m_1, \ldots, m_d) = a + b(m_1, \ldots, m_d).$$

On the other hand, if `b` is a scalar, then the output is computed via

$$y(m_1, \ldots, m_d) = a(m_1, \ldots, m_d) + b.$$

4.12.3 Examples

Here are some examples of using the addition operator. First, a straight-forward usage of the plus operator. The first example is straightforward:

```
--> 3 + 8

ans =
 11
```

Next, we add a scalar to a vector of values:

```
--> 3.1 + [2,4,5,6,7]

ans =
    5.1000    7.1000    8.1000    9.1000   10.1000
```

With complex values

```
--> a = 3 + 4*i

a =
   3.0000 +  4.0000i

--> b = a + 2

b =
   5.0000 +  4.0000i
```

Finally, the element-wise version:

```
--> a = [1,2;3,4]

a =
 1 2
 3 4

--> b = [2,3;6,7]

b =
 2 3
 6 7

--> c = a + b

c =
  3  5
  9 11
```

4.13 RIGHTDIVIDE Matrix Equation Solver/Divide Operator

4.13.1 Usage

The divide operator `/` is really a combination of three operators, all of which have the same general syntax:

```
Y = A / B
```

where `A` and `B` are arrays of numerical type. The result `Y` depends on which of the following three situations applies to the arguments `A` and `B`:

1. `A` is a scalar, `B` is an arbitrary `n`-dimensional numerical array, in which case the output is the scalar `A` divided into each element of `B`.

2. `B` is a scalar, `A` is an arbitrary `n`-dimensional numerical array, in which case the output is each element of `A` divided by the scalar `B`.

3. `A,B` are matrices with the same number of columns, i.e., `A` is of size `K x M`, and `B` is of size `L x M`, in which case the output is of size `K x L`.

The output follows the standard type promotion rules, although in the first two cases, if `A` and `B` are integers, the output is an integer also, while in the third case if `A` and `B` are integers, the output is of type `double`.

4.13.2 Function Internals

There are three formulae for the times operator. For the first form

$$Y(m_1,\ldots,m_d) = \frac{A}{B(m_1,\ldots,m_d)},$$

and the second form

$$Y(m_1,\ldots,m_d) = \frac{A(m_1,\ldots,m_d)}{B}.$$

In the third form, the output is defined as:

$$Y = (B'\backslash A')'$$

and is used in the equation `Y B = A`.

4.13.3 Examples

The right-divide operator is much less frequently used than the left-divide operator, but the concepts are similar. It can be used to find least-squares and minimum norm solutions. It can also be used to solve systems of equations in much the same way. Here's a simple example:

```
--> B = [1,1;0,1];
--> A = [4,5]

A =
 4 5

--> A/B

ans =
 4 1
```

4.14 TIMES Matrix Multiply Operator

4.14.1 Usage

Multiplies two numerical arrays. This operator is really a combination of three operators, all of which have the same general syntax:

```
y = a * b
```

where `a` and `b` are arrays of numerical type. The result `y` depends on which of the following three situations applies to the arguments `a` and `b`:

1. `a` is a scalar, `b` is an arbitrary `n`-dimensional numerical array, in which case the output is the element-wise product of `b` with the scalar `a`.

2. `b` is a scalar, `a` is an arbitrary `n`-dimensional numerical array, in which case the output is the element-wise product of `a` with the scalar `b`.

3. `a,b` are conformant matrices, i.e., `a` is of size `M x K`, and `b` is of size `K x N`, in which case the output is of size `M x N` and is the matrix product of `a`, and `b`.

Matrix multiplication is only defined for matrices of type `double` and `single`.

4.14.2 Function Internals

There are three formulae for the times operator. For the first form

$$y(m_1, \ldots, m_d) = a \times b(m_1, \ldots, m_d),$$

and the second form

$$y(m_1, \ldots, m_d) = a(m_1, \ldots, m_d) \times b.$$

In the third form, the output is the matrix product of the arguments

$$y(m, n) = \sum_{k=1}^{K} a(m, k) b(k, n)$$

4.14.3 Examples

Here are some examples of using the matrix multiplication operator. First, the scalar examples (types 1 and 2 from the list above):

```
--> a = [1,3,4;0,2,1]

a =
 1 3 4
 0 2 1

--> b = a * 2

b =
 2 6 8
 0 4 2
```

The matrix form, where the first argument is `2 x 3`, and the second argument is `3 x 1`, so that the product is size `2 x 1`.

```
--> a = [1,2,0;4,2,3]

a =
 1 2 0
 4 2 3

--> b = [5;3;1]

b =
 5
 3
 1

--> c = a*b

c =
 11
 29
```

Note that the output is double precision.

4.15 TRANSPOSE Matrix Transpose Operator

4.15.1 Usage

Performs a transpose of the argument (a 2D matrix). The syntax for its use is

```
  y = a.';
```

where `a` is a `M x N` numerical matrix. The output `y` is a numerical matrix of the same type of size `N x M`. This operator is the non-conjugating transpose, which is different from the Hermitian operator `'` (which conjugates complex values).

4.15.2 Function Internals

The transpose operator is defined simply as

$$y_{i,j} = a_{j,i}$$

where `y\_ij` is the element in the `i`th row and `j`th column of the output matrix `y`.

4.15.3 Examples

A simple transpose example:

```
--> A = [1,2,0;4,1,-1]

A =
  1  2  0
  4  1 -1

--> A.'

ans =
  1  4
  2  1
  0 -1
```

Here, we use a complex matrix to demonstrate how the transpose does *not* conjugate the entries.

```
--> A = [1+i,2-i]

A =
   1.0000 +  1.0000i   2.0000 -  1.0000i

--> A.'

ans =
   1.0000 +  1.0000i
   2.0000 -  1.0000i
```

4.16 TYPERULES Type Rules for Operators

4.16.1 Usage

Starting with FreeMat 4.0, the type of `y` is determined according to the same rules as Matlab. These are the rules:

1. Integer types of the same class can be combined. The answer is the same type as the inputs, and the operation is performed using saturating arithmetic. Integer types can also be combined with double precision values (again, the result is of the integer type).
2. Single precision floating point values can be combined with double precision, logical and character array classes. The result is of class single.
3. Double precision floating point values can be combined with all other types. Except as noted above, the output is of double precision.

These rules look strange, and they are. In general, computations are done in double precision in almost all cases. When single precision values are involved, the computations take place in single precision.

Chapter 5

Flow Control

5.1 BREAK Exit Execution In Loop

5.1.1 Usage

The `break` statement is used to exit a loop prematurely. It can be used inside a `for` loop or a `while` loop. The syntax for its use is

```
   break
```

inside the body of the loop. The `break` statement forces execution to exit the loop immediately.

5.1.2 Example

Here is a simple example of how `break` exits the loop. We have a loop that sums integers from `1` to `10`, but that stops prematurely at `5` using a `break`. We will use a `while` loop.

```
    break_ex.m
function accum = break_ex
  accum = 0;
  i = 1;
  while (i<=10)
    accum = accum + i;
    if (i == 5)
      break;
    end
    i = i + 1;
  end
```

The function is exercised here:

```
--> break_ex

ans =
 15

--> sum(1:5)

ans =
 15
```

5.2 CONTINUE Continue Execution In Loop

5.2.1 Usage

The `continue` statement is used to change the order of execution within a loop. The `continue` statement can be used inside a `for` loop or a `while` loop. The syntax for its use is

```
continue
```

inside the body of the loop. The `continue` statement forces execution to start at the top of the loop with the next iteration. The examples section shows how the `continue` statement works.

5.2.2 Example

Here is a simple example of using a `continue` statement. We want to sum the integers from `1` to `10`, but not the number `5`. We will use a `for` loop and a continue statement.

```
    continue_ex.m
function accum = continue_ex
  accum = 0;
  for i=1:10
    if (i==5)
      continue;
    end
    accum = accum + 1; %skipped if i == 5!
  end
```

The function is exercised here:

```
--> continue_ex

ans =
 9

--> sum([1:4,6:10])

ans =
 50
```

5.3 ERROR Causes an Error Condition Raised

5.3.1 Usage

The `error` function causes an error condition (exception to be raised). The general syntax for its use is

```
error(s),
```

where `s` is the string message describing the error. The `error` function is usually used in conjunction with `try` and `catch` to provide error handling. If the string `s`, then (to conform to the MATLAB API), `error` does nothing.

5.3.2 Example

Here is a simple example of an `error` being issued by a function `evenoddtest`:

```
    evenoddtest.m
function evenoddtest(n)
  if (n==0)
    error('zero is neither even nor odd');
  elseif ( n ~= fix(n) )
    error('expecting integer argument');
  end;
  if (n==int32(n/2)*2)
    printf('%d is even\n',n);
  else
    printf('%d is odd\n',n);
  end
```

The normal command line prompt `-->` simply prints the error that occured.

```
--> evenoddtest(4)
4 is even
--> evenoddtest(5)
5 is odd
--> evenoddtest(0)
In /home/sbasu/Devel/FreeMat/help/tmp/evenoddtest.m(evenoddtest) at line 3
    In scratch() at line 1
    In base(base)
    In base()
    In global()
Error: zero is neither even nor odd
--> evenoddtest(pi)
In /home/sbasu/Devel/FreeMat/help/tmp/evenoddtest.m(evenoddtest) at line 5
    In scratch() at line 1
    In base(base)
    In base()
    In global()
Error: expecting integer argument
```

5.4 FOR For Loop

5.4.1 Usage

The `for` loop executes a set of statements with an index variable looping through each element in a vector. The syntax of a `for` loop is one of the following:

```
  for (variable=expression)
     statements
  end
```

Alternately, the parenthesis can be eliminated

```
  for variable=expression
     statements
  end
```

or alternately, the index variable can be pre-initialized with the vector of values it is going to take:

```
  for variable
     statements
  end
```

The third form is essentially equivalent to `for variable=variable`, where `variable` is both the index variable and the set of values over which the for loop executes. See the examples section for an example of this form of the `for` loop.

5.4.2 Examples

Here we write `for` loops to add all the integers from `1` to `100`. We will use all three forms of the `for` statement.

```
--> accum = 0;
--> for (i=1:100); accum = accum + i; end
--> accum

ans =
 5050
```

The second form is functionally the same, without the extra parenthesis

```
--> accum = 0;
--> for i=1:100; accum = accum + i; end
--> accum

ans =
 5050
```

In the third example, we pre-initialize the loop variable with the values it is to take

5.5 IF-ELSEIF-ELSE Conditional Statements

5.5.1 Usage

The `if` and `else` statements form a control structure for conditional execution. The general syntax involves an `if` test, followed by zero or more `elseif` clauses, and finally an optional `else` clause:

```
  if conditional_expression_1
    statements_1
  elseif conditional_expression_2
    statements_2
  elseif conditional_expresiion_3
    statements_3
  ...
  else
    statements_N
  end
```

Note that a conditional expression is considered true if the real part of the result of the expression contains any non-zero elements (this strange convention is adopted for compatibility with MATLAB).

5.5.2 Examples

Here is an example of a function that uses an `if` statement

```
    if_test.m
function c = if_test(a)
  if (a == 1)
```

```
    c = 'one';
  elseif (a==2)
    c = 'two';
  elseif (a==3)
    c = 'three';
  else
    c = 'something else';
  end
```

Some examples of `if\_test` in action:

```
--> if_test(1)

ans =
one
--> if_test(2)

ans =
two
--> if_test(3)

ans =
three
--> if_test(pi)

ans =
something else
```

5.6 KEYBOARD Initiate Interactive Debug Session

5.6.1 Usage

The `keyboard` statement is used to initiate an interactive session at a specific point in a function. The general syntax for the `keyboard` statement is

```
keyboard
```

A `keyboard` statement can be issued in a `script`, in a `function`, or from within another `keyboard` session. The result of a `keyboard` statement is that execution of the program is halted, and you are given a prompt of the form:

```
[scope,n] -->
```

where `scope` is the current scope of execution (either the name of the function we are executing, or `base` otherwise). And `n` is the depth of the `keyboard` session. If, for example, we are in a `keyboard` session, and we call a function that issues another `keyboard` session, the depth of that second session will be one higher. Put another way, `n` is the number of `return` statements you have to issue to get back to the base workspace. Incidentally, a `return` is how you exit the `keyboard` session and resume execution of the program from where it left off. A `retall` can be used to shortcut execution and return to the base workspace.

The `keyboard` statement is an excellent tool for debugging FreeMat code, and along with `eval` provide a unique set of capabilities not usually found in compiled environments. Indeed, the `keyboard` statement is equivalent to a debugger breakpoint in more traditional environments, but with significantly more inspection power.

5.6.2 Example

Here we demonstrate a two-level `keyboard` situation. We have a simple function that calls `keyboard` internally:

```
    key_one.m
function c = key_one(a,b)
c = a + b;
keyboard
```

Now, we execute the function from the base workspace, and at the `keyboard` prompt, we call it again. This action puts us at depth 2. We can confirm that we are in the second invocation of the function by examining the arguments. We then issue two `return` statements to return to the base workspace.

```
--> key_one(1,2)
[key_one,3]--> key_one(5,7)
[key_one,3]--> a

ans =
 5

[key_one,3]--> b

ans =
 7

[key_one,3]--> c

ans =
 12

[key_one,3]--> return

ans =
 12

[key_one,3]--> a

ans =
 1

[key_one,3]--> b

ans =
 2

[key_one,3]--> c

ans =
 3

[key_one,3]--> return

ans =
 3
```

5.7 LASTERR Retrieve Last Error Message

5.7.1 Usage

Either returns or sets the last error message. The general syntax for its use is either

```
msg = lasterr
```

which returns the last error message that occured, or

```
lasterr(msg)
```

which sets the contents of the last error message.

5.7.2 Example

Here is an example of using the `error` function to set the last error, and then retrieving it using lasterr.

```
--> try; error('Test error message'); catch; end;
--> lasterr

ans =
Test error message
```

Or equivalently, using the second form:

```
--> lasterr('Test message');
--> lasterr

ans =
Test message
```

5.8 RETALL Return From All Keyboard Sessions

5.8.1 Usage

The `retall` statement is used to return to the base workspace from a nested `keyboard` session. It is equivalent to forcing execution to return to the main prompt, regardless of the level of nesting of `keyboard` sessions, or which functions are running. The syntax is simple

```
retall
```

The `retall` is a convenient way to stop debugging. In the process of debugging a complex program or set of functions, you may find yourself 5 function calls down into the program only to discover the problem. After fixing it, issueing a `retall` effectively forces FreeMat to exit your program and return to the interactive prompt.

5.8.2 Example

Here we demonstrate an extreme example of `retall`. We are debugging a recursive function `self` to calculate the sum of the first N integers. When the function is called, a `keyboard` session is initiated after the function has called itself N times. At this `keyboard` prompt, we issue another call to `self` and get another `keyboard` prompt, this time with a depth of 2. A `retall` statement returns us to the top level without executing the remainder of either the first or second call to `self`:

```
    self.m
function y = self(n)
  if (n>1)
    y = n + self(n-1);
    printf('y is %d\n',y);
  else
    y = 1;
    printf('y is initialized to one\n');
    keyboard
  end
```

```
--> self(4)
y is initialized to one
[self,8]--> self(6)
y is initialized to one
[self,8]--> retall
```

5.9 RETURN Return From Function

5.9.1 Usage

The `return` statement is used to immediately return from a function, or to return from a `keyboard` session. The syntax for its use is

```
  return
```

Inside a function, a `return` statement causes FreeMat to exit the function immediately. When a `keyboard` session is active, the `return` statement causes execution to resume where the `keyboard` session started.

5.9.2 Example

In the first example, we define a function that uses a `return` to exit the function if a certain test condition is satisfied.

```
    return_func.m
function ret = return_func(a,b)
  ret = 'a is greater';
  if (a > b)
    return;
  end
  ret = 'b is greater';
  printf('finishing up...\n');
```

Next we exercise the function with a few simple test cases:

```
--> return_func(1,3)
finishing up...

ans =
b is greater
--> return_func(5,2)

ans =
a is greater
```

In the second example, we take the function and rewrite it to use a `keyboard` statement inside the `if` statement.

```
    return_func2.m
function ret = return_func2(a,b)
  if (a > b)
     ret = 'a is greater';
     keyboard;
  else
     ret = 'b is greater';
  end
  printf('finishing up...\n');
```

Now, we call the function with a larger first argument, which triggers the `keyboard` session. After verifying a few values inside the `keyboard` session, we issue a `return` statement to resume execution.

```
--> return_func2(2,4)
finishing up...

ans =
b is greater
--> return_func2(5,1)
[return_func2,4]--> ret

ans =
a is greater
[return_func2,4]--> a

ans =
 5

[return_func2,4]--> b

ans =
 1

[return_func2,4]--> return
finishing up...

ans =
a is greater
```

5.10 SWITCH Switch statement

5.10.1 Usage

The `switch` statement is used to selective execute code based on the value of either scalar value or a string. The general syntax for a `switch` statement is

```
  switch(expression)
    case test_expression_1
      statements
    case test_expression_2
      statements
    otherwise
      statements
  end
```

The `otherwise` clause is optional. Note that each test expression can either be a scalar value, a string to test against (if the switch expression is a string), or a `cell-array` of expressions to test against. Note that unlike `C` `switch` statements, the FreeMat `switch` does not have fall-through, meaning that the statements associated with the first matching case are executed, and then the `switch` ends. Also, if the `switch` expression matches multiple `case` expressions, only the first one is executed.

5.10.2 Examples

Here is an example of a `switch` expression that tests against a string input:

```
    switch_test.m
function c = switch_test(a)
  switch(a)
    case {'lima beans','root beer'}
      c = 'food';
    case {'red','green','blue'}
      c = 'color';
    otherwise
      c = 'not sure';
  end
```

Now we exercise the switch statements

```
--> switch_test('root beer')

ans =
food
--> switch_test('red')

ans =
color
--> switch_test('carpet')

ans =
not sure
```

5.11 TRY-CATCH Try and Catch Statement

5.11.1 Usage

The `try` and `catch` statements are used for error handling and control. A concept present in `C++`, the `try` and `catch` statements are used with two statement blocks as follows

```
   try
     statements_1
   catch
     statements_2
   end
```

The meaning of this construction is: try to execute `statements\_1`, and if any errors occur during the execution, then execute the code in `statements\_2`. An error can either be a FreeMat generated error (such as a syntax error in the use of a built in function), or an error raised with the `error` command.

5.11.2 Examples

Here is an example of a function that uses error control via `try` and `catch` to check for failures in `fopen`.

```
    read_file.m
function c = read_file(filename)
try
   fp = fopen(filename,'r');
   c = fgetline(fp);
   fclose(fp);
catch
   c = ['could not open file because of error :' lasterr]
end
```

Now we try it on an example file - first one that does not exist, and then on one that we create (so that we know it exists).

```
--> read_file('this_filename_is_invalid')

c =
could not open file because of error :Invalid handle!

ans =
could not open file because of error :Invalid handle!
--> fp = fopen('test_text.txt','w');
--> fprintf(fp,'a line of text\n');
--> fclose(fp);
--> read_file('test_text.txt')

ans =
a line of text
```

5.12 WARNING Emits a Warning Message

5.12.1 Usage

The `warning` function causes a warning message to be sent to the user. The general syntax for its use is

```
warning(s)
```

where `s` is the string message containing the warning.

The `warning` function can also be used to turn off warnings, and to retrieve the current state of the warning flag. To turn off warnings use the syntax

```
warning off
```

at which point, warnings will not be displayed. To turn on warnings use the syntax

```
warning on
```

In both cases, you can also retrieve the current state of the warnings flag

```
y = warning('on')
y = warning('off')
```

5.13 WHILE While Loop

5.13.1 Usage

The `while` loop executes a set of statements as long as a the test condition remains `true`. The syntax of a `while` loop is

```
  while test_expression
     statements
  end
```

Note that a conditional expression is considered true if the real part of the result of the expression contains any non-zero elements (this strange convention is adopted for compatibility with MATLAB).

5.13.2 Examples

Here is a `while` loop that adds the integers from `1` to `100`:

```
--> accum = 0;
--> k=1;
--> while (k<=100), accum = accum + k; k = k + 1; end
--> accum

ans =
 5050
```

Chapter 6

FreeMat Functions

6.1 ADDPATH Add

6.1.1 Usage

The `addpath` routine adds a set of directories to the current path. The first form takes a single directory and adds it to the beginning or top of the path:

```
addpath('directory')
```

The second form add several directories to the top of the path:

```
addpath('dir1','dir2',...,'dirn')
```

Finally, you can provide a flag to control where the directories get added to the path

```
addpath('dir1','dir2',...,'dirn','-flag')
```

where if `flag` is either `'-0'` or `'-begin'`, the directories are added to the top of the path, and if the `flag` is either `'-1'` or `'-end'` the directories are added to the bottom (or end) of the path.

6.2 ASSIGNIN Assign Variable in Workspace

6.2.1 Usage

The `assignin` function allows you to assign a value to a variable in either the callers work space or the base work space. The syntax for `assignin` is

```
assignin(workspace,variablename,value)
```

The argument `workspace` must be either 'caller' or 'base'. If it is 'caller' then the variable is assigned in the caller's work space. That does not mean the caller of `assignin`, but the caller of the current function or script. On the other hand if the argument is 'base', then the assignment is done in the base work space. Note that the variable is created if it does not already exist.

6.3 BLASLIB Select BLAS library

6.3.1 Usage

The `blaslib` function allows you to select the FreeMat blas library. It has two modes of operation. The first is to select blas library:

```
blaslib LIB_NAME
```

If you want to see current blas library selected issue a `blaslib` command with no arguments.

```
blaslib
```

the returned list of libraries will include an asterix next to the library currently selected. If no library is selected, FreeMat will use its internal (reference) BLAS implementation, which is slow, but portable.

Location of the optimized library is specified in blas.ini file which is installed in the binary directory (i.e. directory where FreeMat executable is located).

The format of blas.ini file is following:

```
 [Linux64]
ATLAS64\libname=/usr/lib64/atlas/libblas.so
ATLAS64\capfnames=false
ATLAS64\prefix=
ATLAS64\suffix=_
ATLAS64\desc="ATLAS BLAS. Optimized."
```

Where Linux64 is the OS flavor for the blas library described below it. Other options are [Win32], [Linux32], [OSX]. Note that Linux is our name for all unix platforms.

- `ATLAS64` - name of the library as it will appear in the list when you type blaslib command in FreeMat.
- `ATLAS64bname` - path to the library. It has to be a shared library (Linux), DLL (Windows), Bundle (? OSX). This library has to be a Fortran BLAS library, not cblas!
- `ATLAS64\capfnames` - does the library use capital letters for function names (usually false).
- `ATLAS64refix` - prefix (characters that are put in front of) for all blas functions in the library (e.g. `ATL\_` or `AMD\_`).
- `ATLAS64\suffix` - suffix (characters that are put after) for all blas function in the library (e.g. `\_`)
- `ATLAS64\desc` - text description of the library.

On FreeMat startup it looks at the blas.ini file, and tries to load each library described in the section for the given OS flavor. FreeMat will use the first library it can successfully load. If you want to switch the BLAS libraries dynamically in the running FreeMat session you need to use blaslib command.

If FreeMat can't load any library it will default to using built in BLAS.

You should be a careful when using non-default BLAS libraries. Some libraries do not implement all the BLAS functions correctly. You should run FreeMat test suite (type `run\_tests()`) and use common sense when evaluating the results of numerical computations.

6.4 BUILTIN Evaulate Builtin Function

6.4.1 Usage

The `builtin` function evaluates a built in function with the given name, bypassing any overloaded functions. The syntax of `builtin` is

```
[y1,y2,...,yn] = builtin(fname,x1,x2,...,xm)
```

where `fname` is the name of the function to call. Apart from the fact that `fname` must be a string, and that `builtin` always calls the non-overloaded method, it operates exactly like `feval`. Note that unlike MATLAB, `builtin` does not force evaluation to an actual compiled function. It simply subverts the activation of overloaded method calls.

6.5 CLC Clear Dislplay

6.5.1 Usage

The `clc` function clears the current display. The syntax for its use is

```
clc
```

6.6 CLOCK Get Current Time

6.6.1 Usage

Returns the current date and time as a vector. The syntax for its use is

```
y = clock
```

where `y` has the following format:

```
y = [year month day hour minute seconds]
```

6.6.2 Example

Here is the time that this manual was last built:

```
--> clock

ans =

   1.0e+03 *
    2.0110    0.0070    0.0100    0.0200    0.0110    0.0457
```

6.7 CLOCKTOTIME Convert Clock Vector to Epoch Time

6.7.1 Usage

Given the output of the `clock` command, this function computes the epoch time, i.e, the time in seconds since January 1,1970 at 00:00:00 UTC. This function is most useful for calculating elapsed times using the clock, and should be accurate to less than a millisecond (although the true accuracy depends on accuracy of the argument vector). The usage for `clocktotime` is

```
y = clocktotime(x)
```

where `x` must be in the form of the output of `clock`, that is

```
x = [year month day hour minute seconds]
```

6.7.2 Example

Here is an example of using `clocktotime` to time a delay of 1 second

```
--> x = clock

x =

   1.0e+03 *
    2.0110    0.0070    0.0100    0.0200    0.0110    0.0457

--> sleep(1)
```

```
--> y = clock

y =

   1.0e+03 *
    2.0110    0.0070    0.0100    0.0200    0.0110    0.0467

--> clocktotime(y) - clocktotime(x)

ans =
 1
```

6.8 COMPUTER Computer System FreeMat is Running On

6.8.1 Usage

Returns a string describing the name of the system FreeMat is running on. The exact value of this string is subject to change, although the `'MAC'` and `'PCWIN'` values are probably fixed.

```
str = computer
```

Currently, the following return values are defined

- `'PCWIN'` - MS Windows
- `'MAC'` - Mac OS X
- `'UNIX'` - All others

6.9 DIARY Create a Log File of Console

6.9.1 Usage

The `diary` function controls the creation of a log file that duplicates the text that would normally appear on the console. The simplest syntax for the command is simply:

```
diary
```

which toggles the current state of the diary command. You can also explicitly set the state of the diary command via the syntax

```
diary off
```

or

```
diary on
```

To specify a filename for the log (other than the default of `diary`), you can use the form:

```
diary filename
```

or

```
diary('filename')
```

which activates the diary with an output filename of `filename`. Note that the `diary` command is thread specific, but that the output is appended to a given file. That means that if you call `diary` with the same filename on multiple threads, their outputs will be intermingled in the log file (just as on the console). Because the `diary` state is tied to individual threads, you cannot retrieve the current diary state using the `get(0,'Diary')` syntax from MATLAB. Instead, you must call the `diary` function with no inputs and one output:

```
state = diary
```

which returns a logical `1` if the output of the current thread is currently going to a diary, and a logical `0` if not.

6.10 DOCLI Start a Command Line Interface

6.10.1 Usage

The `docli` function is the main function that you interact with when you run FreeMat. I am not sure why you would want to use it, but hey - its there if you want to use it.

6.11 EDIT Open Editor Window

6.11.1 Usage

Brings up the editor window. The arguments of `edit` function are names of files for editing:

```
edit file1 file2 file3
```

6.12 EDITOR Open Editor Window

6.12.1 Usage

Brings up the editor window. The `editor` function takes no arguments:

```
editor
```

6.13 ERRORCOUNT Retrieve the Error Counter for the Interpreter

6.13.1 Usage

This routine retrieves the internal counter for the interpreter, and resets it to zero. The general syntax for its use is

```
count = errorcount
```

6.14 ETIME Elapsed Time Function

6.14.1 Usage

The `etime` calculates the elapsed time between two `clock` vectors `x1` and `x2`. The syntax for its use is

```
y = etime(x1,x2)
```

where `x1` and `x2` are in the `clock` output format

```
x = [year month day hour minute seconds]
```

6.14.2 Example

Here we use `etime` as a substitute for `tic` and `toc`

```
--> x1 = clock;
--> sleep(1);
--> x2 = clock;
--> etime(x2,x1);
```

6.15 EVAL Evaluate a String

6.15.1 Usage

The `eval` function evaluates a string. The general syntax for its use is

```
eval(s)
```

where `s` is the string to evaluate. If `s` is an expression (instead of a set of statements), you can assign the output of the `eval` call to one or more variables, via

```
x = eval(s)
[x,y,z] = eval(s)
```

Another form of `eval` allows you to specify an expression or set of statements to execute if an error occurs. In this form, the syntax for `eval` is

```
eval(try_clause,catch_clause),
```

or with return values,

```
x = eval(try_clause,catch_clause)
[x,y,z] = eval(try_clause,catch_clause)
```

These later forms are useful for specifying defaults. Note that both the `try\_clause` and `catch\_clause` must be expressions, as the equivalent code is

```
try
  [x,y,z] = try_clause
catch
  [x,y,z] = catch_clause
end
```

so that the assignment must make sense in both cases.

6.15.2 Example

Here are some examples of `eval` being used.

```
--> eval('a = 32')

a =
 32

--> b = eval('a')

b =
 32
```

The primary use of the `eval` statement is to enable construction of expressions at run time.

```
--> s = ['b = a' ' + 2']

s =
b = a + 2
--> eval(s)

b =
 34
```

Here we demonstrate the use of the catch-clause to provide a default value

```
--> a = 32

a =
 32

--> b = eval('a','1')

b =
 32

--> b = eval('z','a+1')

b =
 33
```

Note that in the second case, `b` takes the value of 33, indicating that the evaluation of the first expression failed (because `z` is not defined).

6.16 EVALIN Evaluate a String in Workspace

6.16.1 Usage

The `evalin` function is similar to the `eval` function, with an additional argument up front that indicates the workspace that the expressions are to be evaluated in. The various syntaxes for `evalin` are:

```
evalin(workspace,expression)
x = evalin(workspace,expression)
[x,y,z] = evalin(workspace,expression)
evalin(workspace,try_clause,catch_clause)
x = evalin(workspace,try_clause,catch_clause)
[x,y,z] = evalin(workspace,try_clause,catch_clause)
```

The argument `workspace` must be either 'caller' or 'base'. If it is 'caller', then the expression is evaluated in the caller's work space. That does not mean the caller of `evalin`, but the caller of the current function or script. On the other hand if the argument is 'base', then the expression is evaluated in the base work space. See `eval` for details on the use of each variation.

6.17 EXIT Exit Program

6.17.1 Usage

The usage is

```
exit
```

Quits FreeMat. This script is a simple synonym for `quit`.

6.18 FEVAL Evaluate a Function

6.18.1 Usage

The `feval` function executes a function using its name. The syntax of `feval` is

```
[y1,y2,...,yn] = feval(f,x1,x2,...,xm)
```

where `f` is the name of the function to evaluate, and `xi` are the arguments to the function, and `yi` are the return values.

Alternately, `f` can be a function handle to a function (see the section on `function handles` for more information).

Finally, FreeMat also supports `f` being a user defined class in which case it will atttempt to invoke the `subsref` method of the class.

6.18.2 Example

Here is an example of using `feval` to call the `cos` function indirectly.

```
--> feval('cos',pi/4)

ans =
    0.7071
```

Now, we call it through a function handle

```
--> c = @cos

c =
 @cos
--> feval(c,pi/4)

ans =
    0.7071
```

Here we construct an inline object (which is a user-defined class) and use `feval` to call it

```
--> afunc = inline('cos(t)+sin(t)','t')

afunc =
  inline function object
  f(t) = cos(t)+sin(t)
--> feval(afunc,pi)

ans =
   -1.0000

--> afunc(pi)

ans =
   -1.0000
```

In both cases, (the `feval` call and the direct invokation), FreeMat calls the `subsref` method of the class, which computes the requested function.

6.19 FILESEP Directory Separation Character

6.19.1 Usage

The `filesep` routine returns the character used to separate directory names on the current platform (basically, a forward slash for Windows, and a backward slash for all other OSes). The syntax is simple:

```
  x = filesep
```

6.20 HELP Help

6.20.1 Usage

Displays help on a function available in FreeMat. The help function takes one argument:

```
help topic
```

where `topic` is the topic to look for help on. For scripts, the result of running `help` is the contents of the comments at the top of the file. If FreeMat finds no comments, then it simply displays the function declaration.

6.21 HELPWIN Online Help Window

6.21.1 Usage

Brings up the online help window with the FreeMat manual. The `helpwin` function takes no arguments:

```
helpwin
helpwin FunctionName
```

6.22 JITCONTROL Control the Just In Time Compiler

6.22.1 Usage

The `jitcontrol` functionality in FreeMat allows you to control the use of the Just In Time (JIT) compiler. Starting in FreeMat version 4, the JIT compiler is enabled by default on all platforms where it is successfully built. The JIT compiler should significantly improve the performance of loop intensive, scalar code. As development progresses, more and more functionality will be enabled under the JIT. In the mean time (if you use the GUI version of FreeMat) you can use the JIT chat window to get information on why your code was JIT compiled (or not).

6.23 MFILENAME Name of Current Function

6.23.1 Usage

Returns a string describing the name of the current function. For M-files this string will be the complete filename of the function. This is true even for subfunctions. The syntax for its use is

```
y = mfilename
```

6.24 PATH Get or Set FreeMat Path

6.24.1 Usage

The `path` routine has one of the following syntaxes. In the first form

```
x = path
```

`path` simply returns the current path. In the second, the current path is replaced by the argument string `'thepath'`

```
path('thepath')
```

In the third form, a new path is appended to the current search path

```
path(path,'newpath')
```

In the fourth form, a new path is prepended to the current search path

```
path('newpath',path)
```

In the final form, the path command prints out the current path

```
path
```

6.25 PATHSEP Path Directories Separation Character

6.25.1 Usage

The `pathsep` routine returns the character used to separate multiple directories on a path string for the current platform (basically, a semicolon for Windows, and a regular colon for all other OSes). The syntax is simple:

```
x = pathsep
```

6.26 PATHTOOL Open Path Setting Tool

6.26.1 Usage

Brings up the pathtool dialog. The `pathtool` function takes no arguments:

```
pathtool
```

6.27 PCODE Convert a Script or Function to P-Code

6.27.1 Usage

Writes out a script or function as a P-code function. The general syntax for its use is:

```
pcode fun1 fun2 ...
```

The compiled functions are written to the current directory.

6.28 PROFILER Control Profiling

6.28.1 Usage

The `profile` function allows you to control the FreeMat profiler. It has two modes of operation. The first is to enable-disable the profiler. To turn on profiling:

```
profiler on
```

to turn off profiling, use

```
profiler off
```

Note that regardless of the state of the profiler, only functions and scripts are profiled. Commands entered on the command line are not profiled. To see information that has accumulated in a profile, you use the variant of the command:

```
profiler list
```

which lists current sorted profiling resuls. You can use this form to obtain profiler results as a cell array

```
r=profiler('list')
```

If you want to see current profile status issue a `profile` command with no arguments.

```
profiler
```

6.29 QUIET Control the Verbosity of the Interpreter

6.29.1 Usage

The `quiet` function controls how verbose the interpreter is when executing code. The syntax for the function is

```
quiet flag
```

where `flag` is one of

- `'normal'` - normal output from the interpreter
- `'quiet'` - only intentional output (e.g. `printf` calls and `disp` calls) is printed. The output of expressions that are not terminated in semicolons are not printed.
- `'silent'` - nothing is printed to the output.

The `quiet` command also returns the current quiet flag.

6.30 QUIT Quit Program

6.30.1 Usage

The `quit` statement is used to immediately exit the FreeMat application. The syntax for its use is

```
quit
```

6.31 REHASH Rehash Directory Caches

6.31.1 Usage

Usually, FreeMat will automatically determine when M Files have changed, and pick up changes you have made to M files. Sometimes, you have to force a refresh. Use the `rehash` command for this purpose. The syntax for its use is

```
rehash
```

6.32 RESCAN Rescan M Files for Changes

6.32.1 Usage

Usually, FreeMat will automatically determine when M Files have changed, and pick up changes you have made to M files. Sometimes, you have to force a refresh. Use the `rescan` command for this purpose. The syntax for its use is

```
rescan
```

6.33 ROOTPATH Set FreeMat Root Path

6.33.1 Usage

In order to function properly, FreeMat needs to know where to find the `toolbox` directory as well as the `help` directory. These directories are located on what is known as the `root path`. Normally, FreeMat should know where these directories are located. However under some circumstances (usually when FreeMat is installed into a non-default location), it may be necessary to indicate a different root path location, or to specify a particular one. Note that on the Mac OS platform, FreeMat is installed as a bundle, and will use the toolbox

that is installed in the bundle regardless of the setting for `rootpath`. For Linux, FreeMat will typically use `/usr/local/share/FreeMat-<Version>/` for the root path. Installations from source code will generally work, but binary installations (e.g., from an `RPM`) may need to have the rootpath set.

The `rootpath` function has two forms. The first form takes no arguments and returns the current root path

```
rootpath
```

The second form will set a rootpath directly from the command line

```
rootpath(path)
```

where `path` is the full path to where the `toolbox` and `help` directories are located. For example, `rootpath('/usr/share/Freel`
The third form enables the GUI form

```
rootpath gui
```

which activates a dialog box to pick a directory that is the root directory of the FreeMat installation (e.g., where `help` and `toolbox` are located. Changes to `rootpath` are persistent (you do not need to run it every time you start FreeMat).

6.34 SIMKEYS Simulate Keypresses from the User

6.34.1 Usage

This routine simulates keystrokes from the user on FreeMat. The general syntax for its use is

```
otext = simkeys(text)
```

where `text` is a string to simulate as input to the console. The output of the commands are captured and returned in the string `otext`. This is primarily used by the testing infrastructure.

6.35 SLEEP Sleep For Specified Number of Seconds

6.35.1 Usage

Suspends execution of FreeMat for the specified number of seconds. The general syntax for its use is

```
sleep(n),
```

where `n` is the number of seconds to wait.

6.36 SOURCE Execute an Arbitrary File

6.36.1 Usage

The `source` function executes the contents of the given filename one line at a time (as if it had been typed at the `-->` prompt). The `source` function syntax is

```
source(filename)
```

where `filename` is a `string` containing the name of the file to process.

6.36.2 Example

First, we write some commands to a file (note that it does not end in the usual `.m` extension):

```
    source_test
a = 32;
b = a;
printf('a is %d and b is %d\n',a,b);
```

Now we source the resulting file.

```
--> clear a b
--> source source_test
a is 32 and b is 32
```

6.37 STARTUP Startup Script

6.37.1 Usage

Upon starting, FreeMat searches for a script names `startup.m`, and if it finds it, it executes it. This script can be in the current directory, or on the FreeMat path (set using `setpath`). The contents of startup.m must be a valid script (not a function).

6.38 TIC Start Stopwatch Timer

6.38.1 Usage

Starts the stopwatch timer, which can be used to time tasks in FreeMat. The `tic` takes no arguments, and returns no outputs. You must use `toc` to get the elapsed time. The usage is

```
  tic
```

6.38.2 Example

Here is an example of timing the solution of a large matrix equation.

```
--> A = rand(100);
--> b = rand(100,1);
--> tic; c = A\b; toc

ans =
    0.0020
```

6.39 TOC Stop Stopwatch Timer

6.39.1 Usage

Stop the stopwatch timer, which can be used to time tasks in FreeMat. The `toc` function takes no arguments, and returns no outputs. You must use `toc` to get the elapsed time. The usage is

```
  toc
```

6.39.2 Example

Here is an example of timing the solution of a large matrix equation.

```
--> A = rand(100);
--> b = rand(100,1);
--> tic; c = A\b; toc

ans =
    0.0010
```

6.40 VERSION The Current Version Number

6.40.1 Usage

The `version` function returns the current version number for FreeMat (as a string). The general syntax for its use is

```
v = version
```

6.40.2 Example

The current version of FreeMat is

```
--> version

ans =
4.1
```

6.41 VERSTRING The Current Version String

6.41.1 Usage

The `verstring` function returns the current version string for FreeMat. The general syntax for its use is

```
version = verstring
```

6.41.2 Example

The current version of FreeMat is

```
--> verstring

ans =
FreeMat v4.1
```

Chapter 7

Debugging FreeMat Code

7.1 DBAUTO Control Dbauto Functionality

7.1.1 Usage

The dbauto functionality in FreeMat allows you to debug your FreeMat programs. When `dbauto` is `on`, then any error that occurs while the program is running causes FreeMat to stop execution at that point and return you to the command line (just as if you had placed a `keyboard` command there). You can then examine variables, modify them, and resume execution using `return`. Alternately, you can exit out of all running routines via a `retall` statement. Note that errors that occur inside of `try/catch` blocks do not (by design) cause auto breakpoints. The `dbauto` function toggles the dbauto state of FreeMat. The syntax for its use is

```
dbauto(state)
```

where `state` is either

```
dbauto('on')
```

to activate dbauto, or

```
dbauto('off')
```

to deactivate dbauto. Alternately, you can use FreeMat's string-syntax equivalence and enter

```
dbauto on
```

or

```
dbauto off
```

to turn dbauto on or off (respectively). Entering `dbauto` with no arguments returns the current state (either 'on' or 'off').

7.2 DBDELETE Delete a Breakpoint

7.2.1 Usage

The `dbdelete` function deletes a breakpoint. The syntax for the `dbdelete` function is

```
dbdelete(num)
```

where `num` is the number of the breakpoint to delete.

7.3 DBDown Move Down One Debug Level

7.3.1 Usage

The `dbdown` function moves up one level in the debug hierarchy. The syntax for the `dbdown` function is

```
dbdown
```

7.4 DBLIST List Breakpoints

7.4.1 Usage

List the current set of breakpoints. The syntax for the `dblist` is simply

```
dblist
```

7.5 DBSTEP Step N Statements

7.5.1 Usage

Step `N` statements during debug mode. The synax for this is either

```
dbstep(N)
```

to step `N` statements, or

```
dbstep
```

to step one statement.

7.6 DBSTOP

7.6.1 Usage

Set a breakpoint. The syntax for this is:

```
dbstop(funcname,linenumber)
```

where `funcname` is the name of the function where we want to set the breakpoint, and `linenumber` is the line number.

7.7 DBUP Move Up One Debug Level

7.7.1 Usage

The `dbup` function moves up one level in the debug hierarchy. The syntax for the `dbup` function is

```
dbup
```

7.8 FDUMP Dump Information on Function

7.8.1 Usage

Dumps information about a function (diagnostic information only)

```
fdump fname
```

Chapter 8

Sparse Matrix Support

8.1 EIGS Sparse Matrix Eigendecomposition

8.1.1 Usage

Computes the eigendecomsition of a sparse square matrix. The `eigs` function has several forms. The most general form is

```
[V,D] = eigs(A,k,sigma)
```

where `A` is the matrix to analyze, `k` is the number of eigenvalues to compute and `sigma` determines which eigenvallues to solve for. Valid values for `sigma` are 'lm' - largest magnitude 'sm' - smallest magnitude 'la' - largest algebraic (for real symmetric problems) 'sa' - smallest algebraic (for real symmetric problems) 'be' - both ends (for real symmetric problems) 'lr' - largest real part 'sr' - smallest real part 'li' - largest imaginary part 'si' - smallest imaginary part scalar - find the eigenvalues closest to `sigma`. The returned matrix `V` contains the eigenvectors, and `D` stores the eigenvalues. The related form

```
d = eigs(A,k,sigma)
```

computes only the eigenvalues and not the eigenvectors. If `sigma` is omitted, as in the forms

```
[V,D] = eigs(A,k)
```

and

```
d = eigs(A,k)
```

then `eigs` returns the largest magnitude eigenvalues (and optionally the associated eigenvectors). As an even simpler form, the forms

```
[V,D] = eigs(A)
```

and

```
d = eigs(A)
```

then `eigs` returns the six largest magnitude eigenvalues of `A` and optionally the eigenvectors. The `eigs` function uses ARPACK to compute the eigenvectors and/or eigenvalues. Note that due to a limitation in the interface into ARPACK from FreeMat, the number of eigenvalues that are to be computed must be strictly smaller than the number of columns (or rows) in the matrix.

8.1.2 Example

Here is an example of using `eigs` to calculate eigenvalues of a matrix, and a comparison of the results with `eig`

```
--> a = sparse(rand(9));
--> eigs(a)

ans =
   4.1831 +  0.0000i
   0.3249 -  0.5504i
   0.3249 +  0.5504i
   0.5932 -  0.1774i
   0.5932 +  0.1774i
  -0.5572 +  0.0000i

--> eig(full(a))

ans =
   4.1831 +  0.0000i
   0.5932 +  0.1774i
   0.5932 -  0.1774i
   0.3249 +  0.5504i
   0.3249 -  0.5504i
  -0.5572 +  0.0000i
  -0.1285 +  0.0901i
  -0.1285 -  0.0901i
  -0.3219 +  0.0000i
```

Next, we exercise some of the variants of `eigs`:

```
--> eigs(a,4,'sm')

ans =
  -0.1285 +  0.0901i
  -0.1285 -  0.0901i
  -0.3219 +  0.0000i
  -0.5572 +  0.0000i

--> eigs(a,4,'lr')

ans =
   4.1831 +  0.0000i
   0.5932 +  0.1774i
   0.5932 -  0.1774i
   0.3249 +  0.5504i

--> eigs(a,4,'sr')

ans =
  -0.5572 +  0.0000i
  -0.3219 +  0.0000i
  -0.1285 +  0.0901i
  -0.1285 -  0.0901i
```

8.2 FULL Convert Sparse Matrix to Full Matrix

8.2.1 Usage

Converts a sparse matrix to a full matrix. The syntax for its use is

```
y = full(x)
```

The type of `x` is preserved. Be careful with the function. As a general rule of thumb, if you can work with the `full` representation of a function, you probably do not need the sparse representation.

8.2.2 Example

Here we convert a full matrix to a sparse one, and back again.

```
--> a = [1,0,4,2,0;0,0,0,0,0;0,1,0,0,2]

a =
 1 0 4 2 0
 0 0 0 0 0
 0 1 0 0 2

--> A = sparse(a)

A =
 1 1 1
 3 2 1
 1 3 4
 1 4 2
 3 5 2
--> full(A)

ans =
 1 0 4 2 0
 0 0 0 0 0
 0 1 0 0 2
```

8.3 SPARSE Construct a Sparse Matrix

8.3.1 Usage

Creates a sparse matrix using one of several formats. The first creates a sparse matrix from a full matrix

```
y = sparse(x).
```

The second form creates a sparse matrix containing all zeros that is of the specified size (the sparse equivalent of `zeros`).

```
y = sparse(m,n)
```

where `m` and `n` are integers. Just like the `zeros` function, the sparse matrix returned is of type `float`. The third form constructs a sparse matrix from the IJV syntax. It has two forms. The first version autosizes the sparse matrix

```
y = sparse(i,j,v)
```

while the second version uses an explicit size specification

```
y = sparse(i,j,v,m,n)
```

8.4 SPEYE Sparse Identity Matrix

8.4.1 Usage

Creates a sparse identity matrix of the given size. The syntax for its use is

```
y = speye(m,n)
```

which forms an `m x n` sparse matrix with ones on the main diagonal, or

```
y = speye(n)
```

which forms an `n x n` sparse matrix with ones on the main diagonal. The matrix type is a `float` single precision matrix.

8.4.2 Example

The following creates a 5000 by 5000 identity matrix, which would be difficult to do using `sparse(eye(5000))` because of the large amount of intermediate storage required.

```
--> I = speye(5000);
--> who I
  Variable Name       Type   Flags             Size
              I    double   sparse           [5000x5000]
--> full(I(1:10,1:10))

ans =
 1 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 1
```

8.5 SPONES Sparse Ones Function

8.5.1 Usage

Returns a sparse `float` matrix with ones where the argument matrix has nonzero values. The general syntax for it is

```
y = spones(x)
```

where `x` is a matrix (it may be full or sparse). The output matrix `y` is the same size as `x`, has type `float`, and contains ones in the nonzero positions of `x`.

8.5.2 Examples

Here are some examples of the `spones` function

```
--> a = [1,0,3,0,5;0,0,2,3,0;1,0,0,0,1]

a =
```

```
 1 0 3 0 5
 0 0 2 3 0
 1 0 0 0 1

--> b = spones(a)

b =
 1 1 1
 1 2 1
 1 3 1
 1 4 1
 1 5 1
--> full(b)

ans =
 1 1 1 1 1
 0 0 0 0 0
 0 0 0 0 0
```

8.6 SPRAND Sparse Uniform Random Matrix

8.6.1 Usage

Creates a sparse matrix with uniformly distributed random entries (on [0,1]). The syntax for its use is

```
y = sprand(x)
```

where `x` is a sparse matrix, where `y` is a sparse matrix that has random entries where `x` is nonzero. The second form specifies the size of the matrix and the density

```
y = sprand(m,n,density)
```

where `m` is the number of rows in the output, `n` is the number of columns in the output, and `density` (which is between 0 and 1) is the density of nonzeros in the resulting matrix. Note that for very high densities the actual density of the output matrix may differ from the density you specify. This difference is a result of the way the random entries into the matrix are generated. If you need a very dense random matrix, it is better to generate a full matrix and zero out the entries you do not need.

8.6.2 Examples

Here we seed `sprand` with a full matrix (to demonstrate how the structure of the output is determined by the input matrix when using the first form).

```
--> x = [1,0,0;0,0,1;1,0,0]

x =
 1 0 0
 0 0 1
 1 0 0

--> y = sprand(x)

y =
 1 1 0.171364
 3 1 0.245464
 2 3 0.0426635
```

```
--> full(y)

ans =
    0.1714         0         0
         0         0    0.0427
    0.2455         0         0
```

The more generic version with a density of `0.001`. On many systems the following is impossible using full matrices

```
--> y = sprand(10000,10000,.001);
--> nnz(y)/10000^2

ans =
 9.9946e-04
```

8.7 SPRANDN Sparse Normal Random Matrix

8.7.1 Usage

Creates a sparse matrix with normally distributed random entries (mean 0, sigma 1). The syntax for its use is

```
  y = sprandn(x)
```

where `x` is a sparse matrix, where `y` is a sparse matrix that has random entries where `x` is nonzero. The second form specifies the size of the matrix and the density

```
  y = sprandn(m,n,density)
```

where `m` is the number of rows in the output, `n` is the number of columns in the output, and `density` (which is between 0 and 1) is the density of nonzeros in the resulting matrix. Note that for very high densities the actual density of the output matrix may differ from the density you specify. This difference is a result of the way the random entries into the matrix are generated. If you need a very dense random matrix, it is better to generate a full matrix and zero out the entries you do not need.

8.7.2 Examples

Here we seed `sprandn` with a full matrix (to demonstrate how the structure of the output is determined by the input matrix when using the first form).

```
--> x = [1,0,0;0,0,1;1,0,0]

x =
 1 0 0
 0 0 1
 1 0 0

--> y = sprandn(x)

y =
 1 1 -0.498012
 3 1 0.813313
 2 3 -1.10282
--> full(y)
```

```
ans =
   -0.4980         0         0
         0         0   -1.1028
    0.8133         0         0
```

The more generic version with a density of `0.001`. On many systems the following is impossible using full matrices

```
--> y = sprandn(10000,10000,.001);
--> nnz(y)/10000^2

ans =
 9.9952e-04
```

8.8 SPY Visualize Sparsity Pattern of a Sparse Matrix

8.8.1 Usage

Plots the sparsity pattern of a sparse matrix. The syntax for its use is

```
spy(x)
```

which uses a default color and symbol. Alternately, you can use

```
spy(x,colspec)
```

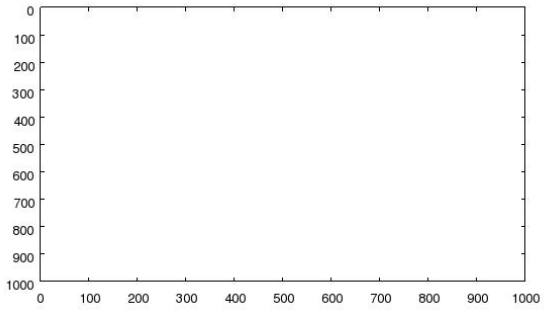
where `colspec` is any valid color and symbol spec accepted by `plot`.

8.8.2 Example

First, an example of a random sparse matrix.

```
--> y = sprand(1000,1000,.001);
--> spy(y,'ro')
```

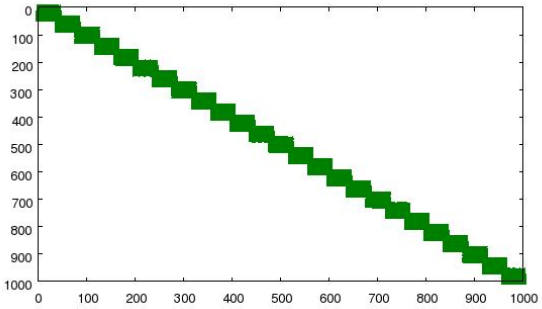
which is shown here



Here is a sparse matrix with a little more structure. First we build a sparse matrix with block diagonal structure, and then use `spy` to visualize the structure.

```
--> A = sparse(1000,1000);
--> for i=1:25; A((1:40) + 40*(i-1),(1:40) + 40*(i-1)) = 1; end;
--> spy(A,'gx')
```

with the result shown here


0
100
200
300
400
500
600
700
800
900
1000
0
100
200
300
400
500
600
700
800
900
1000

Chapter 9

Mathematical Functions

9.1 ACOS Inverse Trigonometric Arccosine Function

9.1.1 Usage

Computes the `acos` function for its argument. The general syntax for its use is

```
y = acos(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `acos` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.1.2 Function Internals

Mathematically, the `acos` function is defined for all arguments `x` as

$$\text{acos}x \equiv \frac{pi}{2} + i \log\left(ix + \sqrt{1 - x^2}\right).$$

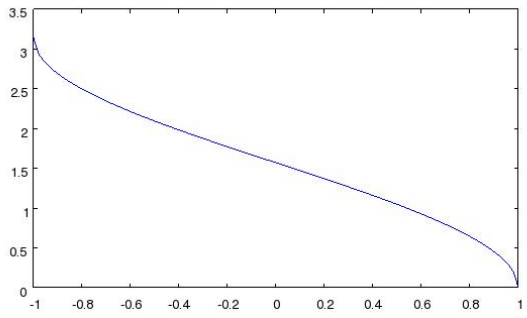
For real valued variables `x` in the range `[-1,1]`, the function is computed directly using the standard C library's numerical `acos` function. For both real and complex arguments `x`, note that generally

$$\text{acos}(\cos(x)) \neq x,$$

9.1.3 Example

The following code demonstates the `acos` function over the range `[-1,1]`.

```
--> t = linspace(-1,1);
--> plot(t,acos(t))
```



9.2 ACOSD Inverse Cosine Degrees Function

9.2.1 Usage

Computes the inverse cosine of the argument, but returns the argument in degrees instead of radians (as is the case for `acos`. The syntax for its use is

```
y = acosd(x)
```

9.2.2 Examples

The inverse cosine of `sqrt(2)/2` should be 45 degrees:

```
--> acosd(sqrt(2)/2)

ans =
  45.0000 +  0.0000i
```

and the inverse cosine of `0.5` should be 60 degrees:

```
--> acosd(0.5)

ans =
  60.0000 +  0.0000i
```

9.3 ACOSH Inverse Hyperbolic Cosine Function

9.3.1 Usage

Computes the inverse hyperbolic cosine of its argument. The general syntax for its use is

```
y = acosh(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.3.2 Function Internals

The `acosh` function is computed from the formula

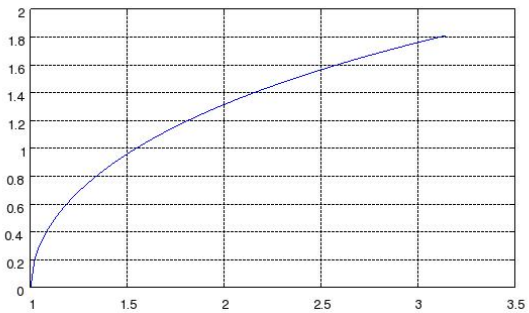
$$\cosh^{-1}(x) = \log\left(x + (x^2 - 1)^0.5\right)$$

where the `log` (and square root) is taken in its most general sense.

9.3.3 Examples

Here is a simple plot of the inverse hyperbolic cosine function

```
--> x = linspace(1,pi);
--> plot(x,acosh(x)); grid('on');
```



9.4 ACOT Inverse Cotangent Function

9.4.1 Usage

Computes the inverse cotangent of its argument. The general syntax for its use is

```
y = acot(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.4.2 Function Internals

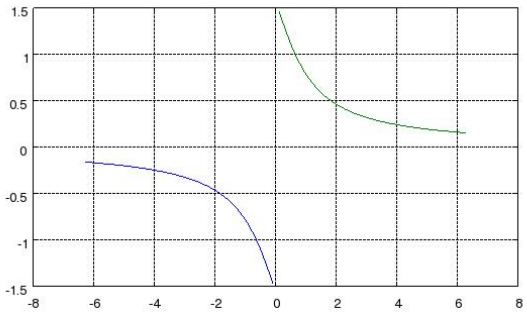
The `acot` function is computed from the formula

$$\cot^{-1}(x) = \tan^{-1}\left(\frac{1}{x}\right)$$

9.4.3 Examples

Here is a simple plot of the inverse cotangent function

```
--> x1 = -2*pi:pi/30:-0.1;
--> x2 = 0.1:pi/30:2*pi;
--> plot(x1,acot(x1),x2,acot(x2)); grid('on');
```



9.5 ACOTD Inverse Cotangent Degrees Function

9.5.1 Usage

Computes the inverse cotangent of its argument in degrees. The general syntax for its use is

```
y = acotd(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.6 ACOTH Inverse Hyperbolic Cotangent Function

9.6.1 Usage

Computes the inverse hyperbolic cotangent of its argument. The general syntax for its use is

```
y = acoth(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.6.2 Function Internals

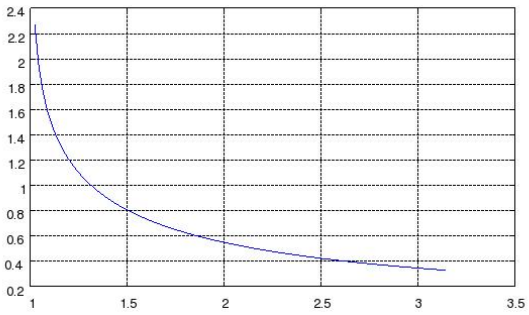
The `acoth` function is computed from the formula

$$\coth^{-1}(x) = \tanh^{-1}\left(\frac{1}{x}\right)$$

9.6.3 Examples

Here is a simple plot of the inverse hyperbolic cotangent function

```
--> x = linspace(1,pi);
--> plot(x,acoth(x)); grid('on');
```



9.7 ACSC Inverse Cosecant Function

9.7.1 Usage

Computes the inverse cosecant of its argument. The general syntax for its use is

```
y = acsc(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.7.2 Function Internals

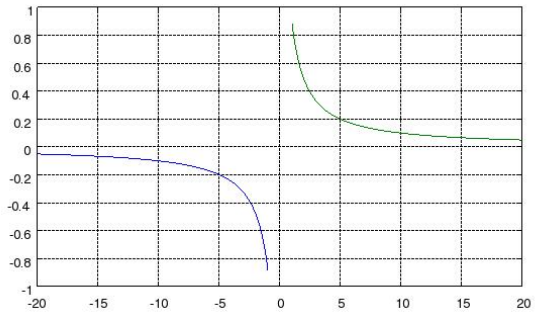
The `acosh` function is computed from the formula

$$\csc^{-1}(x) = \sin^{-1}\left(\frac{1}{x}\right)$$

9.7.3 Examples

Here is a simple plot of the inverse cosecant function

```
--> x1 = -10:.01:-1.01;
--> x2 = 1.01:.01:10;
--> plot(x1,acsc(x1),x2,acsc(x2)); grid('on');
```



9.8 ACSCD Inverse Cosecant Degrees Function

9.8.1 Usage

Computes the inverse cosecant of the argument, but returns the argument in degrees instead of radians (as is the case for `acsc`. The syntax for its use is

```
y = acscd(x)
```

9.8.2 Examples

The inverse cosecant of `2/sqrt(2)` should be 45 degrees:

```
--> acscd(2/sqrt(2))

ans =
   45.0000
```

and the inverse cosecant of `2` should be 30 degrees:

```
--> acscd(0.5)

ans =
  90.0000 - 75.4561i
```

9.9 ACSCH Inverse Hyperbolic Cosecant Function

9.9.1 Usage

Computes the inverse hyperbolic cosecant of its argument. The general syntax for its use is

```
y = acsch(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.9.2 Function Internals

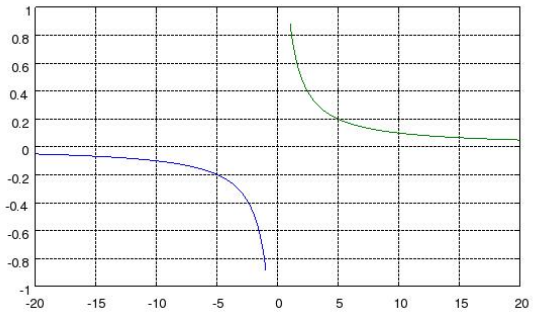
The `acsch` function is computed from the formula

$$\operatorname{csch}^{-1}(x) = \sinh^{-1}\left(\frac{1}{x}\right)$$

9.9.3 Examples

Here is a simple plot of the inverse hyperbolic cosecant function

```
--> x1 = -20:.01:-1;
--> x2 = 1:.01:20;
--> plot(x1,acsch(x1),x2,acsch(x2)); grid('on');
```



9.10 ANGLE Phase Angle Function

9.10.1 Usage

Compute the phase angle in radians of a complex matrix. The general syntax for its use is

```
p = angle(c)
```

where `c` is an `n`-dimensional array of numerical type.

9.10.2 Function Internals

For a complex number `x`, its polar representation is given by

$$x = |x|e^{j\theta}$$

and we can compute

$$\theta = \text{atan2}(\Im x, \Re x)$$

9.10.3 Example

Here are some examples of the use of `angle` in the polar decomposition of a complex number.

```
--> x = 3+4*i

x =
   3.0000 +  4.0000i

--> a = abs(x)

a =
 5

--> t = angle(x)

t =
    0.9273
```

```
--> a*exp(i*t)

ans =
   3.0000 +  4.0000i
```

M version contributor: M.W. Vogel 01-30-06

9.11 ASEC Inverse Secant Function

9.11.1 Usage

Computes the inverse secant of its argument. The general syntax for its use is

```
y = asec(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.11.2 Function Internals

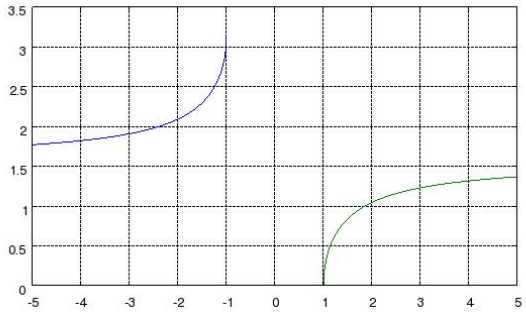
The `acosh` function is computed from the formula

$$\sec^{-1}(x) = \cos^{-1}\left(\frac{1}{x}\right)$$

9.11.3 Examples

Here is a simple plot of the inverse secant function

```
--> x1 = -5:.01:-1;
--> x2 = 1:.01:5;
--> plot(x1,asec(x1),x2,asec(x2)); grid('on');
```



9.12 ASECD Inverse Secant Degrees Function

9.12.1 Usage

Computes the inverse secant of the argument, but returns the argument in degrees instead of radians (as is the case for `asec`. The syntax for its use is

```
y = asecd(x)
```

9.12.2 Examples

The inverse secant of `2/sqrt(2)` should be 45 degrees:

```
--> asecd(2/sqrt(2))

ans =
  45.0000 +  0.0000i
```

and the inverse secant of `2` should be 60 degrees:

```
--> asecd(2)

ans =
  60.0000 +  0.0000i
```

9.13 ASECH Inverse Hyperbolic Secant Function

9.13.1 Usage

Computes the inverse hyperbolic secant of its argument. The general syntax for its use is

```
y = asech(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.13.2 Function Internals

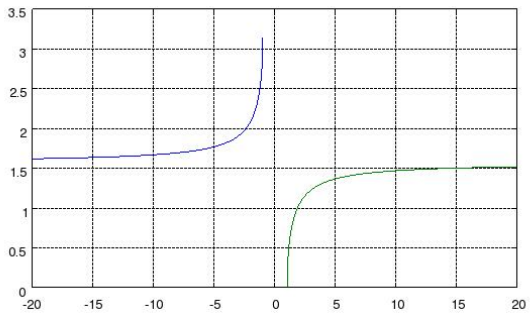
The `asech` function is computed from the formula

$$\operatorname{sech}^{-1}(x) = \cosh^{-1}\left(\frac{1}{x}\right)$$

9.13.3 Examples

Here is a simple plot of the inverse hyperbolic secant function

```
--> x1 = -20:.01:-1;
--> x2 = 1:.01:20;
--> plot(x1,imag(asech(x1)),x2,imag(asech(x2))); grid('on');
```



9.14 ASIN Inverse Trigonometric Arcsine Function

9.14.1 Usage

Computes the `asin` function for its argument. The general syntax for its use is

```
y = asin(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `asin` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.14.2 Function Internals

Mathematically, the `asin` function is defined for all arguments `x` as

$$\text{asin}x \equiv -i \log\left(ix + \sqrt{1-x^2}\right).$$

For real valued variables `x` in the range `[-1,1]`, the function is computed directly using the standard C library's numerical `asin` function. For both real and complex arguments `x`, note that generally

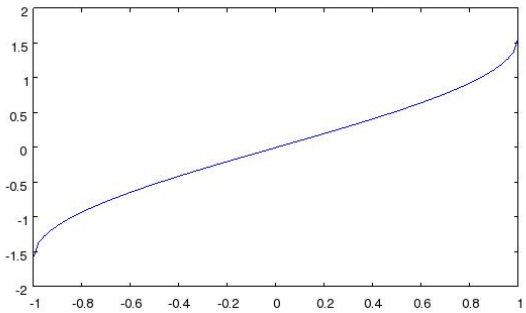
$$\text{asin}(\sin(x)) \neq x,$$

due to the periodicity of `sin(x)`.

9.14.3 Example

The following code demonstates the `asin` function over the range `[-1,1]`.

```
--> t = linspace(-1,1);
--> plot(t,asin(t))
```



9.15 ASIND Inverse Sine Degrees Function

9.15.1 Usage

Computes the inverse sine of the argument, but returns the argument in degrees instead of radians (as is the case for `asin`). The syntax for its use is

```
y = asind(x)
```

9.15.2 Examples

The inverse sine of `sqrt(2)/2` should be 45 degrees:

```
--> asind(sqrt(2)/2)

ans =
   45.0000
```

and the inverse sine of `0.5` should be 30 degrees:

```
--> asind(0.5)

ans =
   30.0000
```

9.16 ASINH Inverse Hyperbolic Sine Function

9.16.1 Usage

Computes the inverse hyperbolic sine of its argument. The general syntax for its use is

```
y = asinh(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.16.2 Function Internals

The `asinh` function is computed from the formula

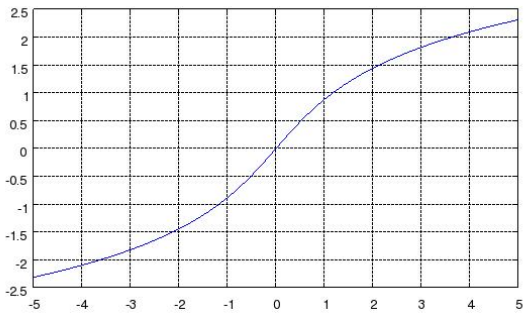
$$\sinh^{-1}(x) = \log\left(x + (x^2+1)^0.5\right)$$

where the `log` (and square root) is taken in its most general sense.

9.16.3 Examples

Here is a simple plot of the inverse hyperbolic sine function

```
--> x = -5:.01:5;
--> plot(x,asinh(x)); grid('on');
```



9.17 ATAN Inverse Trigonometric Arctangent Function

9.17.1 Usage

Computes the `atan` function for its argument. The general syntax for its use is

```
y = atan(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `atan` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.17.2 Function Internals

Mathematically, the `atan` function is defined for all arguments `x` as

$$\text{atan}x \equiv \frac{i}{2}\left(\log(1-ix) - \log(ix+1)\right).$$

For real valued variables `x`, the function is computed directly using the standard C library's numerical `atan` function. For both real and complex arguments `x`, note that generally

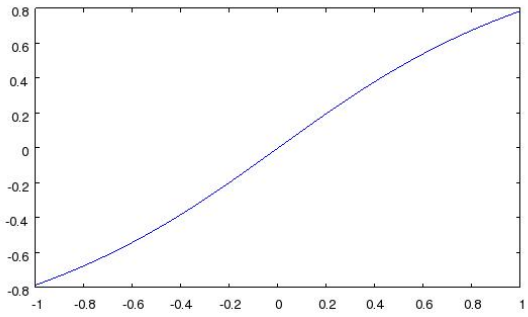
$$\text{atan}(\tan(x)) \neq x,$$

due to the periodicity of `tan(x)`.

9.17.3 Example

The following code demonstates the `atan` function over the range `[-1,1]`.

```
--> t = linspace(-1,1);
--> plot(t,atan(t))
```



9.18 ATAN2 Inverse Trigonometric 4-Quadrant Arctangent Function

9.18.1 Usage

Computes the `atan2` function for its argument. The general syntax for its use is

```
z = atan2(y,x)
```

where `x` and `y` are `n`-dimensional arrays of numerical type. Integer types are promoted to the `double` type prior to calculation of the `atan2` function. The size of the output depends on the size of `x` and `y`. If `x` is a scalar, then `z` is the same size as `y`, and if `y` is a scalar, then `z` is the same size as `x`. The type of the output is equal to the type of —y/x—.

9.18.2 Function Internals

The function is defined (for real values) to return an angle between `-pi` and `pi`. The signs of `x` and `y` are used to find the correct quadrant for the solution. For complex arguments, the two-argument arctangent is computed via

$$\text{atan2}(y,x) \equiv -i \log\left(\frac{x+iy}{\sqrt{x^2+y^2}}\right)$$

For real valued arguments `x,y`, the function is computed directly using the standard C library's numerical `atan2` function. For both real and complex arguments `x`, note that generally

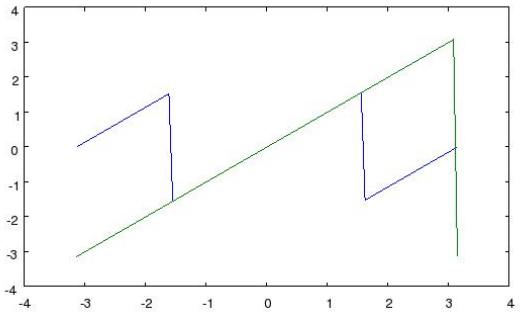
$$\text{atan2}(\sin(x),\cos(x)) \neq x,$$

due to the periodicities of `cos(x)` and `sin(x)`.

9.18.3 Example

The following code demonstates the difference between the `atan2` function and the `atan` function over the range `[-pi,pi]`.

```
--> x = linspace(-pi,pi);
--> sx = sin(x); cx = cos(x);
--> plot(x,atan(sx./cx),x,atan2(sx,cx))
```



Note how the two-argument `atan2` function (green line) correctly "unwraps" the phase of the angle, while the `atan` function (red line) wraps the angle to the interval `[-i/2,i/2]`.

9.19 ATAND Inverse Tangent Degrees Function

9.19.1 Usage

Computes the inverse tangent of the argument, but returns the argument in degrees instead of radians (as is the case for `atan`. The syntax for its use is

```
y = atand(x)
```

9.19.2 Examples

The inverse tangent of `1` should be 45 degrees:

```
--> atand(1)

ans =
 45
```

9.20 ATANH Inverse Hyperbolic Tangent Function

9.20.1 Usage

Computes the inverse hyperbolic tangent of its argument. The general syntax for its use is

```
y = atanh(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.20.2 Function Internals

The `atanh` function is computed from the formula

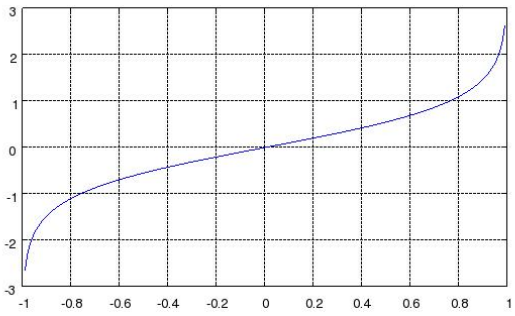
$$\tanh^{-1}(x) = \frac{1}{2}\log\left(\frac{1+x}{1-x}\right)$$

where the `log` (and square root) is taken in its most general sense.

9.20.3 Examples

Here is a simple plot of the inverse hyperbolic tangent function

```
--> x = -0.99:.01:0.99;
--> plot(x,atanh(x)); grid('on');
```



9.21 BETAINC Incomplete Beta Function

9.21.1 Usage

Computes the incomplete beta function. The `betainc` function takes 3 or 4 arguments

```
A = betainc(X,Y,Z)
```

```
A = betainc(X,Y,Z,tail)
```

where `X` is either a `float` or `double` array with elements in [0,1] interval, `Y` and `Z` are real non-negative arrays. `tail` specifies the tail of the incomplete beta function. If `tail` is 'lower' (default) than the integral from 0 to x is computed. If `tail` is 'upper' than the integral from x to 1 is computed. All arrays must be the same size or be scalar. The output vector `A` is the same size (and type) as input arrays.

9.21.2 Function Internals

The incomplete beta function is defined by the integral:

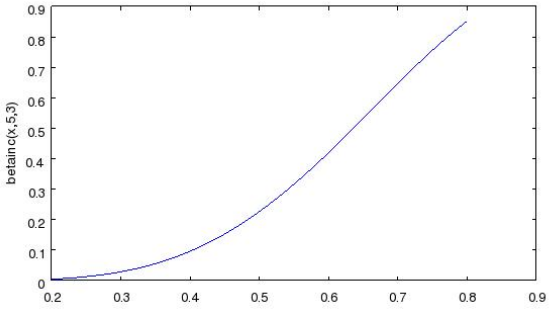
$$BetaI_x(a,b) = B_x(a,b)/B(a,b) where B_x(a,b) = \int_0^x t^{a-1}(1-t)^{b-1}dt for 0 <= x <= 1. For a > 0, b > 0$$

9.21.3 Example

Here is a plot of the betainc function over the range `[.2,.8]`.

```
--> x=.2:.01:.8;
--> y = betainc(x,5,3);
--> plot(x,y); xlabel('x'); ylabel('betainc(x,5,3)');
```

which results in the following plot.



9.22 COS Trigonometric Cosine Function

9.22.1 Usage

Computes the `cos` function for its argument. The general syntax for its use is

```
y = cos(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `cos` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.22.2 Function Internals

Mathematically, the `cos` function is defined for all real valued arguments `x` by the infinite summation

$$\cos x \equiv \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}.$$

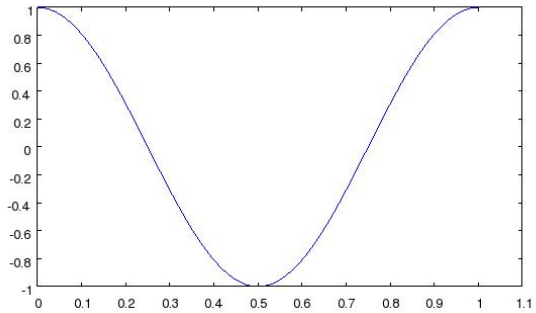
For complex valued arguments `z`, the cosine is computed via

$$\cos z \equiv \cos \Re z \cosh \Im z - \sin \Re z \sinh \Im z.$$

9.22.3 Example

The following piece of code plots the real-valued `cos(2 pi x)` function over one period of `[0,1]`:

```
--> x = linspace(0,1);
--> plot(x,cos(2*pi*x))
```



9.23 COSD Cosine Degrees Function

9.23.1 Usage

Computes the cosine of the argument, but takes the argument in degrees instead of radians (as is the case for `cos`). The syntax for its use is

```
y = cosd(x)
```

9.23.2 Examples

The cosine of 45 degrees should be `sqrt(2)/2`

```
--> cosd(45)

ans =
    0.7071
```

and the cosine of `60` degrees should be 0.5:

```
--> cosd(60)

ans =
    0.5000
```

9.24 COSH Hyperbolic Cosine Function

9.24.1 Usage

Computes the hyperbolic cosine of the argument. The syntax for its use is

```
y = cosh(x)
```

9.24.2 Function Internals

The `cosh` function is computed from the formula

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

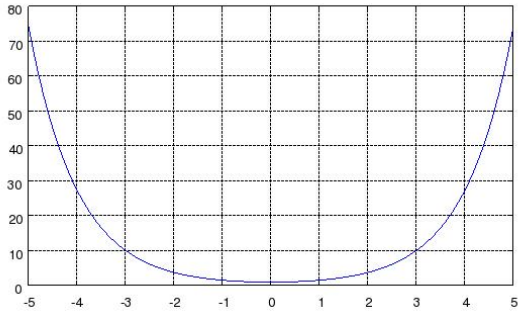
For `x` complex, it follows that

$$\cosh(a + i * b) = \frac{e^a(\cos(b) + i * \sin(b)) + e^{-a}(\cos(-b) + i * \sin(-b))}{2}$$

9.24.3 Examples

Here is a simple plot of the hyperbolic cosine function

```
--> x = linspace(-5,5);
--> plot(x,cosh(x)); grid('on');
```



9.25 COT Trigonometric Cotangent Function

9.25.1 Usage

Computes the `cot` function for its argument. The general syntax for its use is

```
y = cot(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `cot` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.25.2 Function Internals

Mathematically, the `cot` function is defined for all arguments `x` as

$$\cot x \equiv \frac{\cos x}{\sin x}$$

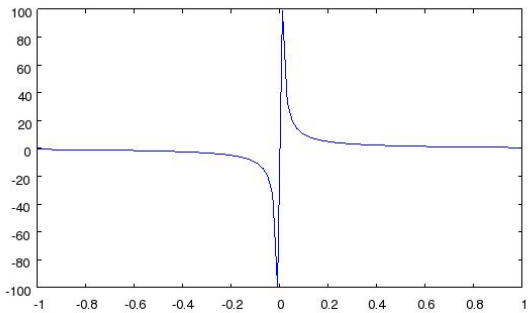
For complex valued arguments `z`, the cotangent is computed via

$$\cot z \equiv \frac{\cos 2\Re z + \cosh 2\Im z}{\sin 2\Re z + i \sinh 2\Im z}.$$

9.25.3 Example

The following piece of code plots the real-valued `cot(x)` function over the interval `[-1,1]`:

```
--> t = linspace(-1,1);
--> plot(t,cot(t))
```



9.26 COTD Cotangent Degrees Function

9.26.1 Usage

Computes the cotangent of the argument, but takes the argument in degrees instead of radians (as is the case for `cot`). The syntax for its use is

```
y = cotd(x)
```

9.26.2 Examples

The cotangent of 45 degrees should be 1.

```
--> cotd(45)

ans =
    1.0000
```

9.27 COTH Hyperbolic Cotangent Function

9.27.1 Usage

Computes the hyperbolic cotangent of the argument. The syntax for its use is

```
y = coth(x)
```

9.27.2 Function Internals

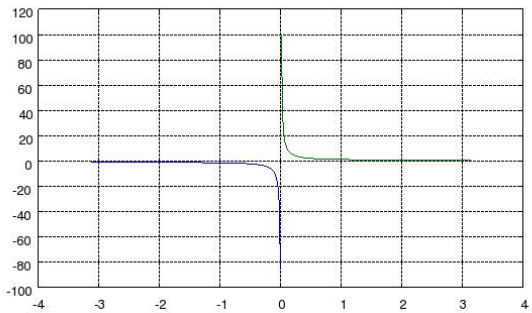
The `coth` function is computed from the formula

$$\coth(x) = \frac{1}{\tanh(x)}$$

9.27.3 Examples

Here is a simple plot of the hyperbolic cotangent function

```
--> x1 = -pi+.01:.01:-.01;
--> x2 = .01:.01:pi-.01;
--> plot(x1,coth(x1),x2,coth(x2)); grid('on');
```



9.28 CROSS Cross Product of Two Vectors

9.28.1 Usage

Computes the cross product of two vectors. The general syntax for its use is

```
c = cross(a,b)
```

where `a` and `b` are 3-element vectors.

9.29 CSC Trigonometric Cosecant Function

9.29.1 Usage

Computes the `csc` function for its argument. The general syntax for its use is

```
y = csc(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `csc` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.29.2 Function Internals

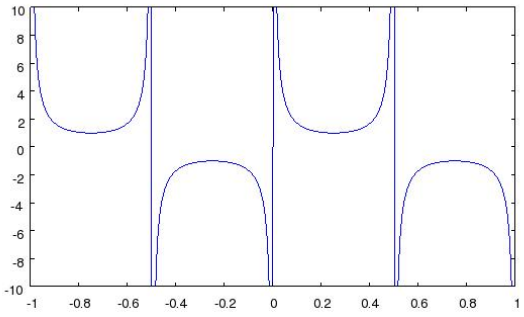
Mathematically, the `csc` function is defined for all arguments as

$$\csc x \equiv \frac{1}{\sin x}.$$

9.29.3 Example

The following piece of code plots the real-valued `csc(2 pi x)` function over the interval of `[-1,1]`:

```
--> t = linspace(-1,1,1000);
--> plot(t,csc(2*pi*t))
--> axis([-1,1,-10,10]);
```



9.30 CSCD Cosecant Degrees Function

9.30.1 Usage

Computes the cosecant of the argument, but takes the argument in degrees instead of radians (as is the case for `csc`). The syntax for its use is

```
y = cscd(x)
```

9.31 CSCH Hyperbolic Cosecant Function

9.31.1 Usage

Computes the hyperbolic cosecant of the argument. The syntax for its use is

```
y = csch(x)
```

9.31.2 Function Internals

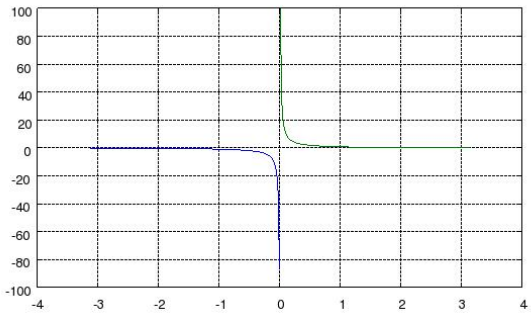
The `csch` function is computed from the formula

$$\mathrm{csch}(x) = \frac{1}{\sinh(x)}$$

9.31.3 Examples

Here is a simple plot of the hyperbolic cosecant function

```
--> x1 = -pi+.01:.01:-.01;
--> x2 = .01:.01:pi-.01;
--> plot(x1,csch(x1),x2,csch(x2)); grid('on');
```



9.32 DEG2RAD Convert From Degrees To Radians

9.32.1 Usage

Converts the argument from degrees to radians. The syntax for its use is

```
y = deg2rad(x)
```

where `x` is a numeric array. Conversion is done by simply multiplying `x` by `pi/180`.

9.32.2 Example

How many radians in a circle:

```
--> deg2rad(360) - 2*pi

ans =
 0
```

9.33 ERF Error Function

9.33.1 Usage

Computes the error function for real arguments. The `erf` function takes only a single argument

```
y = erf(x)
```

where `x` is either a `float` or `double` array. The output vector `y` is the same size (and type) as `x`.

9.33.2 Function Internals

The erf function is defined by the integral:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt,$$

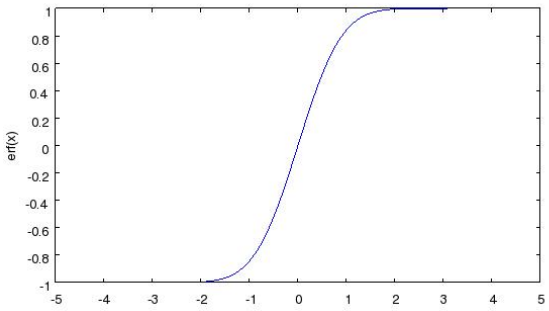
and is the integral of the normal distribution.

9.33.3 Example

Here is a plot of the erf function over the range `[-5,5]`.

```
--> x = linspace(-5,5);
--> y = erf(x);
--> plot(x,y); xlabel('x'); ylabel('erf(x)');
```

which results in the following plot.



9.34 ERFC Complimentary Error Function

9.34.1 Usage

Computes the complimentary error function for real arguments. The `erfc` function takes only a single argument

```
y = erfc(x)
```

where `x` is either a `float` or `double` array. The output vector `y` is the same size (and type) as `x`.

9.34.2 Function Internals

The erfc function is defined by the integral:

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2}\, dt,$$

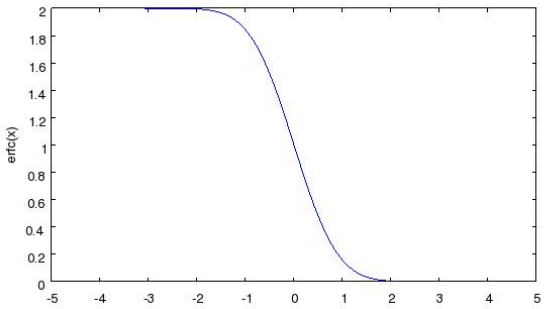
and is the integral of the normal distribution.

9.34.3 Example

Here is a plot of the `erfc` function over the range `[-5,5]`.

```
--> x = linspace(-5,5);
--> y = erfc(x);
--> plot(x,y); xlabel('x'); ylabel('erfc(x)');
```

which results in the following plot.



9.35 ERFINV Inverse Error Function

9.35.1 Usage

Computes the inverse error function for each element of x. The `erf` function takes only a single argument

```
y = erfinv(x)
```

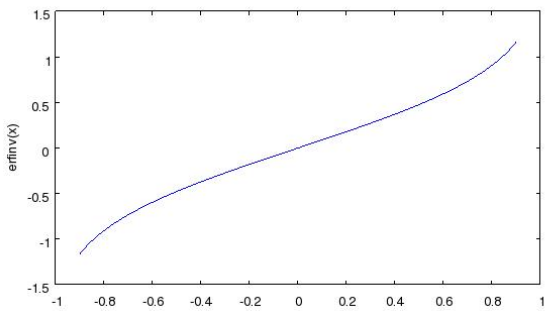
where `x` is either a `float` or `double` array. The output vector `y` is the same size (and type) as `x`. For values outside the interval [-1, 1] function returns NaN.

9.35.2 Example

Here is a plot of the erf function over the range `[-.9,.9]`.

```
--> x = linspace(-.9,.9,100);
--> y = erfinv(x);
--> plot(x,y); xlabel('x'); ylabel('erfinv(x)');
```

which results in the following plot.



9.36 EXP Exponential Function

9.36.1 Usage

Computes the `exp` function for its argument. The general syntax for its use is

```
y = exp(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `exp` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.36.2 Function Internals

Mathematically, the `exp` function is defined for all real valued arguments `x` as

$$\exp x \equiv e^x,$$

where

$$e = \sum_0^\infty \frac{1}{k!}$$

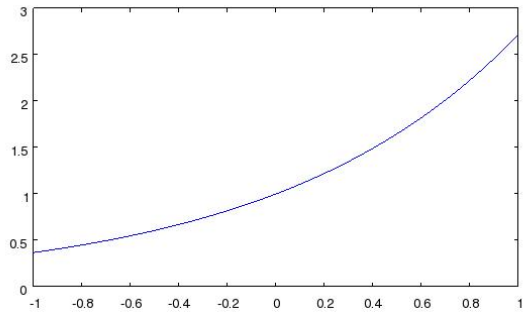
and is approximately `2.718281828459045` (returned by the function `e`). For complex values `z`, the famous Euler formula is used to calculate the exponential

$$e^z = e^{|z|}\left[\cos \Re z + i \sin \Re z\right]$$

9.36.3 Example

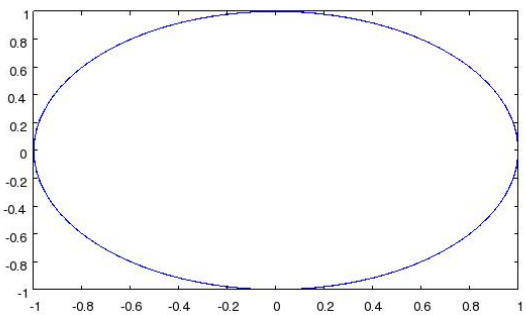
The following piece of code plots the real-valued `exp` function over the interval `[-1,1]`:

```
--> x = linspace(-1,1);
--> plot(x,exp(x))
```



In the second example, we plot the unit circle in the complex plane `e\^{i 2 pi x}` for `x in [-1,1]`.

```
--> x = linspace(-1,1);
--> plot(exp(-i*x*2*pi))
```



9.37 EXPM1 Exponential Minus One Function

9.37.1 Usage

Computes `exp(x)-1` function accurately for `x` small. The syntax for its use is

```
y = expm1(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.38 FIX Round Towards Zero

9.38.1 Usage

Rounds the argument array towards zero. The syntax for its use is

```
y = fix(x)
```

where `x` is a numeric array. For positive elements of `x`, the output is the largest integer smaller than `x`. For negative elements of `x` the output is the smallest integer larger than `x`. For complex `x`, the operation is applied seperately to the real and imaginary parts.

9.38.2 Example

Here is a simple example of the `fix` operation on some values

```
--> a = [-1.8,pi,8,-pi,-0.001,2.3+0.3i]

a =
  -1.8000 +  0.0000i   3.1416 +  0.0000i   8.0000 +  0.0000i  -3.1416 +  0.0000i  -0.0010 +  0.0000i   1

--> fix(a)

ans =
  -1.0000 +  0.0000i   3.0000 +  0.0000i   8.0000 +  0.0000i  -3.0000 +  0.0000i        0             2
```

9.39 GAMMA Gamma Function

9.39.1 Usage

Computes the gamma function for real arguments. The `gamma` function takes only a single argument

```
y = gamma(x)
```

where `x` is either a `float` or `double` array. The output vector `y` is the same size (and type) as `x`.

9.39.2 Function Internals

The gamma function is defined by the integral:

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1}\, dt$$

The gamma function obeys the interesting relationship

$$\Gamma(x) = (x-1)\Gamma(x-1),$$

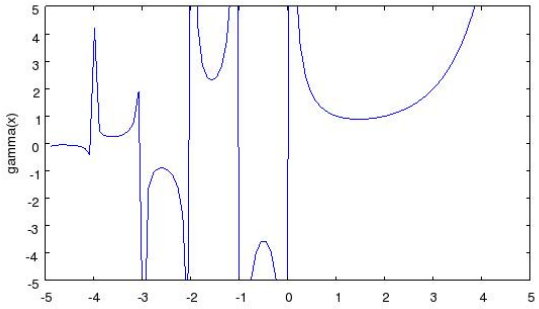
and for integer arguments, is equivalent to the factorial function.

9.39.3 Example

Here is a plot of the gamma function over the range `[-5,5]`.

```
--> x = linspace(-5,5);
--> y = gamma(x);
--> plot(x,y); xlabel('x'); ylabel('gamma(x)');
--> axis([-5,5,-5,5]);
```

which results in the following plot.



9.40 GAMMALN Log Gamma Function

9.40.1 Usage

Computes the natural log of the gamma function for real arguments. The `gammaln` function takes only a single argument

```
y = gammaln(x)
```

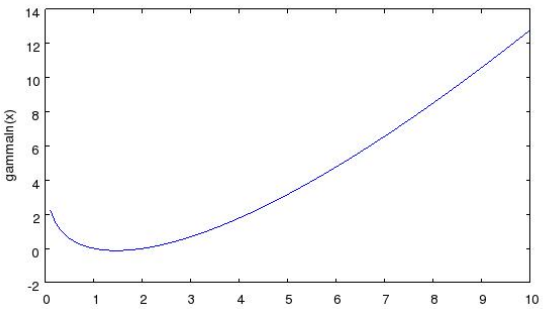
where `x` is either a `float` or `double` array. The output vector `y` is the same size (and type) as `x`.

9.40.2 Example

Here is a plot of the `gammaln` function over the range `[-5,5]`.

```
--> x = linspace(0,10);
--> y = gammaln(x);
--> plot(x,y); xlabel('x'); ylabel('gammaln(x)');
```

which results in the following plot.



9.41 IDIV Integer Division Operation

9.41.1 Usage

Computes the integer division of two arrays. The syntax for its use is

```
y = idiv(a,b)
```

where `a` and `b` are arrays or scalars. The effect of the `idiv` is to compute the integer division of `b` into `a`.

9.41.2 Example

The following examples show some uses of `idiv` arrays.

```
--> idiv(27,6)

ans =
 4

--> idiv(4,-2)

ans =
 -2

--> idiv(15,3)

ans =
 5
```

9.42 LEGENDRE Associated Legendre Polynomial

9.42.1 Usage

Computes the associated Legendre function of degree n.

```
y = legendre(n,x)
```

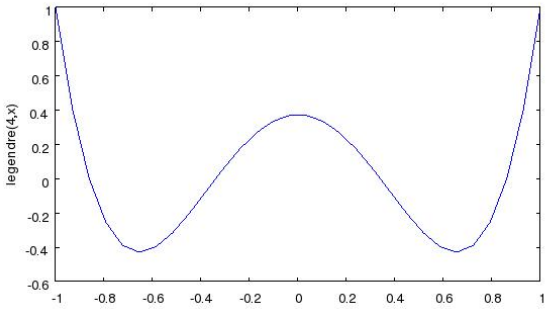
where `x` is either a `float` or `double` array in range `[-1,1]`, `n` is integer scalar. The output vector `y` is the same size (and type) as `x`.

9.42.2 Example

Here is a plot of the `legendre` function over the range `[-1,1]`.

```
--> x = linspace(-1,1,30);
--> y = legendre(4,x);
--> plot(x,y); xlabel('x'); ylabel('legendre(4,x)');
```

which results in the following plot.



9.43 LOG Natural Logarithm Function

9.43.1 Usage

Computes the `log` function for its argument. The general syntax for its use is

```
y = log(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `log` function. Output `y` is of the same size as the input `x`. For strictly positive, real inputs, the output type is the same as the input. For negative and complex arguments, the output is complex.

9.43.2 Function Internals

Mathematically, the `log` function is defined for all real valued arguments `x` by the integral

$$\log x \equiv \int_1^x \frac{d\,t}{t}.$$

For complex-valued arguments, `z`, the complex logarithm is defined as

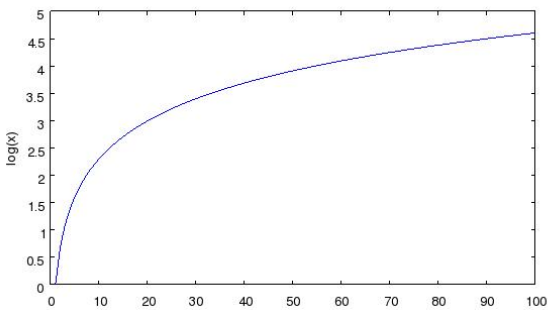
$$\log z \equiv \log |z| + i \arg z,$$

where `arg` is the complex argument of `z`.

9.43.3 Example

The following piece of code plots the real-valued `log` function over the interval `[1,100]`:

```
--> x = linspace(1,100);
--> plot(x,log(x))
--> xlabel('x');
--> ylabel('log(x)');
```



9.44 LOG10 Base-10 Logarithm Function

9.44.1 Usage

Computes the `log10` function for its argument. The general syntax for its use is

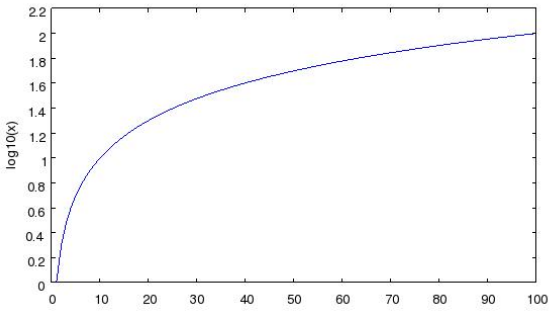
```
y = log10(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `log10` function. Output `y` is of the same size as the input `x`. For strictly positive, real inputs, the output type is the same as the input. For negative and complex arguments, the output is complex.

9.44.2 Example

The following piece of code plots the real-valued `log10` function over the interval `[1,100]`:

```
--> x = linspace(1,100);
--> plot(x,log10(x))
--> xlabel('x');
--> ylabel('log10(x)');
```



9.45 LOG1P Natural Logarithm of 1+P Function

9.45.1 Usage

Computes the `log` function for one plus its argument. The general syntax for its use is

```
y = log1p(x)
```

where `x` is an `n`-dimensional array of numerical type.

9.46 LOG2 Base-2 Logarithm Function

9.46.1 Usage

Computes the `log2` function for its argument. The general syntax for its use is

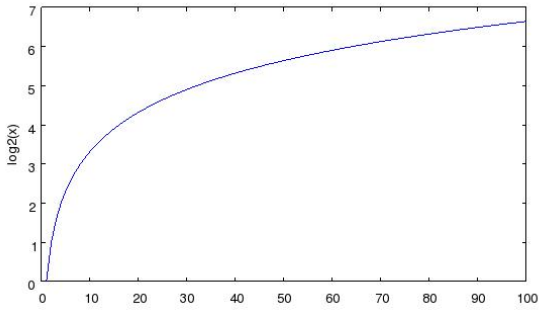
```
y = log2(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `log2` function. Output `y` is of the same size as the input `x`. For strictly positive, real inputs, the output type is the same as the input. For negative and complex arguments, the output is complex.

9.46.2 Example

The following piece of code plots the real-valued `log2` function over the interval `[1,100]`:

```
--> x = linspace(1,100);
--> plot(x,log2(x))
--> xlabel('x');
--> ylabel('log2(x)');
```



9.47 MOD Modulus Operation

9.47.1 Usage

Computes the modulus of an array. The syntax for its use is

```
y = mod(x,n)
```

where `x` is matrix, and `n` is the base of the modulus. The effect of the `mod` operator is to add or subtract multiples of `n` to the vector `x` so that each element `x\_i` is between `0` and `n` (strictly). Note that `n` does not have to be an integer. Also, `n` can either be a scalar (same base for all elements of `x`), or a vector (different base for each element of `x`).

Note that the following are defined behaviors:

1. `mod(x,0) = x`@
2. `mod(x,x) = 0`@
3. `mod(x,n)`@ has the same sign as `n` for all other cases.

9.47.2 Example

The following examples show some uses of `mod` arrays.

```
--> mod(18,12)

ans =
 6

--> mod(6,5)

ans =
 1

--> mod(2*pi,pi)

ans =
 0
```

Here is an example of using `mod` to determine if integers are even or odd:

```
--> mod([1,3,5,2],2)

ans =
 1 1 1 0
```

Here we use the second form of `mod`, with each element using a separate base.

```
--> mod([9 3 2 0],[1 0 2 2])

ans =
 0 3 0 0
```

9.48 MPOWER Matrix Power Function

9.48.1 Usage

Computes the matrix power operator for two arrays. It is an M-file version of the `\^` operator. The syntax for its use is

```
y = mpower(a,b)
```

where `y=a\^b`. See the `matrixpower` documentation for more details on what this function actually does.

9.49 POWER Element-wise Power Function

9.49.1 Usage

Computes the element-wise power operator for two arrays. It is an M-file version of the `.\^` operator. The syntax for its use is

```
y = power(a,b)
```

where `y=a.\^b`. See the `dotpower` documentation for more details on what this function actually does.

9.50 RAD2DEG Radians To Degrees Conversion Function

9.50.1 Usage

Converts the argument array from radians to degrees. The general syntax for its use is

```
y = rad2deg(x)
```

Note that the output type will be the same as the input type, and that complex arguments are allowed. The output is not wrapped to `[0,360)`.

9.50.2 Examples

Some known conversion factors

```
--> rad2deg(1) % one radian is about 57 degrees

ans =
   57.2958

--> rad2deg(pi/4) % should be 45 degrees
```

```
ans =
 45

--> rad2deg(2*pi) % Note that this is 360 not 0 degrees

ans =
 360
```

9.51 REM Remainder After Division

9.51.1 Usage

Computes the remainder after division of an array. The syntax for its use is

```
y = rem(x,n)
```

where `x` is matrix, and `n` is the base of the modulus. The effect of the `rem` operator is to add or subtract multiples of `n` to the vector `x` so that each element `x\_i` is between `0` and `n` (strictly). Note that `n` does not have to be an integer. Also, `n` can either be a scalar (same base for all elements of `x`), or a vector (different base for each element of `x`).

Note that the following are defined behaviors:

1. `rem(x,0) = nan@`
2. `rem(x,x) = 0@` for nonzero `x`
3. `rem(x,n)@` has the same sign as `x` for all other cases.

Note that `rem` and `mod` return the same value if `x` and `n` are of the same sign. But differ by `n` if `x` and `y` have different signs.

9.51.2 Example

The following examples show some uses of `rem` arrays.

```
--> rem(18,12)

ans =
 6

--> rem(6,5)

ans =
 1

--> rem(2*pi,pi)

ans =
 0
```

Here is an example of using `rem` to determine if integers are even or odd:

```
--> rem([1,3,5,2],2)

ans =
 1 1 1 0
```

Here we use the second form of `rem`, with each element using a separate base.

```
--> rem([9 3 2 0],[1 0 2 2])

ans =
         0 NaN         0         0
```

9.52 SEC Trigonometric Secant Function

9.52.1 Usage

Computes the `sec` function for its argument. The general syntax for its use is

```
y = sec(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `sec` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.52.2 Function Internals

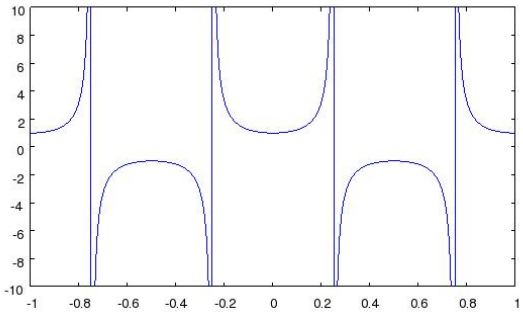
Mathematically, the `sec` function is defined for all arguments as

$$\sec x \equiv \frac{1}{\cos x}.$$

9.52.3 Example

The following piece of code plots the real-valued `sec(2 pi x)` function over the interval of `[-1,1]`:

```
--> t = linspace(-1,1,1000);
--> plot(t,sec(2*pi*t))
--> axis([-1,1,-10,10]);
```



9.53 SECD Secant Degrees Function

9.53.1 Usage

Computes the secant of the argument, but takes the argument in degrees instead of radians (as is the case for `sec`). The syntax for its use is

```
y = secd(x)
```

9.54 SECH Hyperbolic Secant Function

9.54.1 Usage

Computes the hyperbolic secant of the argument. The syntax for its use is

```
y = sech(x)
```

9.54.2 Function Internals

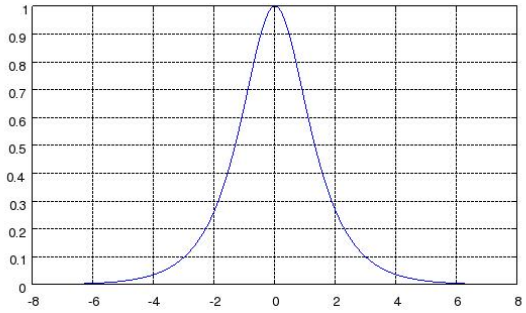
The `sech` function is computed from the formula

$$\operatorname{sech}(x) = \frac{1}{\cosh(x)}$$

9.54.3 Examples

Here is a simple plot of the hyperbolic secant function

```
--> x = -2*pi:.01:2*pi;
--> plot(x,sech(x)); grid('on');
```



9.55 SIN Trigonometric Sine Function

9.55.1 Usage

Computes the `sin` function for its argument. The general syntax for its use is

```
y = sin(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `sin` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.55.2 Function Internals

Mathematically, the `sin` function is defined for all real valued arguments `x` by the infinite summation

$$\sin x \equiv \sum_{n=1}^{\infty} \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!}.$$

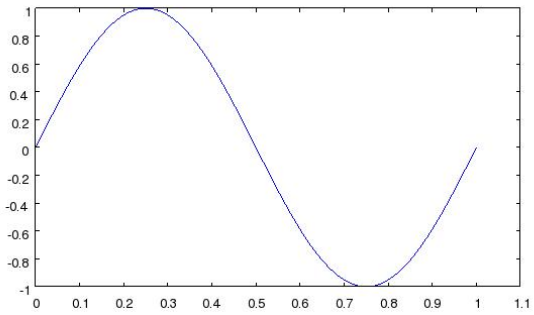
For complex valued arguments `z`, the sine is computed via

$$\sin z \equiv \sin \Re z \cosh \Im z - i \cos \Re z \sinh \Im z.$$

9.55.3 Example

The following piece of code plots the real-valued `sin(2 pi x)` function over one period of `[0,1]`:

```
--> x = linspace(0,1);
--> plot(x,sin(2*pi*x))
```



9.56 SIND Sine Degrees Function

9.56.1 Usage

Computes the sine of the argument, but takes the argument in degrees instead of radians (as is the case for `cos`). The syntax for its use is

```
y = sind(x)
```

9.56.2 Examples

The sine of 45 degrees should be `sqrt(2)/2`

```
--> sind(45)

ans =
    0.7071
```

and the sine of `30` degrees should be 0.5:

```
--> sind(30)

ans =
    0.5000
```

9.57 SINH Hyperbolic Sine Function

9.57.1 Usage

Computes the hyperbolic sine of the argument. The syntax for its use is

```
y = sinh(x)
```

9.57.2 Function Internals

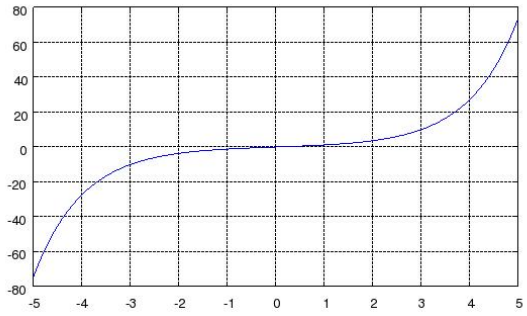
The `sinh` function is computed from the formula

$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

9.57.3 Examples

Here is a simple plot of the hyperbolic sine function

```
--> x = linspace(-5,5);
--> plot(x,sinh(x)); grid('on');
```



9.58 SQRT Square Root of an Array

9.58.1 Usage

Computes the square root of the argument matrix. The general syntax for its use is

```
y = sqrt(x)
```

where `x` is an N-dimensional numerical array.

9.58.2 Example

Here are some examples of using `sqrt`

```
--> sqrt(9)

ans =
 3

--> sqrt(i)

ans =
   0.7071 +  0.7071i

--> sqrt(-1)

ans =
   0.0000 +  1.0000i

--> x = rand(4)

x =
    0.4871    0.5309    0.3343    0.1123
    0.7049    0.6431    0.3320    0.7799
    0.5845    0.8331    0.9892    0.9155
    0.5407    0.9178    0.3408    0.2274
```

```
--> sqrt(x)

ans =
    0.6980    0.7286    0.5782    0.3352
    0.8396    0.8020    0.5762    0.8831
    0.7645    0.9127    0.9946    0.9568
    0.7354    0.9580    0.5838    0.4769
```

9.59 TAN Trigonometric Tangent Function

9.59.1 Usage

Computes the `tan` function for its argument. The general syntax for its use is

```
y = tan(x)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the `tan` function. Output `y` is of the same size and type as the input `x`, (unless `x` is an integer, in which case `y` is a `double` type).

9.59.2 Function Internals

Mathematically, the `tan` function is defined for all real valued arguments `x` by the infinite summation

$$\tan x \equiv x + \frac{x^3}{3} + \frac{2x^5}{15} + \cdots,$$

or alternately by the ratio

$$\tan x \equiv \frac{\sin x}{\cos x}$$

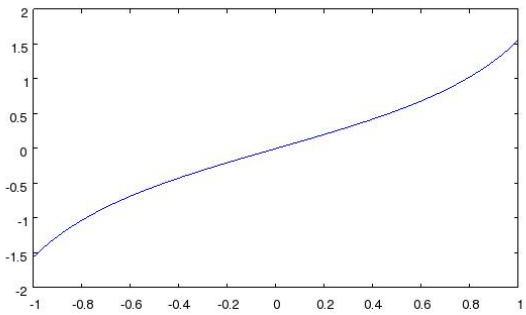
For complex valued arguments `z`, the tangent is computed via

$$\tan z \equiv \frac{\sin 2\Re z + i \sinh 2\Im z}{\cos 2\Re z + \cosh 2\Im z}.$$

9.59.3 Example

The following piece of code plots the real-valued `tan(x)` function over the interval `[-1,1]`:

```
--> t = linspace(-1,1);
--> plot(t,tan(t))
```



9.60 TAND Tangent Degrees Function

9.60.1 Usage

Computes the tangent of the argument, but takes the argument in degrees instead of radians (as is the case for `cos`). The syntax for its use is

```
y = tand(x)
```

9.60.2 Examples

The tangent of 45 degrees should be `1`

```
--> tand(45)

ans =
    1.0000
```

9.61 TANH Hyperbolic Tangent Function

9.61.1 Usage

Computes the hyperbolic tangent of the argument. The syntax for its use is

```
y = tanh(x)
```

9.61.2 Function Internals

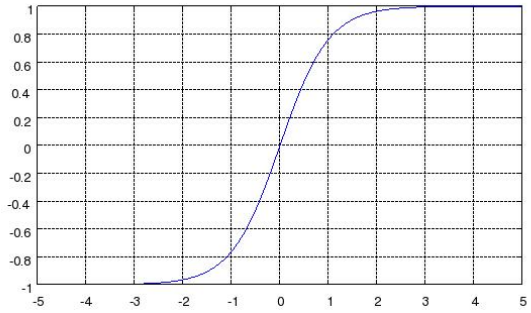
The `tanh` function is computed from the formula

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

9.61.3 Examples

Here is a simple plot of the hyperbolic tangent function

```
--> x = linspace(-5,5);
--> plot(x,tanh(x)); grid('on');
```



Chapter 10

Base Constants

10.1 E Euler Constant (Base of Natural Logarithm)

10.1.1 Usage

Returns a `double` (64-bit floating point number) value that represents Euler's constant, the base of the natural logarithm. Typical usage

```
y = e
```

This value is approximately 2.718281828459045.

10.1.2 Example

The following example demonstrates the use of the `e` function.

```
--> e

ans =
    2.7183

--> log(e)

ans =
 1
```

10.2 EPS Double Precision Floating Point Relative Machine Precision Epsilon

10.2.1 Usage

Returns `eps`, which quantifies the relative machine precision of floating point numbers (a machine specific quantity). The syntax for `eps` is:

```
y = eps
y = eps('double')
y = eps(X)
```

First form returns `eps` for `double` precision values. For most typical processors, this value is approximately `2\^-52`, or 2.2204e-16. Second form return `eps` for class `double` or `single`. Third form returns distance to the next value greater than X.

10.2.2 Example

The following example demonstrates the use of the `eps` function, and one of its numerical consequences.

```
--> eps

ans =
 2.2204e-16

--> 1.0+eps

ans =
    1.0000

--> eps(1000.)

ans =
 1.1369e-13
```

10.3 FALSE Logical False

10.3.1 Usage

Returns a logical 0. The syntax for its use is

```
y = false
```

You can also create an array of logical ones using the syntax

```
y = false(d1,d2,...,dn)
```

or the syntax

```
y = false([d1,d2,...,dn])
```

10.4 FEPS Single Precision Floating Point Relative Machine Precision Epsilon

10.4.1 Usage

Returns `feps`, which quantifies the relative machine precision of floating point numbers (a machine specific quantity). The syntax for `feps` is:

```
y = feps
```

which returns `feps` for `single` precision values. For most typical processors, this value is approximately `2\^-24`, or 5.9604e-8.

10.4.2 Example

The following example demonstrates the use of the `feps` function, and one of its numerical consequences.

```
--> feps

ans =
 1.1921e-07
```

```
--> 1.0f+eps

ans =
 1
```

10.5 I-J Square Root of Negative One

10.5.1 Usage

Returns a `complex` value that represents the square root of -1. There are two functions that return the same value:

```
y = i
```

and

```
y = j.
```

This allows either `i` or `j` to be used as loop indices. The returned value is a 32-bit complex value.

10.5.2 Example

The following examples demonstrate a few calculations with `i`.

```
--> i

ans =
   0.0000 +  1.0000i

--> i^2

ans =
  -1.0000 +  0.0000i
```

The same calculations with `j`:

```
--> j

ans =
   0.0000 +  1.0000i

--> j^2

ans =
  -1.0000 +  0.0000i
```

Here is an example of how `i` can be used as a loop index and then recovered as the square root of -1.

```
--> accum = 0; for i=1:100; accum = accum + i; end; accum

ans =
 5050

--> i

ans =
 100
```

```
--> clear i
--> i

ans =
   0.0000 +  1.0000i
```

10.6 INF Infinity Constant

10.6.1 Usage

Returns a value that represents positive infinity for both 32 and 64-bit floating point values. There are several forms for the `Inf` function. The first form returns a double precision `Inf`.

```
y = inf
```

The next form takes a class name that can be either `'double'`

```
y = inf('double')
```

or `'single'`:

```
y = inf('single')
```

With a single parameter it generates a square matrix of `inf`s.

```
y = inf(n)
```

Alternatively, you can specify the dimensions of the array via

```
y = inf(m,n,p,...)
```

or

```
y = inf([m,n,p,...])
```

Finally, you can add a classname of either `'single'` or `'double'`.

10.6.2 Function Internals

The infinity constant has several interesting properties. In particular:

$$\begin{aligned} \infty \times 0 &= \text{NaN} \\ \infty \times a &= \infty \,\text{forall}\, a > 0 \\ \infty \times a &= -\infty \,\text{forall}\, a < 0 \\ \infty / \infty &= \text{NaN} \\ \infty / 0 &= \infty \end{aligned}$$

Note that infinities are not preserved under type conversion to integer types (see the examples below).

10.6.3 Example

The following examples demonstrate the various properties of the infinity constant.

```
--> inf*0

ans =
 NaN
```

```
--> inf*2

ans =
 Inf

--> inf*-2

ans =
 -Inf

--> inf/inf

ans =
 NaN

--> inf/0

ans =
 Inf

--> inf/nan

ans =
 NaN
```

Note that infinities are preserved under type conversion to floating point types (i.e., `float`, `double`, `complex` and `dcomplex` types), but not integer types.

```
--> uint32(inf)

ans =
 4294967295

--> complex(inf)

ans =
 Inf
```

10.7 PI Constant Pi

10.7.1 Usage

Returns a `double` (64-bit floating point number) value that represents pi (ratio between the circumference and diameter of a circle...). Typical usage

```
y = pi
```

This value is approximately 3.141592653589793.

10.7.2 Example

The following example demonstrates the use of the `pi` function.

```
--> pi

ans =
```

```
    3.1416

--> cos(pi)

ans =
 -1
```

10.8 TEPS Type-based Epsilon Calculation

10.8.1 Usage

Returns `eps` for double precision arguments and `feps` for single precision arguments. The syntax for `teps` is

```
y = teps(x)
```

The `teps` function is most useful if you need to compute epsilon based on the type of the array.

10.8.2 Example

The following example demonstrates the use of the `teps` function, and one of its numerical consequences.

```
--> teps(float(3.4))

ans =
 1.1921e-07

--> teps(complex(3.4+i*2))

ans =
 2.2204e-16
```

10.9 TRUE Logical TRUE

10.9.1 Usage

Returns a logical 1. The syntax for its use is

```
y = true
```

You can also create an array of logical ones using the syntax

```
y = true(d1,d2,...,dn)
```

or the syntax

```
y = true([d1,d2,...,dn])
```

Chapter 11

Elementary Functions

11.1 ABS Absolute Value Function

11.1.1 Usage

Returns the absolute value of the input array for all elements. The general syntax for its use is

```
y = abs(x)
```

where `x` is an `n`-dimensional array of numerical type. The output is the same numerical type as the input, unless the input is `complex` or `dcomplex`. For `complex` inputs, the absolute value is a floating point array, so that the return type is `float`. For `dcomplex` inputs, the absolute value is a double precision floating point array, so that the return type is `double`.

11.1.2 Example

The following demonstrates the `abs` applied to a complex scalar.

```
--> abs(3+4*i)

ans =
 5
```

The `abs` function applied to integer and real values:

```
--> abs([-2,3,-4,5])

ans =
 2 3 4 5
```

For a double-precision complex array,

```
--> abs([2.0+3.0*i,i])

ans =
    3.6056    1.0000
```

11.2 ALL All True Function

11.2.1 Usage

Reduces a logical array along a given dimension by testing for all logical 1s. The general syntax for its use is

```
y = all(x,d)
```

where `x` is an `n`-dimensions array of `logical` type. The output is of `logical` type. The argument `d` is optional, and denotes the dimension along which to operate. The output `y` is the same size as `x`, except that it is singular along the operated direction. So, for example, if `x` is a `3 x 3 x 4` array, and we `all` operation along dimension `d=2`, then the output is of size `3 x 1 x 4`.

11.2.2 Function Internals

The output is computed via

$$y(m_1, \ldots, m_{d-1}, 1, m_{d+1}, \ldots, m_p) = \min_k x(m_1, \ldots, m_{d-1}, k, m_{d+1}, \ldots, m_p)$$

If `d` is omitted, then the minimum is taken over all elements of `x`.

11.2.3 Example

The following piece of code demonstrates various uses of the `all` function

```
--> A = [1,0,0;1,0,0;0,0,1]

A =
 1 0 0
 1 0 0
 0 0 1
```

We start by calling `all` without a dimension argument, in which case it defaults to testing all values of `A`

```
--> all(A)

ans =
 0 0 0
```

The `all` function is useful in expressions also.

```
--> all(A>=0)

ans =
 1 1 1
```

Next, we apply the `all` operation along the rows.

```
--> all(A,2)

ans =
 0
 0
 0
```

11.3 ANY Any True Function

11.3.1 Usage

Reduces a logical array along a given dimension by testing for any logical 1s. The general syntax for its use is

```
y = any(x,d)
```

where `x` is an `n`-dimensions array of `logical` type. The output is of `logical` type. The argument `d` is optional, and denotes the dimension along which to operate. The output `y` is the same size as `x`, except that it is singular along the operated direction. So, for example, if `x` is a `3 x 3 x 4` array, and we `any` operation along dimension `d=2`, then the output is of size `3 x 1 x 4`.

11.3.2 Function Internals

The output is computed via

$$y(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \max_k x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p)$$

If `d` is omitted, then the summation is taken along the first non-singleton dimension of `x`.

11.3.3 Example

The following piece of code demonstrates various uses of the summation function

```
--> A = [1,0,0;1,0,0;0,0,1]

A =
 1 0 0
 1 0 0
 0 0 1
```

We start by calling `any` without a dimension argument, in which case it defaults to the first nonsingular dimension (in this case, along the columns or `d = 1`).

```
--> any(A)

ans =
 1 0 1
```

Next, we apply the `any` operation along the rows.

```
--> any(A,2)

ans =
 1
 1
 1
```

11.4 CEIL Ceiling Function

11.4.1 Usage

Computes the ceiling of an n-dimensional array elementwise. The ceiling of a number is defined as the smallest integer that is larger than or equal to that number. The general syntax for its use is

```
y = ceil(x)
```

where `x` is a multidimensional array of numerical type. The `ceil` function preserves the type of the argument. So integer arguments are not modified, and `float` arrays return `float` arrays as outputs, and similarly for `double` arrays. The `ceil` function is not defined for `complex` or `dcomplex` types.

11.4.2 Example

The following demonstrates the `ceil` function applied to various (numerical) arguments. For integer arguments, the ceil function has no effect:

```
--> ceil(3)

ans =
 3
```

```
--> ceil(-3)

ans =
 -3
```

Next, we take the `ceil` of a floating point value:

```
--> ceil(float(3.023))

ans =
 4

--> ceil(float(-2.341))

ans =
 -2
```

Note that the return type is a `float` also. Finally, for a `double` type:

```
--> ceil(4.312)

ans =
 5

--> ceil(-5.32)

ans =
 -5
```

11.5 CONJ Conjugate Function

11.5.1 Usage

Returns the complex conjugate of the input array for all elements. The general syntax for its use is

```
y = conj(x)
```

where `x` is an `n`-dimensional array of numerical type. The output is the same numerical type as the input. The `conj` function does nothing to real and integer types.

11.5.2 Example

The following demonstrates the complex conjugate applied to a complex scalar.

```
--> conj(3+4*i)

ans =
   3.0000 -  4.0000i
```

The `conj` function has no effect on real arguments:

```
--> conj([2,3,4])

ans =
 2 3 4
```

For a double-precision complex array,

```
--> conj([2.0+3.0*i,i])

ans =
   2.0000 -  3.0000i   0.0000 -  1.0000i
```

11.6 COV Covariance Matrix

11.6.1 Usage

Computes the covariance of a matrix or a vector. The general syntax for its use is

```
y = cov(x)
```

where `x` is a matrix or a vector. If `x` is a vector then `cov` returns the variance of `x`. If `x` is a matrix then `cov` returns the covariance matrix of the columns of `x`. You can also call `cov` with two arguments to compute the matrix of cross correlations. The syntax for this mode is

```
y = cov(x,z)
```

where `x` and `z` are matrices of the same size. Finally, you can provide a normalization flag `d` that is either `0` or `1`, which changes the normalization factor from `L-1` (for `d=0`) to `L` (for `d=1`) where `L` is the number of rows in the matrix `x`. In this case, the syntaxes are

```
y = cov(x,z,d)
```

for the two-argument case, and

```
y = cov(x,d)
```

for the one-argument case.

11.6.2 Example

The following demonstrates some uses of the `cov` function

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1

--> B = [4,-2,0;1,5,2;-2,0,1];
```

We start with the covariance matrix for `A`

```
--> cov(A)

ans =
    4.2222   -1.6667    1.5556
   -1.6667    0.6667   -0.6667
    1.5556   -0.6667    0.8889
```

and again with the (biased) normalization

```
--> cov(A,1)

ans =
    4.2222   -1.6667    1.5556
   -1.6667    0.6667   -0.6667
    1.5556   -0.6667    0.8889
```

Here we compute the cross covariance between `A` and `B`

```
--> cov(A,B)

ans =
    2.0988    1.6667
    1.6667    5.1111
```

and again with biased normalization

```
--> cov(A,B,1)

ans =
    2.0988    1.6667
    1.6667    5.1111
```

11.7 CUMPROD Cumulative Product Function

11.7.1 Usage

Computes the cumulative product of an n-dimensional array along a given dimension. The general syntax for its use is

```
  y = cumprod(x,d)
```

where `x` is a multidimensional array of numerical type, and `d` is the dimension along which to perform the cumulative product. The output `y` is the same size of `x`. Integer types are promoted to `int32`. If the dimension `d` is not specified, then the cumulative sum is applied along the first non-singular dimension.

11.7.2 Function Internals

The output is computed via

$$y(m_1,\ldots,m_{d-1},j,m_{d+1},\ldots,m_p) = \prod_{k=1}^{j} x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p).$$

11.7.3 Example

The default action is to perform the cumulative product along the first non-singular dimension.

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1

--> cumprod(A)

ans =
  5  1  3
 15  2  3
  0  6  3
```

To compute the cumulative product along the columns:

```
--> cumprod(A,2)

ans =
  5  5 15
  3  6  6
  0  0  0
```

The cumulative product also works along arbitrary dimensions

```
--> B(:,:,1) = [5,2;8,9];
--> B(:,:,2) = [1,0;3,0]

B =

(:,:,1) =
 5 2
 8 9

(:,:,2) =
 1 0
 3 0

--> cumprod(B,3)

ans =

(:,:,1) =
  5  2
  8  9

(:,:,2) =
  5  0
 24  0
```

11.8 CUMSUM Cumulative Summation Function

11.8.1 Usage

Computes the cumulative sum of an n-dimensional array along a given dimension. The general syntax for its use is

```
y = cumsum(x,d)
```

where `x` is a multidimensional array of numerical type, and `d` is the dimension along which to perform the cumulative sum. The output `y` is the same size of `x`. Integer types are promoted to `int32`. If the dimension `d` is not specified, then the cumulative sum is applied along the first non-singular dimension.

11.8.2 Function Internals

The output is computed via

$$y(m_1,\ldots,m_{d-1},j,m_{d+1},\ldots,m_p) = \sum_{k=1}^{j} x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p).$$

11.8.3 Example

The default action is to perform the cumulative sum along the first non-singular dimension.

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1

--> cumsum(A)

ans =
 5 1 3
 8 3 4
 8 6 5
```

To compute the cumulative sum along the columns:

```
--> cumsum(A,2)

ans =
 5 6 9
 3 5 6
 0 3 4
```

The cumulative sum also works along arbitrary dimensions

```
--> B(:,:,1) = [5,2;8,9];
--> B(:,:,2) = [1,0;3,0]

B =

(:,:,1) =
 5 2
 8 9

(:,:,2) =
 1 0
 3 0

--> cumsum(B,3)

ans =

(:,:,1) =
  5  2
  8  9

(:,:,2) =
  6  2
 11  9
```

11.9 DEAL Multiple Simultaneous Assignments

11.9.1 Usage

When making a function call, it is possible to assign multiple outputs in a single call, (see, e.g., `max` for an example). The `deal` call allows you to do the same thing with a simple assignment. The syntax for its use is

```
[a,b,c,...] = deal(expr)
```

where `expr` is an expression with multiple values. The simplest example is where `expr` is the dereference of a cell array, e.g. `expr <-- A{:}`. In this case, the `deal` call is equivalent to

```
a = A{1}; b = A{2}; C = A{3};
```

Other expressions which are multivalued are structure arrays with multiple entries (non-scalar), where field dereferencing has been applied.

11.10 DEC2HEX Convert Decimal Number to Hexadecimal

11.10.1 Usage

Converts an integer value into its hexadecimal representation. The syntax for its use is

```
y = dec2hex(x)
```

where `x` is an integer (and is promoted to a 64-bit integer if it is not). The returned value `y` is a string containing the hexadecimal representation of that integer. If you require a minimum length for the hexadecimal representation, you can specify an optional second argument

```
y = dec2hex(x,n)
```

where `n` indicates the minimum number of digits in the representation.

11.10.2 Example

Here are some simple examples:

```
--> dec2hex(1023)

ans =
3FF

--> dec2hex(58128493)

ans =
376F86D
```

11.11 DIFF Difference Function

11.11.1 Usage

```
y=diff(x)
y=diff(x,k)
y=diff(x,k,dim)
```

Produce difference of successive vector elements.

If `x` is a vector of length n, `diff (x)` is the vector of first differences

$$[x\_2 - x\_1, ..., x\_n - x\_n - 1].$$

If `x` is a matrix, `diff (x)` is the matrix of column differences along the first non-singleton dimension.

The second argument is optional. If supplied, `diff (x,k)`, where `k` is a nonnegative integer, returns the `k`-th differences. It is possible that `k` is larger than then first non-singleton dimension of the matrix. In this case, `diff` continues to take the differences along the next non-singleton dimension.

The dimension along which to take the difference can be explicitly stated with the optional variable `dim`. In this case the `k`-th order differences are calculated along this dimension. In the case where `k` exceeds `size (x, dim)` then an empty matrix is returned.

11.12 DOT Dot Product Function

11.12.1 Usage

Computes the scalar dot product of its two arguments. The general syntax for its use is

```
y = dot(x,z)
```

where `x` and `z` are numerical vectors of the same length. If `x` and `z` are multi-dimensional arrays of the same size, then the dot product is taken along the first non-singleton dimension. You can also specify the dimension to take the dot product along using the alternate form

```
y = dot(x,z,dim)
```

where `dim` specifies the dimension to take the dot product along.

11.13 FLOOR Floor Function

11.13.1 Usage

Computes the floor of an n-dimensional array elementwise. The floor of a number is defined as the smallest integer that is less than or equal to that number. The general syntax for its use is

```
y = floor(x)
```

where `x` is a multidimensional array of numerical type. The `floor` function preserves the type of the argument. So integer arguments are not modified, and `float` arrays return `float` arrays as outputs, and similarly for `double` arrays. The `floor` function is not defined for complex types.

11.13.2 Example

The following demonstrates the `floor` function applied to various (numerical) arguments. For integer arguments, the floor function has no effect:

```
--> floor(3)

ans =
 3

--> floor(-3)

ans =
 -3
```

Next, we take the `floor` of a floating point value:

```
--> floor(3.023)

ans =
 3

--> floor(-2.341)

ans =
 -3
```

11.14 GETFIELD Get Field Contents

11.14.1 Usage

Given a structure or structure array, returns the contents of the specified field. The first version is for scalar structures, and has the following syntax

```
y = getfield(x,'fieldname')
```

and is equivalent to `y = x.fieldname` where `x` is a scalar (1 x 1) structure. If `x` is not a scalar structure, then `y` is the first value, i.e., it is equivalent to `y = x(1).fieldname`. The second form allows you to specify a subindex into a structure array, and has the following syntax

```
y = getfield(x, {m,n}, 'fieldname')
```

and is equivalent to `y = x(m,n).fieldname`. You can chain multiple references together using this syntax.

11.15 HEX2DEC Convert Hexadecimal Numbers To Decimal

11.15.1 Usage

Converts a hexadecimal number (encoded as a string matrix) into integers. The syntax for its use is

```
y = hex2dec(x)
```

where `x` is a character matrix where each row represents an integer in hexadecimal form. The output is of type `Double`.

11.15.2 Examples

```
--> hex2dec('3ff')

ans =
 1023
```

Or for a more complex example

```
--> hex2dec(['0ff';'2de';'123'])

ans =
 255
 734
 291
```

11.16 IMAG Imaginary Function

11.16.1 Usage

Returns the imaginary part of the input array for all elements. The general syntax for its use is

```
y = imag(x)
```

where `x` is an `n`-dimensional array of numerical type. The output is the same numerical type as the input, unless the input is `complex` or `dcomplex`. For `complex` inputs, the imaginary part is a floating point array, so that the return type is `float`. For `dcomplex` inputs, the imaginary part is a double precision floating point array, so that the return type is `double`. The `imag` function returns zeros for real and integer types.

11.16.2 Example

The following demonstrates `imag` applied to a complex scalar.

```
--> imag(3+4*i)

ans =
 4
```

The imaginary part of real and integer arguments is a vector of zeros, the same type and size of the argument.

```
--> imag([2,4,5,6])

ans =
 0 0 0 0
```

For a double-precision complex array,

```
--> imag([2.0+3.0*i,i])

ans =
 3 1
```

11.17 IND2SUB Convert Linear Indexing To Multiple Indexing

11.17.1 Usage

The `ind2sub` function converts linear indexing expression into a multi-dimensional indexing expressionl The syntax for its use is

```
[d1, d2, ..., dn] = ind2sub(sizevec,index)
```

where `sizevec` is the size of the array being indexed into, index is the index value. Each `di` is a vector of the same length, containing index values.

11.17.2 Example

Suppose we have a simple `3 x 4` matrix `A` containing some random integer elements

```
--> A = randi(ones(3,4),10*ones(3,4))

A =
  6  6  9  6
 10  1  8  6
  9  1  6  2
```

```
--> [d1 d2] = ind2sub(size(A),7)
d1 =
 1

d2 =
 3

--> A(d1,d2)

ans =
 9
```

11.18 MAX Maximum Function

11.18.1 Usage

Computes the maximum of an array along a given dimension, or alternately, computes two arrays (entry-wise) and keeps the smaller value for each array. As a result, the `max` function has a number of syntaxes. The first one computes the maximum of an array along a given dimension. The first general syntax for its use is either

```
[y,n] = max(x,[],d)
```

where `x` is a multidimensional array of numerical type, in which case the output `y` is the maximum of `x` along dimension `d`. The second argument `n` is the index that results in the maximum. In the event that multiple maxima are present with the same value, the index of the first maximum is used. The second general syntax for the use of the `max` function is

```
[y,n] = max(x)
```

In this case, the maximum is taken along the first non-singleton dimension of `x`. For complex data types, the maximum is based on the magnitude of the numbers. NaNs are ignored in the calculations. The third general syntax for the use of the `max` function is as a comparison function for pairs of arrays. Here, the general syntax is

```
y = max(x,z)
```

where `x` and `z` are either both numerical arrays of the same dimensions, or one of the two is a scalar. In the first case, the output is the same size as both arrays, and is defined elementwise by the smaller of the two arrays. In the second case, the output is defined elementwise by the smaller of the array entries and the scalar.

11.18.2 Function Internals

In the general version of the `max` function which is applied to a single array (using the `max(x,[],d)` or `max(x)` syntaxes), The output is computed via

$$y(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \max_k x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p),$$

and the output array `n` of indices is calculated via

$$n(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \arg\max_k x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p)$$

In the two-array version (`max(x,z)`), the single output is computed as

$$y(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \begin{cases} x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p) & x(\cdots) \leq z(\cdots) \\ z(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p) & z(\cdots) < x(\cdots). \end{cases}$$

11.18.3 Example

The following piece of code demonstrates various uses of the maximum function. We start with the one-array version.

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1
```

We first take the maximum along the columns, resulting in a row vector.

```
--> max(A)

ans =
 5 3 3
```

Next, we take the maximum along the rows, resulting in a column vector.

```
--> max(A,[],2)

ans =
 5
 3
 3
```

When the dimension argument is not supplied, `max` acts along the first non-singular dimension. For a row vector, this is the column direction:

```
--> max([5,3,2,9])

ans =
 9
```

For the two-argument version, we can compute the smaller of two arrays, as in this example:

```
--> a = int8(100*randn(4))

a =
  -16   65  -38  -45
  -33  -46  127  -14
 -110   18  -15  -11
  127 -128 -128 -120

--> b = int8(100*randn(4))

b =
  -60  127 -128   91
   71 -128  -36  -53
    8  127 -106 -128
 -128   47  -93  -34

--> max(a,b)

ans =
 -16 127 -38  91
```

```
  71 -46 127 -14
   8 127 -15 -11
 127  47 -93 -34
```

Or alternately, we can compare an array with a scalar

```
--> a = randn(2)

a =
   -0.0574    1.1346
   -1.3497   -2.3248

--> max(a,0)

ans =
         0    1.1346
         0         0
```

11.19 MEAN Mean Function

11.19.1 Usage

Computes the mean of an array along a given dimension. The general syntax for its use is

```
y = mean(x,d)
```

where `x` is an `n`-dimensions array of numerical type. The output is of the same numerical type as the input. The argument `d` is optional, and denotes the dimension along which to take the mean. The output `y` is the same size as `x`, except that it is singular along the mean direction. So, for example, if `x` is a `3 x 3 x 4` array, and we compute the mean along dimension `d=2`, then the output is of size `3 x 1 x 4`.

11.19.2 Function Internals

The output is computed via

$$y(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \frac{1}{N}\sum_{k=1}^{N} x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p)$$

If `d` is omitted, then the mean is taken along the first non-singleton dimension of `x`.

11.19.3 Example

The following piece of code demonstrates various uses of the mean function

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1
```

We start by calling `mean` without a dimension argument, in which case it defaults to the first nonsingular dimension (in this case, along the columns or `d = 1`).

```
--> mean(A)

ans =
    2.6667    2.0000    1.6667
```

Next, we take the mean along the rows.

```
--> mean(A,2)

ans =
    3.0000
    2.0000
    1.3333
```

11.20 MIN Minimum Function

11.20.1 Usage

Computes the minimum of an array along a given dimension, or alternately, computes two arrays (entry-wise) and keeps the smaller value for each array. As a result, the `min` function has a number of syntaxes. The first one computes the minimum of an array along a given dimension. The first general syntax for its use is either

```
[y,n] = min(x,[],d)
```

where `x` is a multidimensional array of numerical type, in which case the output `y` is the minimum of `x` along dimension `d`. The second argument `n` is the index that results in the minimum. In the event that multiple minima are present with the same value, the index of the first minimum is used. The second general syntax for the use of the `min` function is

```
[y,n] = min(x)
```

In this case, the minimum is taken along the first non-singleton dimension of `x`. For complex data types, the minimum is based on the magnitude of the numbers. NaNs are ignored in the calculations. The third general syntax for the use of the `min` function is as a comparison function for pairs of arrays. Here, the general syntax is

```
y = min(x,z)
```

where `x` and `z` are either both numerical arrays of the same dimensions, or one of the two is a scalar. In the first case, the output is the same size as both arrays, and is defined elementwise by the smaller of the two arrays. In the second case, the output is defined elementwise by the smaller of the array entries and the scalar.

11.20.2 Function Internals

In the general version of the `min` function which is applied to a single array (using the `min(x,[],d)` or `min(x)` syntaxes), The output is computed via

$$y(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \min_k x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p),$$

and the output array `n` of indices is calculated via

$$n(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \arg\min_k x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p)$$

In the two-array version (`min(x,z)`), the single output is computed as

$$y(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \begin{cases} x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p) & x(\cdots) \le z(\cdots) \\ z(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p) & z(\cdots) < x(\cdots). \end{cases}$$

11.20.3 Example

The following piece of code demonstrates various uses of the minimum function. We start with the one-array version.

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1
```

We first take the minimum along the columns, resulting in a row vector.

```
--> min(A)

ans =
 0 1 1
```

Next, we take the minimum along the rows, resulting in a column vector.

```
--> min(A,[],2)

ans =
 1
 1
 0
```

When the dimension argument is not supplied, `min` acts along the first non-singular dimension. For a row vector, this is the column direction:

```
--> min([5,3,2,9])

ans =
 2
```

For the two-argument version, we can compute the smaller of two arrays, as in this example:

```
--> a = int8(100*randn(4))

a =
  -66  -74  -74   32
 -128  -14 -110 -128
  127  -96  -49   72
  127   50   83  120

--> b = int8(100*randn(4))

b =
  -94  108  -99  -35
  127   50 -100  113
  -98  -39 -127 -107
  -12  127  103  -44

--> min(a,b)

ans =
  -94  -74  -99  -35
```

```
 -128  -14 -110 -128
  -98  -96 -127 -107
  -12   50   83  -44
```

Or alternately, we can compare an array with a scalar

```
--> a = randn(2)

a =
    0.7713    0.6716
   -1.0581   -1.3734

--> min(a,0)

ans =
         0         0
   -1.0581   -1.3734
```

11.21 NUM2HEX Convert Numbers to IEEE Hex Strings

11.21.1 Usage

Converts single and double precision arrays to IEEE hex strings. The syntax for its use is

```
y = num2hex(x)
```

where `x` is either a `float` or `double` array. The output `y` is a `n-by-p` character array, where `n` is the number of elements in `x`, and `p` is 16 for `double` arrays, and 8 for `single` arrays.

11.21.2 Example

Some interesting numbers

```
--> num2hex([1 0 0.1 -pi inf nan])

ans =
3ff0000000000000
0000000000000000
3fb999999999999a
c00921fb54442d18
7ff0000000000000
fff8000000000000
```

The same in single precision

```
--> num2hex(float([1 0 0.1 -pi inf nan]))

ans =
3f800000
00000000
3dcccccd
c0490fdb
7f800000
fff80000
```

11.22 PROD Product Function

11.22.1 Usage

Computes the product of an array along a given dimension. The general syntax for its use is

```
y = prod(x,d)
```

where `x` is an `n`-dimensions array of numerical type. The output is of the same numerical type as the input, except for integer types, which are automatically promoted to `int32`. The argument `d` is optional, and denotes the dimension along which to take the product. The output is computed via

$$y(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \prod_k x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p)$$

If `d` is omitted, then the product is taken along the first non-singleton dimension of `x`. Note that by definition (starting with FreeMat 2.1) `prod([]) = 1`.

11.22.2 Example

The following piece of code demonstrates various uses of the product function

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1
```

We start by calling `prod` without a dimension argument, in which case it defaults to the first nonsingular dimension (in this case, along the columns or `d = 1`).

```
--> prod(A)

ans =
 0 6 3
```

Next, we take the product along the rows.

```
--> prod(A,2)

ans =
 15
  6
  0
```

11.23 REAL Real Function

11.23.1 Usage

Returns the real part of the input array for all elements. The general syntax for its use is

```
y = real(x)
```

where `x` is an `n`-dimensional array of numerical type. The output is the same numerical type as the input, unless the input is `complex` or `dcomplex`. For `complex` inputs, the real part is a floating point array, so that the return type is `float`. For `dcomplex` inputs, the real part is a double precision floating point array, so that the return type is `double`. The `real` function does nothing to real and integer types.

11.23.2 Example

The following demonstrates the `real` applied to a complex scalar.

```
--> real(3+4*i)

ans =
 3
```

The `real` function has no effect on real arguments:

```
--> real([2,3,4])

ans =
 2 3 4
```

For a double-precision complex array,

```
--> real([2.0+3.0*i,i])

ans =
 2 0
```

11.24 ROUND Round Function

11.24.1 Usage

Rounds an n-dimensional array to the nearest integer elementwise. The general syntax for its use is

```
y = round(x)
```

where `x` is a multidimensional array of numerical type. The `round` function preserves the type of the argument. So integer arguments are not modified, and `float` arrays return `float` arrays as outputs, and similarly for `double` arrays. The `round` function is not defined for `complex` or `dcomplex` types.

11.24.2 Example

The following demonstrates the `round` function applied to various (numerical) arguments. For integer arguments, the round function has no effect:

```
--> round(3)

ans =
 3

--> round(-3)

ans =
 -3
```

Next, we take the `round` of a floating point value:

```
--> round(3.023f)

ans =
 3

--> round(-2.341f)
```

```
ans =
 -2
```

Note that the return type is a `float` also. Finally, for a `double` type:

```
--> round(4.312)

ans =
 4

--> round(-5.32)

ans =
 -5
```

11.25 STD Standard Deviation Function

11.25.1 Usage

Computes the standard deviation of an array along a given dimension. The general syntax for its use is

```
  y = std(x,d)
```

where `x` is an `n`-dimensions array of numerical type. The output is of the same numerical type as the input. The argument `d` is optional, and denotes the dimension along which to take the variance. The output `y` is the same size as `x`, except that it is singular along the mean direction. So, for example, if `x` is a `3 x 3 x 4` array, and we compute the mean along dimension `d=2`, then the output is of size `3 x 1 x 4`.

11.25.2 Example

The following piece of code demonstrates various uses of the `std` function

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1
```

We start by calling `std` without a dimension argument, in which case it defaults to the first nonsingular dimension (in this case, along the columns or `d = 1`).

```
--> std(A)

ans =
    2.5166    1.0000    1.1547
```

Next, we take the variance along the rows.

```
--> std(A,2)

ans =
    2.0000
    1.0000
    1.5275
```

11.26 SUB2IND Convert Multiple Indexing To Linear Indexing

11.26.1 Usage

The `sub2ind` function converts a multi-dimensional indexing expression into a linear (or vector) indexing expression. The syntax for its use is

```
y = sub2ind(sizevec,d1,d2,...,dn)
```

where `sizevec` is the size of the array being indexed into, and each `di` is a vector of the same length, containing index values. The basic idea behind `sub2ind` is that it makes

```
[z(d1(1),d2(1),...,dn(1)),...,z(d1(n),d2(n),...,dn(n))]
```

equivalent to

```
z(sub2ind(size(z),d1,d2,...,dn))
```

where the later form is using vector indexing, and the former one is using native, multi-dimensional indexing.

11.26.2 Example

Suppose we have a simple `3 x 4` matrix `A` containing some random integer elements

```
--> A = randi(ones(3,4),10*ones(3,4))

A =
  2  3  2  3
 10  2  4  8
  5 10  1  2
```

We can extract the elements `(1,3),(2,3),(3,4)` of `A` via `sub2ind`. To calculate which elements of `A` this corresponds to, we can use `sub2ind` as

```
--> n = sub2ind(size(A),1:3,2:4)

n =
  4  8 12

--> A(n)

ans =
 3 4 2
```

11.27 SUM Sum Function

11.27.1 Usage

Computes the summation of an array along a given dimension. The general syntax for its use is

```
y = sum(x,d)
```

where `x` is an `n`-dimensions array of numerical type. The output is of the same numerical type as the input. The argument `d` is optional, and denotes the dimension along which to take the summation. The output `y` is the same size as `x`, except that it is singular along the summation direction. So, for example, if `x` is a `3 x 3 x 4` array, and we compute the summation along dimension `d=2`, then the output is of size `3 x 1 x 4`.

11.27.2 Function Internals

The output is computed via

$$y(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \sum_k x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p)$$

If `d` is omitted, then the summation is taken along the first non-singleton dimension of `x`.

11.27.3 Example

The following piece of code demonstrates various uses of the summation function

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1
```

We start by calling `sum` without a dimension argument, in which case it defaults to the first nonsingular dimension (in this case, along the columns or `d = 1`).

```
--> sum(A)

ans =
 8 6 5
```

Next, we take the sum along the rows.

```
--> sum(A,2)

ans =
 9
 6
 4
```

11.28 TEST Test Function

11.28.1 Usage

Tests for the argument array to be all logical 1s. It is completely equivalent to the `all` function applied to a vectorized form of the input. The syntax for the `test` function is

```
y = test(x)
```

and the result is equivalent to `all(x(:))`.

11.29 VAR Variance Function

11.29.1 Usage

Computes the variance of an array along a given dimension. The general syntax for its use is

```
y = var(x,d)
```

where `x` is an `n`-dimensions array of numerical type. The output is of the same numerical type as the input. The argument `d` is optional, and denotes the dimension along which to take the variance. The output `y` is the same size as `x`, except that it is singular along the mean direction. So, for example, if `x` is a `3 x 3 x 4` array, and we compute the mean along dimension `d=2`, then the output is of size `3 x 1 x 4`.

11.29.2 Function Internals

The output is computed via

$$y(m_1,\ldots,m_{d-1},1,m_{d+1},\ldots,m_p) = \frac{1}{N-1}\sum_{k=1}^{N}\left(x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p) - \bar{x}\right)^2,$$

where

$$\bar{x} = \frac{1}{N}\sum_{k=1}^{N} x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p)$$

If `d` is omitted, then the mean is taken along the first non-singleton dimension of `x`.

11.29.3 Example

The following piece of code demonstrates various uses of the var function

```
--> A = [5,1,3;3,2,1;0,3,1]

A =
 5 1 3
 3 2 1
 0 3 1
```

We start by calling `var` without a dimension argument, in which case it defaults to the first nonsingular dimension (in this case, along the columns or `d = 1`).

```
--> var(A)

ans =
    6.3333    1.0000    1.3333
```

Next, we take the variance along the rows.

```
--> var(A,2)

ans =
    4.0000
    1.0000
    2.3333
```

11.30 VEC Reshape to a Vector

11.30.1 Usage

Reshapes an n-dimensional array into a column vector. The general syntax for its use is

```
y = vec(x)
```

where `x` is an n-dimensional array (not necessarily numeric). This function is equivalent to the expression `y = x(:)`.

11.30.2 Example

A simple example of the `vec` operator reshaping a 2D matrix:

```
--> A = [1,2,4,3;2,3,4,5]

A =
 1 2 4 3
 2 3 4 5

--> vec(A)

ans =
 1
 2
 2
 3
 4
 4
 3
 5
```

Chapter 12

Inspection Functions

12.1 CLEAR Clear or Delete a Variable

12.1.1 Usage

Clears a set of variables from the current context, or alternately, delete all variables defined in the current context. There are several formats for the function call. The first is the explicit form in which a list of variables are provided:

```
clear a1 a2 ...
```

The variables can be persistent or global, and they will be deleted. The second form

```
clear 'all'
```

clears all variables and libraries from the current context. Alternately, you can use the form:

```
clear 'libs'
```

which will unload any libraries or DLLs that have been `import`ed. Optionally, you can specify that persistent variables should be cleared via:

```
clear 'persistent'
```

and similarly for global variables:

```
clear 'global'
```

You can use

```
clear 'classes'
```

to clear all definitions of user-defined classes. With no arguments, `clear` defaults to clearing `'all'`.

12.1.2 Example

Here is a simple example of using `clear` to delete a variable. First, we create a variable called `a`:

```
--> a = 53

a =
 53
```

Next, we clear `a` using the `clear` function, and verify that it is deleted.

```
--> clear a
--> a
Error: Undefined function or variable a
```

12.2 END End Function

12.2.1 Usage

Computes the size of a variable along a given dimension. The syntax for its use is

```
y = end(x,dim,subindexes)
```

where `x` is the array to be analyzed, `dim` is the dimension along which to compute the end, and `subindexes` indicates how many dimensions are involved in the `end` calculation.

12.3 EXIST Test for Existence

12.3.1 Usage

Tests for the existence of a variable, function, directory or file. The general syntax for its use is

```
y = exist(item,kind)
```

where `item` is a string containing the name of the item to look for, and `kind` is a string indicating the type of the search. The `kind` must be one of

- `'builtin'` checks for built-in functions
- `'dir'` checks for directories
- `'file'` checks for files
- `'var'` checks for variables
- `'all'` checks all possibilities (same as leaving out `kind`)

You can also leave the `kind` specification out, in which case the calling syntax is

```
y = exist(item)
```

The return code is one of the following:

- 0 - if `item` does not exist
- 1 - if `item` is a variable in the workspace
- 2 - if `item` is an M file on the search path, a full pathname to a file, or an ordinary file on your search path
- 5 - if `item` is a built-in FreeMat function
- 7 - if `item` is a directory

Note: previous to version `1.10`, `exist` used a different notion of existence for variables: a variable was said to exist if it was defined and non-empty. This test is now performed by `isset`.

12.3.2 Example

Some examples of the `exist` function. Note that generally `exist` is used in functions to test for keywords. For example,

```
function y = testfunc(a, b, c)
if (~exist('c'))
  % c was not defined, so establish a default
  c = 13;
end
y = a + b + c;
```

An example of exist in action.

```
--> a = randn(3,5,2)

a =

(:,:,1) =
    0.7785    0.6357    1.7582    1.5784   -0.8470
    0.7235    1.0468   -0.6919   -0.6796    0.4767
    0.2100    0.0865    1.5704   -0.1267    2.1381

(:,:,2) =
    1.5525   -0.2908   -1.4220    1.1076    0.2419
    0.1652   -0.5668   -0.8018   -0.5975    0.8483
    0.3147   -0.1109   -0.5203    0.5851    1.1503

--> b = []

b =
  []
--> who
  Variable Name       Type   Flags             Size
              a    double                    [3x5x2]
              b    double                    [0x0]
--> exist('a')

ans =
 1

--> exist('b')

ans =
 1

--> exist('c')

ans =
 0
```

12.4 FIELDNAMES Fieldnames of a Structure

12.4.1 Usage

Returns a cell array containing the names of the fields in a structure array. The syntax for its use is

```
x = fieldnames(y)
```

where y is a structure array of object array. The result is a cell array, with one entry per field in y.

12.4.2 Example

We define a simple structure array:

```
--> y.foo = 3; y.goo = 'hello';
--> x = fieldnames(y)
```

```
x =
 [foo]
 [goo]
```

12.5 ISA Test Type of Variable

12.5.1 Usage

Tests the type of a variable. The syntax for its use is

```
y = isa(x,type)
```

where `x` is the variable to test, and `type` is the type. Supported built-in types are

- `'cell'` for cell-arrays
- `'struct'` for structure-arrays
- `'logical'` for logical arrays
- `'uint8'` for unsigned 8-bit integers
- `'int8'` for signed 8-bit integers
- `'uint16'` for unsigned 16-bit integers
- `'int16'` for signed 16-bit integers
- `'uint32'` for unsigned 32-bit integers
- `'int32'` for signed 32-bit integers
- `'uint64'` for unsigned 64-bit integers
- `'int64'` for signed 64-bit integers
- `'single'` for 32-bit floating point numbers
- `'double'` for 64-bit floating point numbers
- `'char'` for string arrays

If the argument is a user-defined type (via the `class` function), then the name of that class is returned.

12.5.2 Examples

Here are some examples of the `isa` call.

```
--> a = {1}

a =
 [1]

--> isa(a,'char')

ans =
 0

--> isa(a,'cell')

ans =
 1
```

Here we use `isa` along with shortcut boolean evaluation to safely determine if a variable contains the string `'hello'`

```
--> a = 'hello'

a =
hello
--> isa(a,'char') && strcmp(a,'hello')

ans =
 1
```

12.6 ISCELL Test For Cell Array

12.6.1 Usage

The syntax for `iscell` is

```
x = iscell(y)
```

and it returns a logical 1 if the argument is a cell array and a logical 0 otherwise.

12.6.2 Example

Here are some examples of `iscell`

```
--> iscell('foo')

ans =
 0

--> iscell(2)

ans =
 0

--> iscell({1,2,3})

ans =
 1
```

12.7 ISCELLSTR Test For Cell Array of Strings

12.7.1 Usage

The syntax for `iscellstr` is

```
x = iscellstr(y)
```

and it returns a logical 1 if the argument is a cell array in which every cell is a character array (or is empty), and a logical 0 otherwise.

12.7.2 Example

Here is a simple example

```
--> A = {'Hello','Yellow';'Mellow','Othello'}

A =
 [Hello] [Yellow]
 [Mellow] [Othello]

--> iscellstr(A)

ans =
 1
```

12.8 ISCHAR Test For Character Array (string)

12.8.1 Usage

The syntax for `ischar` is

```
   x = ischar(y)
```

and it returns a logical 1 if the argument is a string and a logical 0 otherwise.

12.9 ISEMPTY Test For Variable Empty

12.9.1 Usage

The `isempty` function returns a boolean that indicates if the argument variable is empty or not. The general syntax for its use is

```
  y = isempty(x).
```

12.9.2 Examples

Here are some examples of the `isempty` function

```
--> a = []

a =
  []
--> isempty(a)

ans =
 1

--> b = 1:3

b =
 1 2 3

--> isempty(b)

ans =
 0
```

Note that if the variable is not defined, `isempty` does not return true.

```
--> clear x
--> isempty(x)
Error: Undefined function or variable x
```

12.10 ISEQUAL Test For Matrix Equality

12.10.1 Usage

Test two arrays for equality. The general format for its use is

```
y = isequal(a,b)
```

This function returns true if the two arrays are equal (compared element-wise). Unlike `issame` the `isequal` function will type convert where possible to do the comparison.

12.11 ISEQUALWITHEQUALNANS Test For Matrix Equality

12.11.1 Usage

Test two arrays for equality, with NaNs being equal. The general format for its use is

```
y = isequalwithequalnans(a,b)
```

This function returns true if the two arrays are equal (compared element-wise). Unlike `issame` the `isequalwithequalnans` function will type convert where possible to do the comparison.

12.12 ISFIELD Test for Existence of a Structure Field

12.12.1 Usage

Given a structure array, tests to see if that structure array contains a field with the given name. The syntax for its use is

```
y = isfield(x,field)
```

and returns a logical `1` if `x` has a field with the name `field` and a logical `0` if not. It also returns a logical `0` if the argument `x` is not a structure array.

12.12.2 Example

Here we define a simple struct, and then test for some fields

```
--> a.foo = 32

a =
    foo: 32
--> a.goo = 64

a =
    foo: 32
    goo: 64
--> isfield(a,'goo')

ans =
```

```
 1

--> isfield(a,'got')

ans =
 0

--> isfield(pi,'round')

ans =
 0
```

12.13 ISHANDLE Test for Graphics Handle

12.13.1 Usage

Given a constant, this routine will test to see if the constant is a valid graphics handle or not. The syntax for its use is

```
  y = ishandle(h,type)
```

and returns a logical `1` if `x` is a handle of type `type` and a logical `0` if not.

12.14 ISINF Test for infinities

12.14.1 Usage

Returns true for entries of an array that are infs (i.e., infinities). The usage is

```
   y = isinf(x)
```

The result is a logical array of the same size as `x`, which is true if `x` is not-a-number, and false otherwise. Note that for `complex` or `dcomplex` data types that the result is true if either the real or imaginary parts are infinite.

12.14.2 Example

Suppose we have an array of floats with one element that is `inf`:

```
--> a = [1.2 3.4 inf 5]

a =
    1.2000    3.4000 Inf    5.0000

--> isinf(a)

ans =
 0 0 1 0

--> b = 3./[2 5 0 3 1]

b =
    1.5000    0.6000 Inf    1.0000    3.0000
```

12.15 ISINTTYPE Test For Integer-type Array

12.15.1 Usage

The syntax for `isinttype` is

```
x = isinttype(y)
```

and it returns a logical 1 if the argument is an integer type and a logical 0 otherwise. Note that this function only tests the type of the variable, not the value. So if, for example, `y` is a `float` array containing all integer values, it will still return a logical 0.

12.16 ISLOGICAL Test for Logical Array

12.16.1 Usage

The syntax for `islogical` is

```
x = islogical(y)
```

and it returns a logical 1 if the argument is a logical array and a logical 0 otherwise.

12.17 ISMATRIX Test For a 2D Matrix

12.17.1 Usage

This function tests to see if the argument is a matrix. The syntax for `ismatrix` is

```
x = ismatrix(y)
```

and it returns a logical 1 if the argument is size `N x M` or `M x N` and a logical 0 otherwise.

12.18 ISNAN Test for Not-a-Numbers

12.18.1 Usage

Returns true for entries of an array that are NaN's (i.e., Not-a-Numbers). The usage is

```
y = isnan(x)
```

The result is a logical array of the same size as `x`, which is true if `x` is not-a-number, and false otherwise. Note that for complex data types that the result is true if either the real or imaginary parts are NaNs.

12.18.2 Example

Suppose we have an array of floats with one element that is `nan`:

```
--> a = [1.2 3.4 nan 5]

a =
    1.2000    3.4000 NaN    5.0000

--> isnan(a)

ans =
 0 0 1 0
```

12.19 ISNUMERIC Test for Numeric Array

12.19.1 Usage

The syntax for `isnumeric` is

```
x = isnumeric(y)
```

and it returns a logical 1 if the argument is a numeric (i.e., not a structure array, cell array, string or user defined class), and a logical 0 otherwise.

12.20 ISREAL Test For Real Array

12.20.1 Usage

The syntax for `isreal` is

```
x = isreal(y)
```

and it returns a logical 1 if the argument is real valued and a logical 0 otherwise.

12.21 ISSAME Test If Two Arrays Are Identical

12.21.1 Usage

Tests for two arrays to be identical. The syntax for its use is

```
y = issame(a,b)
```

where `a` and `b` are two arrays to compare. This comparison succeeds only if `a` and `b` are of the same data type, size, and contents. Unlike numerical equivalence tests, the `issame` function considers `NaN` to be equal in both arguments.

12.22 ISSCALAR Test For Scalar

12.22.1 Usage

The syntax for `isscalar` is

```
x = isscalar(y)
```

and it returns a logical 1 if the argument is a scalar, and a logical 0 otherwise.

12.23 ISSET Test If Variable Set

12.23.1 Usage

Tests for the existence and non-emptiness of a variable. the general syntax for its use is

```
y = isset('name')
```

where `name` is the name of the variable to test. This is functionally equivalent to

```
y = exist('name','var') & ~isempty(name)
```

It returns a `logical` 1 if the variable is defined in the current workspace, and is not empty, and returns a 0 otherwise.

12.23.2 Example

Some simple examples of using `isset`

```
--> who
  Variable Name       Type   Flags             Size
--> isset('a')

ans =
 0

--> a = [];
--> isset('a')

ans =
 0

--> a = 2;
--> isset('a')

ans =
 1
```

12.24 ISSPARSE Test for Sparse Matrix

12.24.1 Usage

Test a matrix to see if it is sparse or not. The general format for its use is

```
y = issparse(x)
```

This function returns true if `x` is encoded as a sparse matrix, and false otherwise.

12.24.2 Example

Here is an example of using `issparse`:

```
--> a = [1,0,0,5;0,3,2,0]

a =
 1 0 0 5
 0 3 2 0

--> issparse(a)

ans =
 0

--> A = sparse(a)

A =
 1 1 1
 2 2 3
 2 3 2
 1 4 5
--> issparse(A)
```

```
ans =
 1
```

12.25 ISSQUARE Test For a Square matrix

12.25.1 Usage

This function tests to see if the argument is a square matrix. The syntax for `issquare` is

```
x = issquare(y)
```

and it returns a logical 1 if the argument is size `N x N` logical 0 otherwise.

12.26 ISSTR Test For Character Array (string)

12.26.1 Usage

The syntax for `isstr` is

```
x = isstr(y)
```

and it returns a logical 1 if the argument is a string and a logical 0 otherwise.

12.27 ISSTRUCT Test For Structure Array

12.27.1 Usage

The syntax for `isstruct` is

```
x = isstruct(y)
```

and it returns a logical 1 if the argument is a structure array, and a logical 0 otherwise.

12.28 ISVECTOR Test For a Vector

12.28.1 Usage

This function tests to see if the argument is a vector. The syntax for `isvector` is

```
x = isvector(y)
```

and it returns a logical 1 if the argument is size `N x 1` or `1 x N` and a logical 0 otherwise.

12.29 LENGTH Length of an Array

12.29.1 Usage

Returns the length of an array `x`. The syntax for its use is

```
y = length(x)
```

and is defined as the maximum length of `x` along any of its dimensions, i.e., `max(size(x))`. If you want to determine the number of elements in `x`, use the `numel` function instead.

12.29.2 Example

For a `4 x 4 x 3` matrix, the length is `4`, not `48`, as you might expect.

```
--> x = rand(4,4,3);
--> length(x)

ans =
 4
```

12.30 MAXDIM Maximum Dimension in Array

12.30.1 Usage

The `maxdim` function returns the lowest order dimension along which an array is largest. The general syntax for its use is

```
  n = maxdim(x)
```

and is equivalent to min(find(size(x) == max(size(x)))).

12.31 NDIMS Number of Dimensions in Array

12.31.1 Usage

The `ndims` function returns the number of dimensions allocated in an array. The general syntax for its use is

```
  n = ndims(x)
```

and is equivalent to `length(size(x))`.

12.32 NNZ Number of Nonzeros

12.32.1 Usage

Returns the number of nonzero elements in a matrix. The general format for its use is

```
   y = nnz(x)
```

This function returns the number of nonzero elements in a matrix or array. This function works for both sparse and non-sparse arrays. For

12.32.2 Example

```
--> a = [1,0,0,5;0,3,2,0]

a =
 1 0 0 5
 0 3 2 0

--> nnz(a)

ans =
 4
```

```
--> A = sparse(a)

A =
 1 1 1
 2 2 3
 2 3 2
 1 4 5
--> nnz(A)

ans =
 4
```

12.33 NUMEL Number of Elements in an Array

12.33.1 Usage

Returns the number of elements in an array `x`, or in a subindex expression. The syntax for its use is either

```
y = numel(x)
```

or

```
y = numel(x,varargin)
```

Generally, `numel` returns `prod(size(x))`, the number of total elements in `x`. However, you can specify a number of indexing expressions for `varagin` such as `index1, index2, ..., indexm`. In that case, the output of `numel` is `prod(size(x(index1,...,indexm)))`.

12.33.2 Example

For a `4 x 4 x 3` matrix, the length is `4`, not `48`, as you might expect, but `numel` is `48`.

```
--> x = rand(4,4,3);
--> length(x)

ans =
 4

--> numel(x)

ans =
 48
```

Here is an example of using `numel` with indexing expressions.

```
--> numel(x,1:3,1:2,2)

ans =
 6
```

12.34 SIZE Size of a Variable

12.34.1 Usage

Returns the size of a variable. There are two syntaxes for its use. The first syntax returns the size of the array as a vector of integers, one integer for each dimension

```
[d1,d2,...,dn] = size(x)
```

The other format returns the size of `x` along a particular dimension:

```
d = size(x,n)
```

where `n` is the dimension along which to return the size.

12.34.2 Example

```
--> a = randn(23,12,5);
--> size(a)

ans =
 23 12  5
```

Here is an example of the second form of `size`.

```
--> size(a,2)

ans =
 12
```

12.35 TYPEOF Determine the Type of an Argument

12.35.1 Usage

Returns a string describing the type of an array. The syntax for its use is

```
y = typeof(x),
```

The returned string is one of

- `'cell'` for cell-arrays
- `'struct'` for structure-arrays
- `'logical'` for logical arrays
- `'uint8'` for unsigned 8-bit integers
- `'int8'` for signed 8-bit integers
- `'uint16'` for unsigned 16-bit integers
- `'int16'` for signed 16-bit integers
- `'uint32'` for unsigned 32-bit integers
- `'int32'` for signed 32-bit integers
- `'float'` for 32-bit floating point numbers
- `'double'` for 64-bit floating point numbers
- `'string'` for string arrays

12.35.2 Example

The following piece of code demonstrates the output of the `typeof` command for each possible type. The first example is with a simple cell array.

```
--> typeof({1})

ans =
cell
```

The next example uses the `struct` constructor to make a simple scalar struct.

```
--> typeof(struct('foo',3))

ans =
struct
```

The next example uses a comparison between two scalar integers to generate a scalar logical type.

```
--> typeof(3>5)

ans =
logical
```

For the integers, the typecast operations are used to generate the arguments.

```
--> typeof(uint8(3))

ans =
uint8
--> typeof(int8(8))

ans =
int8
--> typeof(uint16(3))

ans =
uint16
--> typeof(int16(8))

ans =
int16
--> typeof(uint32(3))

ans =
uint32
--> typeof(int32(3))

ans =
int32
--> typeof(uint64(3))

ans =
uint64
--> typeof(int64(3))

ans =
int64
```

Float, and double can be created using the suffixes.

```
--> typeof(1.0f)

ans =
single
--> typeof(1.0D)

ans =
double
--> typeof(1.0f+i)

ans =
single
--> typeof(1.0D+2.0D*i)

ans =
double
```

12.36 WHAT List FreeMat Files In Directory

12.36.1 Usage

Lists files in a directory (or the current directory if no argument is supplied) that are relevant to FreeMat. These are `M`-files, `MAT`-files, and class directories. There are several syntaxes for its use. The first is

```
what
```

which lists the aforementioned items. If you provide a path instead

```
what path-to-folder
```

then `what` will list the relevant FreeMat items in the specified directory.

12.37 WHERE Get Information on Program Stack

12.37.1 Usage

Returns information on the current stack. The usage is

```
where
```

The result is a kind of stack trace that indicates the state of the current call stack, and where you are relative to the stack.

12.37.2 Example

Suppose we have the following chain of functions.

```
    chain1.m
function chain1
  a = 32;
  b = a + 5;
  chain2(b)
```

```
    chain2.m
function chain2(d)
  d = d + 5;
  chain3
```

```
    chain3.m
function chain3
  g = 54;
  f = g + 1;
  keyboard
```

The execution of the `where` command shows the stack trace.

```
--> chain1
[chain3,4]--> where
In /home/sbasu/Devel/FreeMat/help/tmp/chain3.m(chain3) at line 4
    In /home/sbasu/Devel/FreeMat/help/tmp/chain2.m(chain2) at line 4
    In /home/sbasu/Devel/FreeMat/help/tmp/chain1.m(chain1) at line 4
    In scratch() at line 2
    In base(base)
    In base()
    In global()
[chain3,4]
```

12.38 WHICH Get Information on Function

12.38.1 Usage

Returns information on a function (if defined). The usage is

```
which(fname)
```

where `fname` is a `string` argument that contains the name of the function. For functions and scripts defined via `.m` files, the `which` command returns the location of the source file:

```
y = which(fname)
```

will return the filename for the `.m` file corresponding to the given function, and an empty string otherwise.

12.38.2 Example

First, we apply the `which` command to a built in function.

```
--> which fft
Function fft is a built in function
```

Next, we apply it to a function defined via a `.m` file.

```
--> which fliplr
Function fliplr, M-File function in file '/home/sbasu/Devel/FreeMat/src/toolbox/array/fliplr.m'
```

12.39 WHO Describe Currently Defined Variables

12.39.1 Usage

Reports information on either all variables in the current context or on a specified set of variables. For each variable, the `who` function indicates the size and type of the variable as well as if it is a global or persistent. There are two formats for the function call. The first is the explicit form, in which a list of variables are provided:

```
who a1 a2 ...
```

In the second form

```
who
```

the `who` function lists all variables defined in the current context (as well as global and persistent variables). Note that there are two alternate forms for calling the `who` function:

```
who 'a1' 'a2' ...
```

and

```
who('a1','a2',...)
```

12.39.2 Example

Here is an example of the general use of `who`, which lists all of the variables defined.

```
--> c = [1,2,3];
--> f = 'hello';
--> p = randn(1,256);
--> who
  Variable Name       Type   Flags             Size
              c    double                    [1x3]
              f      char                    [1x5]
              p    double                    [1x256]
```

In the second case, we examine only a specific variable:

```
--> who c
  Variable Name       Type   Flags             Size
              c    double                    [1x3]
--> who('c')
  Variable Name       Type   Flags             Size
              c    double                    [1x3]
```

12.40 WHOS Describe Currently Defined Variables With Memory Usage

12.40.1 Usage

Reports information on either all variables in the current context or on a specified set of variables. For each variable, the `who` function indicates the size and type of the variable as well as if it is a global or persistent. There are two formats for the function call. The first is the explicit form, in which a list of variables are provided:

```
whos a1 a2 ...
```

In the second form

```
whos
```

the `whos` function lists all variables defined in the current context (as well as global and persistent variables). Note that there are two alternate forms for calling the `whos` function:

```
whos 'a1' 'a2' ...
```

and

```
whos('a1','a2',...)
```

Chapter 13

Type Conversion Functions

13.1 BIN2DEC Convert Binary String to Decimal

13.1.1 USAGE

Converts a binary string to an integer. The syntax for its use is

```
y = bin2dec(x)
```

where `x` is a binary string. If `x` is a matrix, then the resulting `y` is a column vector.

13.1.2 Example

Here we convert some numbers to bits

```
--> bin2dec('101110')

ans =
 46

--> bin2dec('010')

ans =
 2
```

13.2 BIN2INT Convert Binary Arrays to Integer

13.2.1 Usage

Converts the binary decomposition of an integer array back to an integer array. The general syntax for its use is

```
y = bin2int(x)
```

where `x` is a multi-dimensional logical array, where the last dimension indexes the bit planes (see `int2bin` for an example). By default, the output of `bin2int` is unsigned `uint32`. To get a signed integer, it must be typecast correctly. A second form for `bin2int` takes a `'signed'` flag

```
y = bin2int(x,'signed')
```

in which case the output is signed.

13.2.2 Example

The following piece of code demonstrates various uses of the int2bin function. First the simplest example:

```
--> A = [2;5;6;2]

A =
 2
 5
 6
 2

--> B = int2bin(A,8)

B =
 0 0 0 0 0 0 1 0
 0 0 0 0 0 1 0 1
 0 0 0 0 0 1 1 0
 0 0 0 0 0 0 1 0

--> bin2int(B)

ans =
 2
 5
 6
 2

--> A = [1;2;-5;2]

A =
  1
  2
 -5
  2

--> B = int2bin(A,8)

B =
 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 1 0
 1 1 1 1 1 0 1 1
 0 0 0 0 0 0 1 0

--> bin2int(B)

ans =
   1
   2
 251
   2

--> int32(bin2int(B))

ans =
```

```
  1
  2
251
  2
```

13.2.3 Tets

13.3 CAST Typecast Variable to Specified Type

13.3.1 Usage

The `cast` function allows you to typecast a variable from one type to another. The syntax for its use is

```
y = cast(x,toclass)
```

where `toclass` is the name of the class to cast `x` to. Note that the typecast must make sense, and that `toclass` must be one of the builtin types. The current list of supported types is

- `'cell'` for cell-arrays
- `'struct'` for structure-arrays
- `'logical'` for logical arrays
- `'uint8'` for unsigned 8-bit integers
- `'int8'` for signed 8-bit integers
- `'uint16'` for unsigned 16-bit integers
- `'int16'` for signed 16-bit integers
- `'uint32'` for unsigned 32-bit integers
- `'int32'` for signed 32-bit integers
- `'uint64'` for unsigned 64-bit integers
- `'int64'` for signed 64-bit integers
- `'float'` for 32-bit floating point numbers
- `'single'` is a synonym for `'float'`
- `'double'` for 64-bit floating point numbers
- `'char'` for string arrays

13.3.2 Example

Here is an example of a typecast from a float to an 8-bit integer

```
--> cast(pi,'uint8')

ans =
 3
```

and here we cast an array of arbitrary integers to a logical array

```
--> cast([1 0 3 0],'logical')

ans =
 1 0 1 0
```

13.4 CHAR Convert to character array or string

13.4.1 Usage

The `char` function can be used to convert an array into a string. It has several forms. The first form is

```
y = char(x)
```

where `x` is a numeric array containing character codes. FreeMat does not currently support Unicode, so the character codes must be in the range of `[0,255]`. The output is a string of the same size as `x`. A second form is

```
y = char(c)
```

where `c` is a cell array of strings, creates a matrix string where each row contains a string from the corresponding cell array. The third form is

```
y = char(s1, s2, s3, ...)
```

where `si` are a character arrays. The result is a matrix string where each row contains a string from the corresponding argument.

13.4.2 Example

Here is an example of the first technique being used to generate a string containing some ASCII characters

```
--> char([32:64;65:97])

ans =
 !"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`a
```

In the next example, we form a character array from a set of strings in a cell array. Note that the character array is padded with spaces to make the rows all have the same length.

```
--> char({'hello','to','the','world'})

ans =
hello
to
the
world
```

In the last example, we pass the individual strings as explicit arguments to `char`

```
--> char('hello','to','the','world')

ans =
hello
to
the
world
```

13.5 COMPLEX Create a Complex Number

13.5.1 Usage

Converts the two real input arguments into the real and imaginary part (respectively) of a complex number. The syntax for its use is

```
y = complex(x,z)
```

where `x` and `z` are `n`-dimensional numerical arrays. The usual rules for binary operators apply (i.e., one of the arguments can be a scalar, if either is of type `single` the output is single, etc.).

13.6 DCOMPLEX Convert to Double Precision (deprecated)

13.6.1 Usage

The `dcomplex` function used to convert variables into 64-bit complex data types in prior versions of FreeMat. Starting with FreeMat 4, the type rules are the same as Matlab, hence, there is no distinction between a 64-bit complex type and and 64-bit real type. Thus, the `dcomplex` function is just a synonym for `double`.

13.7 DEC2BIN Convert Decimal to Binary String

13.7.1 USAGE

Converts an integer to a binary string. The syntax for its use is

```
y = dec2bin(x,n)
```

where `x` is the positive integer, and `n` is the number of bits to use in the representation. Alternately, if you leave `n` unspecified,

```
y = dec2bin(x)
```

the minimum number of bits needed to represent `x` are used. If `x` is a vector, then the resulting `y` is a character matrix.

13.7.2 Example

Here we convert some numbers to bits

```
--> dec2bin(56)

ans =
111000
--> dec2bin(1039456)

ans =
11111101110001100000
--> dec2bin([63,73,32],5)

ans =
11111
01001
00000
```

13.8 DOUBLE Convert to 64-bit Floating Point

13.8.1 Usage

Converts the argument to a 64-bit floating point number. The syntax for its use is

```
y = double(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows the saturation rules. Note that both `NaN` and `Inf` are both preserved under type conversion.

13.8.2 Example

The following piece of code demonstrates several uses of `double`. First, we convert from an integer (the argument is an integer because no decimal is present):

```
--> double(200)

ans =
 200
```

In the next example, a single precision argument is passed in (the presence of the `f` suffix implies single precision).

```
--> double(400.0f)

ans =
 400
```

In the next example, a complex argument is passed in.

```
--> double(3.0+4.0*i)

ans =
   3.0000 +  4.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> double('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> double({4})
Error: Cannot perform type conversions with this type
```

13.9 FLOAT Convert to 32-bit Floating Point

13.9.1 Usage

Converts the argument to a 32-bit floating point number. The syntax for its use is

```
y = float(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows the saturation rules. Note that both `NaN` and `Inf` are both preserved under type conversion.

13.9.2 Example

The following piece of code demonstrates several uses of `float`. First, we convert from an integer (the argument is an integer because no decimal is present):

```
--> float(200)

ans =
 200
```

In the next example, a double precision argument is passed in

```
--> float(400.0)

ans =
 400
```

In the next example, a complex argument is passed in.

```
--> float(3.0+4.0*i)

ans =
   3.0000 +  4.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> float('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> float({4})
Error: Cannot perform type conversions with this type
```

13.10 INT16 Convert to Signed 16-bit Integer

13.10.1 Usage

Converts the argument to an signed 16-bit Integer. The syntax for its use is

```
y = int16(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows the saturation rules (e.g., if `x` is outside the normal range for a signed 16-bit integer of `[-32767,32767]`, it is truncated to that range). Note that both `NaN` and `Inf` both map to 0.

13.10.2 Example

The following piece of code demonstrates several uses of `int16`. First, the routine uses

```
--> int16(100)

ans =
 100

--> int16(-100)

ans =
 -100
```

In the next example, an integer outside the range of the type is passed in. The result is truncated to the range of the data type.

```
--> int16(40000)

ans =
 32767
```

In the next example, a positive double precision argument is passed in. The result is the signed integer that is closest to the argument.

```
--> int16(pi)

ans =
 3
```

In the next example, a complex argument is passed in. The result is the signed complex integer that is closest to the argument.

```
--> int16(5+2*i)

ans =
   5.0000 +  2.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> int16('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> int16({4})
Error: Cannot perform type conversions with this type
```

13.11 INT2BIN Convert Integer Arrays to Binary

13.11.1 Usage

Computes the binary decomposition of an integer array to the specified number of bits. The general syntax for its use is

```
y = int2bin(x,n)
```

where `x` is a multi-dimensional integer array, and `n` is the number of bits to expand it to. The output array `y` has one extra dimension to it than the input. The bits are expanded along this extra dimension.

13.11.2 Example

The following piece of code demonstrates various uses of the int2bin function. First the simplest example:

```
--> A = [2;5;6;2]

A =
 2
 5
 6
 2

--> int2bin(A,8)

ans =
 0 0 0 0 0 0 1 0
 0 0 0 0 0 1 0 1
```

```
 0 0 0 0 0 1 1 0
 0 0 0 0 0 0 1 0

--> A = [1;2;-5;2]

A =
  1
  2
 -5
  2

--> int2bin(A,8)

ans =
 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 1 0
 1 1 1 1 1 0 1 1
 0 0 0 0 0 0 1 0
```

13.12 INT32 Convert to Signed 32-bit Integer

13.12.1 Usage

Converts the argument to an signed 32-bit Integer. The syntax for its use is

```
y = int32(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows the saturation rules (e.g., if `x` is outside the normal range for a signed 32-bit integer of `[-2147483647,2147483647]`, it is truncated to that range). Note that both `NaN` and `Inf` both map to 0.

13.12.2 Example

The following piece of code demonstrates several uses of `int32`. First, the routine uses

```
--> int32(100)

ans =
 100

--> int32(-100)

ans =
 -100
```

In the next example, an integer outside the range of the type is passed in. The result is truncated to the range of the data type.

```
--> int32(40e9)

ans =
 2147483647
```

In the next example, a positive double precision argument is passed in. The result is the signed integer that is closest to the argument.

```
--> int32(pi)

ans =
 3
```

In the next example, a complex argument is passed in. The result is the signed complex integer that is closest to the argument.

```
--> int32(5+2*i)

ans =
   5.0000 +  2.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> int32('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> int32({4})
Error: Cannot perform type conversions with this type
```

13.13 INT64 Convert to Signed 64-bit Integer

13.13.1 Usage

Converts the argument to an signed 64-bit Integer. The syntax for its use is

```
y = int64(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows the saturation rules (e.g., if `x` is outside the normal range for a signed 64-bit integer of `[-2\^63+1,2\^63-1]`, it is truncated to that range). Note that both `NaN` and `Inf` both map to 0.

13.13.2 Example

The following piece of code demonstrates several uses of `int64`. First, the routine uses

```
--> int64(100)

ans =
 100

--> int64(-100)

ans =
 -100
```

In the next example, an integer outside the range of the type is passed in. The result is truncated to the range of the data type.

```
--> int64(40e9)

ans =
 40000000000
```

In the next example, a positive double precision argument is passed in. The result is the signed integer that is closest to the argument.

```
--> int64(pi)

ans =
 3
```

In the next example, a complex argument is passed in. The result is the complex signed integer that is closest to the argument.

```
--> int64(5+2*i)

ans =
   5.0000 +  2.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> int64('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> int64({4})
Error: Cannot perform type conversions with this type
```

13.14 INT8 Convert to Signed 8-bit Integer

13.14.1 Usage

Converts the argument to an signed 8-bit Integer. The syntax for its use is

```
y = int8(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows the saturation rules (e.g., if `x` is outside the normal range for a signed 8-bit integer of `[-127,127]`, it is truncated to that range. Note that both `NaN` and `Inf` both map to 0.

13.14.2 Example

The following piece of code demonstrates several uses of `int8`. First, the routine uses

```
--> int8(100)

ans =
 100

--> int8(-100)

ans =
 -100
```

In the next example, an integer outside the range of the type is passed in. The result is truncated to the range of the type.

```
--> int8(400)

ans =
 127
```

In the next example, a positive double precision argument is passed in. The result is the signed integer that is closest to the argument.

```
--> int8(pi)

ans =
 3
```

In the next example, a complex argument is passed in. The result is the signed complex integer that is closest to the argument.

```
--> int8(5+2*i)

ans =
   5.0000 +  2.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> int8('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> int8({4})
Error: Cannot perform type conversions with this type
```

13.15 LOGICAL Convert to Logical

13.15.1 Usage

Converts the argument to a logical array. The syntax for its use is

```
y = logical(x)
```

where `x` is an `n`-dimensional numerical array. Any nonzero element maps to a logical 1.

13.15.2 Example

Here we convert an integer array to `logical`:

```
--> logical([1,2,3,0,0,0,5,2,2])

ans =
 1 1 1 0 0 0 1 1 1
```

The same example with double precision values:

```
--> logical([pi,pi,0,e,0,-1])

ans =
 1 1 0 1 0 1
```

13.16 SINGLE Convert to 32-bit Floating Point

13.16.1 Usage

A synonym for the `float` function, converts the argument to a 32-bit floating point number. The syntax for its use is

```
y = single(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows the general C rules. Note that both `NaN` and `Inf` are both preserved under type conversion.

13.17 STRING Convert Array to String

13.17.1 Usage

Converts the argument array into a string. The syntax for its use is

```
y = string(x)
```

where `x` is an `n`-dimensional numerical array.

13.17.2 Example

Here we take an array containing ASCII codes for a string, and convert it into a string.

```
--> a = [104,101,108,108,111]

a =
 104 101 108 108 111

--> string(a)

ans =
hello
```

13.18 UINT16 Convert to Unsigned 16-bit Integer

13.18.1 Usage

Converts the argument to an unsigned 16-bit Integer. The syntax for its use is

```
y = uint16(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows saturation rules (e.g., if `x` is outside the normal range for an unsigned 16-bit integer of `[0,65535]`, it is truncated to that range. Note that both `NaN` and `Inf` both map to 0.

13.18.2 Example

The following piece of code demonstrates several uses of `uint16`.

```
--> uint16(200)

ans =
 200
```

In the next example, an integer outside the range of the type is passed in. The result is truncated to the maximum value of the data type.

```
--> uint16(99400)

ans =
 65535
```

In the next example, a negative integer is passed in. The result is truncated to zero.

```
--> uint16(-100)

ans =
 0
```

In the next example, a positive double precision argument is passed in. The result is the unsigned integer that is closest to the argument.

```
--> uint16(pi)

ans =
 3
```

In the next example, a complex argument is passed in. The result is the complex unsigned integer that is closest to the argument.

```
--> uint16(5+2*i)

ans =
   5.0000 +  2.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> uint16('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> uint16({4})
Error: Cannot perform type conversions with this type
```

13.19 UINT32 Convert to Unsigned 32-bit Integer

13.19.1 Usage

Converts the argument to an unsigned 32-bit Integer. The syntax for its use is

```
y = uint32(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows saturation rules (e.g., if `x` is outside the normal range for an unsigned 32-bit integer of `[0,4294967295]`, it is truncated to that range. Note that both `NaN` and `Inf` both map to 0.

13.19.2 Example

The following piece of code demonstrates several uses of `uint32`.

```
--> uint32(200)

ans =
 200
```

In the next example, an integer outside the range of the type is passed in. The result is truncated to the maximum value of the data type.

```
--> uint32(40e9)

ans =
 4294967295
```

In the next example, a negative integer is passed in. The result is truncated to zero.

```
--> uint32(-100)

ans =
 0
```

In the next example, a positive double precision argument is passed in. The result is the unsigned integer that is closest to the argument.

```
--> uint32(pi)

ans =
 3
```

In the next example, a complex argument is passed in. The result is the complex unsigned integer that is closest to the argument.

```
--> uint32(5+2*i)

ans =
   5.0000 +  2.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> uint32('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> uint32({4})
Error: Cannot perform type conversions with this type
```

13.20 UINT64 Convert to Unsigned 64-bit Integer

13.20.1 Usage

Converts the argument to an unsigned 64-bit Integer. The syntax for its use is

```
y = uint64(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows saturation rules (e.g., if `x` is outside the normal range for an unsigned 64-bit integer of `[0,2\^64-1]`, it is truncated to that range. Note that both `NaN` and `Inf` both map to 0.

13.20.2 Example

The following piece of code demonstrates several uses of `uint64`.

```
--> uint64(200)

ans =
 200
```

In the next example, an integer outside the range of the type is passed in. The result is truncated to the maximum value of the data type.

```
--> uint64(40e9)

ans =
 40000000000
```

In the next example, a negative integer is passed in. The result is zero.

```
--> uint64(-100)

ans =
 0
```

In the next example, a positive double precision argument is passed in. The result is the unsigned integer that is closest to the argument.

```
--> uint64(pi)

ans =
 3
```

In the next example, a complex argument is passed in. The result is the complex unsigned integer that is closest to the argument.

```
--> uint64(5+2*i)

ans =
   5.0000 +  2.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> uint64('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> uint64({4})
Error: Cannot perform type conversions with this type
```

13.21 UINT8 Convert to Unsigned 8-bit Integer

13.21.1 Usage

Converts the argument to an unsigned 8-bit Integer. The syntax for its use is

```
y = uint8(x)
```

where `x` is an `n`-dimensional numerical array. Conversion follows saturation rules (e.g., if `x` is outside the normal range for an unsigned 8-bit integer of `[0,255]`, it is truncated to that range. Note that both `NaN` and `Inf` both map to 0.

13.21.2 Example

The following piece of code demonstrates several uses of `uint8`.

```
--> uint8(200)

ans =
 200
```

In the next example, an integer outside the range of the type is passed in. The result is truncated to the maximum value of the data type.

```
--> uint8(400)

ans =
 255
```

In the next example, a negative integer is passed in. The result is trunated to zero.

```
--> uint8(-100)

ans =
 0
```

In the next example, a positive double precision argument is passed in. The result is the unsigned integer that is closest to the argument.

```
--> uint8(pi)

ans =
 3
```

In the next example, a complex argument is passed in. The result is complex unsigned integer that is closest to the argument.

```
--> uint8(5+2*i)

ans =
   5.0000 +  2.0000i
```

In the next example, a string argument is passed in. The string argument is converted into an integer array corresponding to the ASCII values of each character.

```
--> uint8('helo')

ans =
 104 101 108 111
```

In the last example, a cell-array is passed in. For cell-arrays and structure arrays, the result is an error.

```
--> uint8({4})
Error: Cannot perform type conversions with this type
```

Chapter 14

Array Generation and Manipulations

14.1 ARRAYFUN Apply a Function To Elements of an Array

14.1.1 Usage

The `arrayfun` function is used to apply a function handle to each element of an input array (or arrays), and to collect the outputs into an array. The general syntax for its use is

```
y = arrayfun(fun, x)
```

where `x` is an N-dimensional array. In this case, each element of the output `y\_i` is defined as `fun(x\_i)`. You can also supply multiple arguments to `arrayfun`, provided all of the arguments are the same size

```
y = arrayfun(fun, x, z,...)
```

in which case each output `y\_i = fun(x\_i,z\_i,...)`.

If the function returns multiple outputs, then `arrayfun` can be called with multiple outputs, in which case each output goes to a separate array output

```
[y1,y2,...] = arrayfun(fun, x, z, ...)
```

The assumption is that the output types for each call to `fun` is the same across the inputs.

Finally, some hints can be provided to `arrayfun` using the syntax

```
[y1,y2,...] = arrayfun(fun, x, z, ..., 'param', value, 'param', value)
```

where `param` and `value` take on the following possible values:

- `'UniformOutput'` - if the `value` is `true` then each output of `fun` must be a scalar, and the outputs are concatenated into an array the same size as the input arrays. If the `value` is `false` then the outputs are encapsulated into a cell array, with each entry in the cell array containing the call to `fun(x\_i,z\_i,...)`.
- `'ErrorHandler'` - in this case `value` is a function handle that gets called when `fun` throws an error. If `'ErrorHandler'` is not specified, then `arrayfun` allows the error to propogate (i.e., and exception is thrown).

14.2 ASSIGN Making assignments

14.2.1 Usage

FreeMat assignments take a number of different forms, depending on the type of the variable you want to make an assignment to. For numerical arrays and strings, the form of an assignment is either

```
a(ndx) = val
```

where `ndx` is a set of vector indexing coordinates. This means that the values `ndx` takes reference the elements of `a` in column order. So, if, for example `a` is an `N x M` matrix, the first column has vector indices `1,2,...,N`, and the second column has indices `N+1,N+2,...,2N`, and so on. Alternately, you can use multi-dimensional indexing to make an assignment:

```
a(ndx_1,ndx_2,..,ndx_m) = val
```

where each indexing expression `ndx\_i` corresponds to the `i-th` dimension of `a`. In both cases, (vector or multi-dimensional indexing), the right hand side `val` must either be a scalar, an empty matrix, or of the same size as the indices. If `val` is an empty matrix, the assignment acts like a delete. Note that the type of `a` may be modified by the assignment. So, for example, assigning a `double` value to an element of a `float` array `a` will cause the array `a` to become `double`.

For cell arrays, the above forms of assignment will still work, but only if `val` is also a cell array. If you want to assign the contents of a cell in a cell array, you must use one of the two following forms, either

```
a{ndx} = val
```

or

```
a{ndx_1,ndx_2,...,ndx_m} = val
```

which will modify the contents of the cell.

14.3 CELL Cell Array of Empty Matrices

14.3.1 Usage

Creates a cell array of empty matrix entres. Two seperate syntaxes are possible. The first syntax specifies the array dimensions as a sequence of scalar dimensions:

```
y = cell(d1,d2,...,dn).
```

The resulting array has the given dimensions, and is filled with all zeros. The type of `y` is `cell`, a cell array.

The second syntax specifies the array dimensions as a vector, where each element in the vector specifies a dimension length:

```
y = cell([d1,d2,...,dn]).
```

This syntax is more convenient for calling `zeros` using a variable for the argument. In both cases, specifying only one dimension results in a square matrix output.

14.3.2 Example

The following examples demonstrate generation of some zero arrays using the first form.

```
--> cell(2,3,2)

ans =

(:,:,1) =
 [] [] []
 [] [] []

(:,:,2) =
 [] [] []
 [] [] []
```

```
--> cell(1,3)

ans =
 [] [] []
```

The same expressions, using the second form.

```
--> cell([2,6])

ans =
 [] [] [] [] [] []
 [] [] [] [] [] []

--> cell([1,3])

ans =
 [] [] []
```

14.4 CELLFUN Appy a Function To Elements of a Cell Array

14.4.1 Usage

The `cellfun` function is used to apply a function handle (or anonymous function) to each element of a cell array and to collect the outputs into an array. The general syntax for its use is

```
y = cellfun(fun, x)
```

where `x` is an N-dimensional array. In this case, each element of the output `y\_i` is defined as `fun(x{i})`. You can also supply multiple arguments to `cellfun`, provided all of the arguments are the same size

```
y = cellfun(fun, x, z, ...)
```

in which case each output `y\_i` is defined as `fun(x{i},z{i},...)`. Note that unlike `arrayfun`, the `cellfun` function will allow for different types (if there are overloaded versions of the function `fun`) for each element.

If the function returns multiple outputs, then `arrayfun` can be called with multiple outputs, in which case each output goes to a separate array output

```
[y1,y2,...] = cellfun(fun, x, z, ...)
```

The assumption is that the output types for each call to `fun` is the same across the inputs.

Finally, some hints can be provided to `cellfun` using the syntax

```
[y1,y2,...] = cellfun(fun, x, z, ..., 'param', value, 'param', value)
```

where `param` and `value` take on the following possible values:

- `'UniformOutput'` - if the `value` is `true` then each output of `fun` must be a scalar, and the outputs are concatenated into an array the same size as the input arrays. If the `value` is `false` then the outputs are encapsulated into a cell array, with each entry in the cell array containing the call to `fun(x\_i,z\_i,...)`.

- `'ErrorHandler'` - in this case `value` is a function handle that gets called when `fun` throws an error. If `'ErrorHandler'` is not specified, then `arrayfun` allows the error to propogate (i.e., and exception is thrown).

14.5 CIRCSHIFT Circularly Shift an Array

14.5.1 USAGE

Applies a circular shift along each dimension of a given array. The syntax for its use is

```
y = circshift(x,shiftvec)
```

where `x` is an n-dimensional array, and `shiftvec` is a vector of integers, each of which specify how much to shift `x` along the corresponding dimension.

14.5.2 Example

The following examples show some uses of `circshift` on N-dimensional arrays.

```
--> x = int32(rand(4,5)*10)

x =
  4  8  3  2  9
  0  8  0  5  3
  9  1  5  8  2
  4  5 10  3  7

--> circshift(x,[1,0])

ans =
  4  5 10  3  7
  4  8  3  2  9
  0  8  0  5  3
  9  1  5  8  2

--> circshift(x,[0,-1])

ans =
  8  3  2  9  4
  8  0  5  3  0
  1  5  8  2  9
  5 10  3  7  4

--> circshift(x,[2,2])

ans =
  8  2  9  1  5
  3  7  4  5 10
  2  9  4  8  3
  5  3  0  8  0

--> x = int32(rand(4,5,3)*10)

x =

(:,:,1) =
  2  7  7  3 10
  2  2  3  7  0
  4  8  1  4  0
 10  2  7  8  9
```

```
(:,:,2) =
  5  7 10  9  4
  0  3  5  0  4
  4  5  1  3  6
  9  1  5  1  5

(:,:,3) =
  1  5  6  9  2
  8 10  6  5  7
  6  2  1  6  8
  1  9  6  5  3

--> circshift(x,[1,0,0])

ans =

(:,:,1) =
 10  2  7  8  9
  2  7  7  3 10
  2  2  3  7  0
  4  8  1  4  0

(:,:,2) =
  9  1  5  1  5
  5  7 10  9  4
  0  3  5  0  4
  4  5  1  3  6

(:,:,3) =
  1  9  6  5  3
  1  5  6  9  2
  8 10  6  5  7
  6  2  1  6  8

--> circshift(x,[0,-1,0])

ans =

(:,:,1) =
  7  7  3 10  2
  2  3  7  0  2
  8  1  4  0  4
  2  7  8  9 10

(:,:,2) =
  7 10  9  4  5
  3  5  0  4  0
  5  1  3  6  4
  1  5  1  5  9

(:,:,3) =
  5  6  9  2  1
 10  6  5  7  8
```

```
  2  1  6  8  6
  9  6  5  3  1

--> circshift(x,[0,0,-1])

ans =

(:,:,1) =
  5  7 10  9  4
  0  3  5  0  4
  4  5  1  3  6
  9  1  5  1  5

(:,:,2) =
  1  5  6  9  2
  8 10  6  5  7
  6  2  1  6  8
  1  9  6  5  3

(:,:,3) =
  2  7  7  3 10
  2  2  3  7  0
  4  8  1  4  0
 10  2  7  8  9

--> circshift(x,[2,-3,1])

ans =

(:,:,1) =
  6  8  6  2  1
  5  3  1  9  6
  9  2  1  5  6
  5  7  8 10  6

(:,:,2) =
  4  0  4  8  1
  8  9 10  2  7
  3 10  2  7  7
  7  0  2  2  3

(:,:,3) =
  3  6  4  5  1
  1  5  9  1  5
  9  4  5  7 10
  0  4  0  3  5
```

14.6 COND Condition Number of a Matrix

14.6.1 Usage

Calculates the condition number of a matrix. To compute the 2-norm condition number of a matrix (ratio of largest to smallest singular values), use the syntax

```
y = cond(A)
```

where A is a matrix. If you want to compute the condition number in a different norm (e.g., the 1-norm), use the second syntax

```
y = cond(A,p)
```

where `p` is the norm to use when computing the condition number. The following choices of `p` are supported

- `p = 1` returns the 1-norm, or the max column sum of A
- `p = 2` returns the 2-norm (largest singular value of A)
- `p = inf` returns the infinity norm, or the max row sum of A
- `p = 'fro'` returns the Frobenius-norm (vector Euclidean norm, or RMS value)

14.6.2 Function Internals

The condition number is defined as

$$\frac{\|A\|_p}{\|A^{-1}\|_p}$$

This equation is precisely how the condition number is computed for the case `p ~= 2`. For the `p=2` case, the condition number can be computed much more efficiently using the ratio of the largest and smallest singular values.

14.6.3 Example

The condition number of this matrix is large

```
--> A = [1,1;0,1e-15]

A =
    1.0000    1.0000
         0    0.0000

--> cond(A)

ans =
 2.0000e+15

--> cond(A,1)

ans =
 2000000000000002
```

You can also (for the case `p=1` use `rcond` to calculate an estimate of the condition number

```
--> 1/rcond(A)

ans =
 2.0000e+15
```

14.7 DET Determinant of a Matrix

14.7.1 Usage

Calculates the determinant of a matrix. Note that for all but very small problems, the determinant is not particularly useful. The condition number `cond` gives a more reasonable estimate as to the suitability of a matrix for inversion than comparing `det(A)` to zero. In any case, the syntax for its use is

```
y = det(A)
```

where A is a square matrix.

14.7.2 Function Internals

The determinant is calculated via the `LU` decomposition. Note that the determinant of a product of matrices is the product of the determinants. Then, we have that

$$LU = PA$$

where `L` is lower triangular with 1s on the main diagonal, `U` is upper triangular, and `P` is a row-permutation matrix. Taking the determinant of both sides yields

$$|LU| = |L||U| = |U| = |PA| = |P||A|$$

where we have used the fact that the determinant of `L` is 1. The determinant of `P` (which is a row exchange matrix) is either 1 or -1.

14.7.3 Example

Here we assemble a random matrix and compute its determinant

```
--> A = rand(5);
--> det(A)

ans =
   -0.0489
```

Then, we exchange two rows of `A` to demonstrate how the determinant changes sign (but the magnitude is the same)

```
--> B = A([2,1,3,4,5],:);
--> det(B)

ans =
    0.0489
```

14.8 DIAG Diagonal Matrix Construction/Extraction

14.8.1 Usage

The `diag` function is used to either construct a diagonal matrix from a vector, or return the diagonal elements of a matrix as a vector. The general syntax for its use is

```
y = diag(x,n)
```

If `x` is a matrix, then `y` returns the `n`-th diagonal. If `n` is omitted, it is assumed to be zero. Conversely, if `x` is a vector, then `y` is a matrix with `x` set to the `n`-th diagonal.

14.8.2 Examples

Here is an example of `diag` being used to extract a diagonal from a matrix.

```
--> A = int32(10*rand(4,5))

A =
  5  8  8  3  6
  4  8  4  3  7
  9  5  8  4  2
  1  0 10  0  4

--> diag(A)

ans =
 5
 8
 8
 0

--> diag(A,1)

ans =
 8
 4
 4
 4
```

Here is an example of the second form of `diag`, being used to construct a diagonal matrix.

```
--> x = int32(10*rand(1,3))

x =
 6 3 9

--> diag(x)

ans =
 6 0 0
 0 3 0
 0 0 9

--> diag(x,-1)

ans =
 0 0 0 0
 6 0 0 0
 0 3 0 0
 0 0 9 0
```

14.9 EXPM Matrix Exponential

14.9.1 Usage

Calculates `e\^A` for a square, full rank matrix `A`. The syntax for its use is

```
y = expm(A)
```

Internally, `expm` is mapped to a simple `e\^A` expression (which in turn uses the eigenvalue expansion of `A` to compute the exponential).

14.9.2 Example

An example of `expm`

```
--> A = [1 1 0; 0 0 2; 0 0 -1]

A =
  1  1  0
  0  0  2
  0  0 -1

--> expm(A)

ans =
    2.7183    1.7183    1.0862
         0    1.0000    1.2642
         0         0    0.3679
```

14.10 EYE Identity Matrix

14.10.1 USAGE

Creates an identity matrix of the specified size. The syntax for its use is

```
y = eye(n)
```

where `n` is the size of the identity matrix. The type of the output matrix is `float`.

14.10.2 Example

The following example demonstrates the identity matrix.

```
--> eye(3)

ans =
 1 0 0
 0 1 0
 0 0 1
```

14.11 FIND Find Non-zero Elements of An Array

14.11.1 Usage

Returns a vector that contains the indicies of all non-zero elements in an array. The usage is

```
y = find(x)
```

The indices returned are generalized column indices, meaning that if the array `x` is of size `[d1,d2,...,dn]`, and the element `x(i1,i2,...,in)` is nonzero, then `y` will contain the integer

$$i_1 + (i_2 - 1)d_1 + (i_3 - 1)d_1 d_2 + \ldots$$

The second syntax for the `find` command is

```
[r,c] = find(x)
```

which returns the row and column index of the nonzero entries of `x`. The third syntax for the `find` command also returns the values

```
[r,c,v] = find(x).
```

Note that if the argument is a row vector, then the returned vectors are also row vectors. This form is particularly useful for converting sparse matrices into IJV form.

The `find` command also supports some additional arguments. Each of the above forms can be combined with an integer indicating how many results to return:

```
y = find(x,k)
```

where `k` is the maximum number of results to return. This form will return the first `k` results. You can also specify an optional flag indicating whether to take the first or last `k` values:

```
y = find(x,k,'first')
y = find(x,k,'last')
```

in the case of the `'last'` argument, the last `k` values are returned.

14.11.2 Example

Some simple examples of its usage, and some common uses of `find` in FreeMat programs.

```
--> a = [1,2,5,2,4];
--> find(a==2)

ans =
 2 4
```

Here is an example of using find to replace elements of `A` that are `0` with the number `5`.

```
--> A = [1,0,3;0,2,1;3,0,0]

A =
 1 0 3
 0 2 1
 3 0 0

--> n = find(A==0)

n =
 2
 4
 6
 9

--> A(n) = 5

A =
 1 5 3
 5 2 1
 3 5 5
```

Incidentally, a better way to achieve the same concept is:

```
--> A = [1,0,3;0,2,1;3,0,0]

A =
 1 0 3
 0 2 1
 3 0 0

--> A(A==0) = 5

A =
 1 5 3
 5 2 1
 3 5 5
```

Now, we can also return the indices as row and column indices using the two argument form of `find`:

```
--> A = [1,0,3;0,2,1;3,0,0]

A =
 1 0 3
 0 2 1
 3 0 0

--> [r,c] = find(A)
r =
 1
 3
 2
 1
 2

c =
 1
 1
 2
 3
 3
```

Or the three argument form of `find`, which returns the value also:

```
--> [r,c,v] = find(A)
r =
 1
 3
 2
 1
 2

c =
 1
 1
 2
 3
 3

v =
```

```
 1
 3
 2
 3
 1
```

14.12 FLIPDIM Reverse a Matrix Along a Given Dimension

14.12.1 USAGE

Reverses an array along the given dimension. The syntax for its use is

```
   y = flipdim(x,n)
```

where `x` is matrix, and `n` is the dimension to reverse.

14.12.2 Example

The following examples show some uses of `flipdim` on N-dimensional arrays.

```
--> x = int32(rand(4,5,3)*10)

x =

(:,:,1) =
  5  2  4  2  8
  7  6  6  6  7
  7  0  1  1  0
  3  2  1  9  9

(:,:,2) =
 10  6  3  3  1
  1  2  5  7 10
  9  7  5  1  4
  3 10  4  4  3

(:,:,3) =
  3  6  5  8  9
  9  8  5  3  0
  1  7  9  4  8
  4  6  4  9  5

--> flipdim(x,1)

ans =

(:,:,1) =
  3  2  1  9  9
  7  0  1  1  0
  7  6  6  6  7
  5  2  4  2  8

(:,:,2) =
  3 10  4  4  3
  9  7  5  1  4
```

```
  1  2  5  7 10
 10  6  3  3  1

(:,:,3) =
  4  6  4  9  5
  1  7  9  4  8
  9  8  5  3  0
  3  6  5  8  9

--> flipdim(x,2)

ans =

(:,:,1) =
  8  2  4  2  5
  7  6  6  6  7
  0  1  1  0  7
  9  9  1  2  3

(:,:,2) =
  1  3  3  6 10
 10  7  5  2  1
  4  1  5  7  9
  3  4  4 10  3

(:,:,3) =
  9  8  5  6  3
  0  3  5  8  9
  8  4  9  7  1
  5  9  4  6  4

--> flipdim(x,3)

ans =

(:,:,1) =
  3  6  5  8  9
  9  8  5  3  0
  1  7  9  4  8
  4  6  4  9  5

(:,:,2) =
 10  6  3  3  1
  1  2  5  7 10
  9  7  5  1  4
  3 10  4  4  3

(:,:,3) =
  5  2  4  2  8
  7  6  6  6  7
  7  0  1  1  0
  3  2  1  9  9
```

14.13 FLIPLR Reverse the Columns of a Matrix

14.13.1 USAGE

Reverses the columns of a matrix. The syntax for its use is

```
y = fliplr(x)
```

where `x` is matrix. If `x` is an N-dimensional array then the second dimension is reversed.

14.13.2 Example

The following example shows `fliplr` applied to a 2D matrix.

```
--> x = int32(rand(4)*10)

x =
  6  4  7  4
  8  5  4  1
  5  8  7  9
  1 10  9  9

--> fliplr(x)

ans =
  4  7  4  6
  1  4  5  8
  9  7  8  5
  9  9 10  1
```

For a 3D array, note how the columns in each slice are flipped.

```
--> x = int32(rand(4,4,3)*10)

x =

(:,:,1) =
  4  8  1  8
  1  5  5  2
  2 10  5  8
  4  8  2  1

(:,:,2) =
  0  3  4  1
  6  6 10  8
  4  3  3  6
  2  9  7  3

(:,:,3) =
  6  5  1  1
  6  8 10  3
  4  3  7  9
  9  4  4  3

--> fliplr(x)

ans =
```

```
(:,:,1) =
  8  1  8  4
  2  5  5  1
  8  5 10  2
  1  2  8  4

(:,:,2) =
  1  4  3  0
  8 10  6  6
  6  3  3  4
  3  7  9  2

(:,:,3) =
  1  1  5  6
  3 10  8  6
  9  7  3  4
  3  4  4  9
```

14.14 FLIPUD Reverse the Columns of a Matrix

14.14.1 USAGE

Reverses the rows of a matrix. The syntax for its use is

```
y = flipud(x)
```

where `x` is matrix. If `x` is an N-dimensional array then the first dimension is reversed.

14.14.2 Example

The following example shows `flipud` applied to a 2D matrix.

```
--> x = int32(rand(4)*10)

x =
 9 4 5 3
 8 9 7 4
 4 8 6 3
 6 7 0 9

--> flipud(x)

ans =
 6 7 0 9
 4 8 6 3
 8 9 7 4
 9 4 5 3
```

For a 3D array, note how the rows in each slice are flipped.

```
--> x = int32(rand(4,4,3)*10)

x =

(:,:,1) =
```

```
  7  4  3  3
  0 10 10  6
  2  8  1  8
  1  1  9  8

(:,:,2) =
  4  4  3  2
  4  4  7  1
  4  9  9  8
  5  5  1  6

(:,:,3) =
  9  7  7  5
  8  9  5  6
  5 10  6  8
  4  2  8  3

--> flipud(x)

ans =

(:,:,1) =
  1  1  9  8
  2  8  1  8
  0 10 10  6
  7  4  3  3

(:,:,2) =
  5  5  1  6
  4  9  9  8
  4  4  7  1
  4  4  3  2

(:,:,3) =
  4  2  8  3
  5 10  6  8
  8  9  5  6
  9  7  7  5
```

14.15 IPERMUTE Array Inverse Permutation Function

14.15.1 Usage

The `ipermute` function rearranges the contents of an array according to the inverse of the specified permutation vector. The syntax for its use is

```
y = ipermute(x,p)
```

where `p` is a permutation vector - i.e., a vector containing the integers `1...ndims(x)` each occuring exactly once. The resulting array `y` contains the same data as the array `x`, but ordered according to the inverse of the given permutation. This function and the `permute` function are inverses of each other.

14.15.2 Example

First we create a large multi-dimensional array, then permute it and then inverse permute it, to retrieve the original array:

```
--> A = randn(13,5,7,2);
--> size(A)

ans =
 13  5  7  2

--> B = permute(A,[3,4,2,1]);
--> size(B)

ans =
  7  2  5 13

--> C = ipermute(B,[3,4,2,1]);
--> size(C)

ans =
 13  5  7  2

--> any(A~=C)

ans =

(:,:,1,1) =
 0 0 0 0 0

(:,:,2,1) =
 0 0 0 0 0

(:,:,3,1) =
 0 0 0 0 0

(:,:,4,1) =
 0 0 0 0 0

(:,:,5,1) =
 0 0 0 0 0

(:,:,6,1) =
 0 0 0 0 0

(:,:,7,1) =
 0 0 0 0 0

(:,:,1,2) =
 0 0 0 0 0

(:,:,2,2) =
 0 0 0 0 0

(:,:,3,2) =
```

```
 0 0 0 0 0

(:,:,4,2) =
 0 0 0 0 0

(:,:,5,2) =
 0 0 0 0 0

(:,:,6,2) =
 0 0 0 0 0

(:,:,7,2) =
 0 0 0 0 0
```

14.16 ISFLOAT Test for Floating Point Array

14.16.1 Usage

The syntax for `isfloat` is

```
x = isfloat(y)
```

and it returns a logical 1 if the argument is a floating point array (i.e., a `single` or `double`), and a logical 0 otherwise.

14.17 ISINTEGER Test for Integer Array

14.17.1 Usage

The syntax for `isinteger` is

```
x = isinteger(y)
```

and it returns a logical 1 if the argument is an integer. The decision of whether the argument is an integer or not is made based on the class of `y`, not on its value.

14.18 LINSPACE Linearly Spaced Vector

14.18.1 Usage

Generates a row vector with the specified number of elements, with entries uniformly spaced between two specified endpoints. The syntax for its use is either

```
y = linspace(a,b,count)
```

or, for a default `count = 100`,

```
y = linspace(a,b);
```

14.18.2 Examples

Here is a simple example of using `linspace`

```
--> x = linspace(0,1,5)

x =
         0    0.2500    0.5000    0.7500    1.0000
```

14.19 LOGSPACE Logarithmically Spaced Vector

14.19.1 Usage

Generates a row vector with the specified number number of elements, with entries logarithmically spaced between two specified endpoints. The syntax for its use is either

```
y = logspace(a,b,count)
```

or, for a default `count = 50`,

```
y = logspace(a,b)
```

A third special use is when

```
y = logspace(a,pi)
```

where it generates points between `10\^a` and `pi`

Contributed by Paulo Xavier Candeias under GPL.

14.19.2 Example

Here is an example of the use of `logspace`

```
--> logspace(1,2,3)

ans =
   10.0000   31.6228  100.0000
```

14.20 MESHGRID Generate Grid Mesh For Plots

14.20.1 Usage

The `meshgrid` function generates arrays that can be used for the generation of surface plots. The syntax is one of

```
[X,Y] = meshgrid(x)
[X,Y] = meshgrid(x,y)
[X,Y,Z] = meshgrid(x,y,z)
```

where `x,y,z` are vectors, and `X,Y,Z` are matrices. In the first case `[X,Y] = meshgrid(x)`, the rows of `X` and the columns of `Y` contain copies of the vector `x`. In the second case `[X,Y] = meshgrid(x,y)`, the rows of `X` contain copies of `x`, and the columns of `Y` contain copies of `y`. In the third case, each input is copied along the row, column or slice direction of the corresponding output variable.

14.20.2 Example

In the first example:

```
--> [X,Y] = meshgrid(-2:.4:2)
X =
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
```

```
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000
   -2.0000   -1.6000   -1.2000   -0.8000   -0.4000    0.0000    0.4000    0.8000    1.2000    1.6000

Y =
   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000
   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000
   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000
   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000
   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000
    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000
    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000
    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000
    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000
```

Next, we use different vectors for `X` and for `Y`:

```
--> [X,Y] = meshgrid([1,2,3,4],[6,7,8])
X =
 1 2 3 4
 1 2 3 4
 1 2 3 4

Y =
 6 6 6 6
 7 7 7 7
 8 8 8 8
```

14.21 NAN Not-a-Number Constant

14.21.1 Usage

Returns a value that represents "not-a-number" for both 32 and 64-bit floating point values. This constant is meant to represent the result of arithmetic operations whose output cannot be meaningfully defined (like zero divided by zero). There are several forms for the `NaN` function. The first form returns a double precision `NaN`.

```
y = nan
```

The next form takes a class name that can be either `'double'`

```
y = nan('double')
```

or `'single'`:

```
y = nan('single')
```

With a single parameter it generates a square matrix of `nan`s.

```
y = nan(n)
```

Alternatively, you can specify the dimensions of the array via

```
y = nan(m,n,p,...)
```

or

```
y = nan([m,n,p,...])
```

Finally, you can add a classname of either `'single'` or `'double'`.

14.21.2 Example

The following examples demonstrate a few calculations with the not-a-number constant.

```
--> nan*0

ans =
 NaN

--> nan-nan

ans =
 NaN
```

Note that `NaN`s are preserved under type conversion to floating point types (i.e., `float`, `double`, `complex` and `dcomplex` types), but not integer types.

```
--> uint32(nan)

ans =
 0

--> complex(nan)

ans =
 NaN
```

14.22 NDGRID Generate N-Dimensional Grid

14.22.1 Usage

Generates N-dimensional grids, each of which is constant in all but one dimension. The syntax for its use is either

```
[y1, y2, ..., ym] = ndgrid(x1, x2, ..., xn)
```

where `m <= n` or

```
[y1, y2, ..., ym] = ndgrid(x1)
```

which is equivalent to the first form, with `x1=x2=...=xn`. Each output `yi` is an `n`-dimensional array, with values such that

$$y_i(d_1, \ldots, d_{i-1}, d_i, d_{i+1}, \ldots, d_m) = x_i(d_i)$$

`ndgrid` is useful for evaluating multivariate functionals over a range of arguments. It is a generalization of `meshgrid`, except that `meshgrid` transposes the dimensions corresponding to the first two arguments to better fit graphical applications.

14.22.2 Example

Here is a simple `ndgrid` example

```
--> [a,b] = ndgrid(1:2,3:5)
a =
 1 1 1
 2 2 2

b =
```

```
 3 4 5
 3 4 5

--> [a,b,c] = ndgrid(1:2,3:5,0:1)
a =

(:,:,1) =
 1 1 1
 2 2 2

(:,:,2) =
 1 1 1
 2 2 2

b =

(:,:,1) =
 3 4 5
 3 4 5

(:,:,2) =
 3 4 5
 3 4 5

c =

(:,:,1) =
 0 0 0
 0 0 0

(:,:,2) =
 1 1 1
 1 1 1
```

Here we use the second form

```
--> [a,b,c] = ndgrid(1:3)
a =

(:,:,1) =
 1 1 1
 2 2 2
 3 3 3

(:,:,2) =
 1 1 1
 2 2 2
 3 3 3

(:,:,3) =
 1 1 1
 2 2 2
 3 3 3

b =
```

```
(:,:,1) =
 1 2 3
 1 2 3
 1 2 3

(:,:,2) =
 1 2 3
 1 2 3
 1 2 3

(:,:,3) =
 1 2 3
 1 2 3
 1 2 3

c =

(:,:,1) =
 1 1 1
 1 1 1
 1 1 1

(:,:,2) =
 2 2 2
 2 2 2
 2 2 2

(:,:,3) =
 3 3 3
 3 3 3
 3 3 3
```

14.23 NONZEROS Retrieve Nonzero Matrix Entries

14.23.1 USAGE

Returns a dense column vector containing the nonzero elements of the argument matrix. The syntax for its use is

```
y = nonzeros(x)
```

where `x` is the argument array. The argument matrix may be sparse as well as dense.

14.23.2 Example

Here is an example of using `nonzeros` on a sparse matrix.

```
--> a = rand(8); a(a>0.2) = 0;
--> A = sparse(a)

A =
 1 1 0.0596135
 7 1 0.0283717
 8 1 0.0337801
```

```
 5 2 0.0700267
 1 4 0.0881058
 4 4 0.00699947
 5 4 0.0494723
 8 5 0.0420057
 4 6 0.153486
 7 6 0.0654851
 1 7 0.174397
 4 7 0.0684673
 2 8 0.13853
--> nonzeros(A)

ans =
    0.0596
    0.0284
    0.0338
    0.0700
    0.0881
    0.0070
    0.0495
    0.0420
    0.1535
    0.0655
    0.1744
    0.0685
    0.1385
```

14.24 NORM Norm Calculation

14.24.1 Usage

Calculates the norm of a matrix. There are two ways to use the `norm` function. The general syntax is

```
y = norm(A,p)
```

where `A` is the matrix to analyze, and `p` is the type norm to compute. The following choices of `p` are supported

- `p = 1` returns the 1-norm, or the max column sum of A
- `p = 2` returns the 2-norm (largest singular value of A)
- `p = inf` returns the infinity norm, or the max row sum of A
- `p = 'fro'` returns the Frobenius-norm (vector Euclidean norm, or RMS value)

For a vector, the regular norm calculations are performed:

- `1 <= p < inf` returns `sum(abs(A).\^p)\^(1/p)`
- `p` unspecified returns `norm(A,2)`
- `p = inf` returns max(abs(A))
- `p = -inf` returns min(abs(A))

14.24.2 Examples

Here are the various norms calculated for a sample matrix

```
--> A = float(rand(3,4))

A =
    0.8462    0.9465    0.6874    0.8668
    0.1218    0.9206    0.5877    0.5837
    0.7081    0.6608    0.2035    0.5083

--> norm(A,1)

ans =
    2.5280

--> norm(A,2)

ans =
    2.2997

--> norm(A,inf)

ans =
    3.3470

--> norm(A,'fro')

ans =
    2.3712
```

Next, we calculate some vector norms.

```
--> A = float(rand(4,1))

A =
    0.3458
    0.1427
    0.3998
    0.7194

--> norm(A,1)

ans =
    1.6078

--> norm(A,2)

ans =
    0.9041

--> norm(A,7)

ans =
    0.7217
```

```
--> norm(A,inf)

ans =
    0.7194

--> norm(A,-inf)

ans =
    0.1427
```

14.25 NUM2STR Convert Numbers To Strings

14.25.1 Usage

Converts an array into its string representation. The general syntax for this function is

```
s = num2str(X)
```

where `s` is a string (or string matrix) and `X` is an array. By default, the `num2str` function uses 4 digits of precision and an exponent if required. If you want more digits of precision, you can specify the precition via the form

```
s = num2str(X, precision)
```

where `precision` is the number of digits to include in the string representation. For more control over the format of the output, you can also specify a format specifier (see `printf` for more details).

```
s = num2str(X, format)
```

where `format` is the specifier string.

14.26 ONES Array of Ones

14.26.1 Usage

Creates an array of ones of the specified size. Two seperate syntaxes are possible. The first syntax specifies the array dimensions as a sequence of scalar dimensions:

```
y = ones(d1,d2,...,dn).
```

The resulting array has the given dimensions, and is filled with all ones. The type of `y` is `float`, a 32-bit floating point array. To get arrays of other types, use the typecast functions (e.g., `uint8`, `int8`, etc.).

The second syntax specifies the array dimensions as a vector, where each element in the vector specifies a dimension length:

```
y = ones([d1,d2,...,dn]).
```

This syntax is more convenient for calling `ones` using a variable for the argument. In both cases, specifying only one dimension results in a square matrix output.

14.26.2 Example

The following examples demonstrate generation of some arrays of ones using the first form.

```
--> ones(2,3,2)

ans =
```

```
(:,:,1) =
 1 1 1
 1 1 1

(:,:,2) =
 1 1 1
 1 1 1

--> ones(1,3)

ans =
 1 1 1
```

The same expressions, using the second form.

```
--> ones([2,6])

ans =
 1 1 1 1 1 1
 1 1 1 1 1 1

--> ones([1,3])

ans =
 1 1 1
```

Finally, an example of using the type casting function `uint16` to generate an array of 16-bit unsigned integers with a value of 1.

```
--> uint16(ones(3))

ans =
 1 1 1
 1 1 1
 1 1 1
```

14.27 PERMUTE Array Permutation Function

14.27.1 Usage

The `permute` function rearranges the contents of an array according to the specified permutation vector. The syntax for its use is

```
y = permute(x,p)
```

where `p` is a permutation vector - i.e., a vector containing the integers `1...ndims(x)` each occuring exactly once. The resulting array `y` contains the same data as the array `x`, but ordered according to the permutation. This function is a generalization of the matrix transpose operation.

14.27.2 Example

Here we use `permute` to transpose a simple matrix (note that permute also works for sparse matrices):

```
--> A = [1,2;4,5]

A =
```

```
 1 2
 4 5

--> permute(A,[2,1])

ans =
 1 4
 2 5

--> A'

ans =
 1 4
 2 5
```

Now we permute a larger n-dimensional array:

```
--> A = randn(13,5,7,2);
--> size(A)

ans =
 13  5  7  2

--> B = permute(A,[3,4,2,1]);
--> size(B)

ans =
  7  2  5 13
```

14.28 PINV Moore-Penrose Pseudoinverse

14.28.1 Usage

Calculates the Moore-Penrose pseudoinverse of a matrix. The general syntax for its use is

```
y = pinv(A,tol)
```

or for a default specification of the tolerance `tol`,

```
y = pinv(A)
```

For any `m x n` matrix `A`, the Moore-Penrose pseudoinverse is the unique `n x m` matrix `B` that satisfies the following four conditions

- `A B A = A`
- `B A B = B`
- `(A B)' = A B`
- `(B A)' = B A`

Also, it is true that `B y` is the minimum norm, least squares solution to `A x = y`. The Moore-Penrose pseudoinverse is computed from the singular value decomposition of `A`, with singular values smaller than `tol` being treated as zeros. If `tol` is not specified then it is chosen as

```
tol = max(size(A)) * norm(A) * teps(A).
```

14.28.2 Function Internals

The calculation of the MP pseudo-inverse is almost trivial once the svd of the matrix is available. First, for a real, diagonal matrix with positive entries, the pseudo-inverse is simply

$$\left(\Sigma^{+}\right)_{ii} = \begin{cases} 1/\sigma_{ii} & \sigma_{ii} > 0 \\ 0 & \text{else} \end{cases}$$

One can quickly verify that this choice of matrix satisfies the four properties of the pseudoinverse. Then, the pseudoinverse of a general matrix `A = U S V'` is defined as

$$A^{+} = VS^{+}U'$$

and again, using the facts that `U' U = I` and `V V' = I`, one can quickly verify that this choice of pseudoinverse satisfies the four defining properties of the MP pseudoinverse. Note that in practice, the diagonal pseudoinverse `S\^{+}` is computed with a threshold (the `tol` argument to `pinv`) so that singular values smaller than `tol` are treated like zeros.

14.28.3 Examples

Consider a simple `1 x 2` matrix example, and note the various Moore-Penrose conditions:

```
--> A = float(rand(1,2))

A =
    0.4840    0.0187

--> B = pinv(A)

B =
    2.0630
    0.0796

--> A*B*A

ans =
    0.4840    0.0187

--> B*A*B

ans =
    2.0630
    0.0796

--> A*B

ans =
    1.0000

--> B*A

ans =
    0.9985    0.0385
    0.0385    0.0015
```

To demonstrate that `pinv` returns the least squares solution, consider the following very simple case

```
--> A = float([1;1;1;1])

A =
 1
 1
 1
 1
```

The least squares solution to `A x = b` is just `x = mean(b)`, and computing the `pinv` of `A` demonstrates this

```
--> pinv(A)

ans =
    0.2500    0.2500    0.2500    0.2500
```

Similarly, we can demonstrate the minimum norm solution with the following simple case

```
--> A = float([1,1])

A =
 1 1
```

The solutions of `A x = 5` are those `x\_1` and `x\_2` such that `x\_1 + x\_2 = 5`. The norm of `x` is `x\_1\^ + x\_2\^2`, which is `x\_1\^2 + (5-x\_1)\^2`, which is minimized for `x\_1 = x\_2 = 2.5`:

```
--> pinv(A) * 5.0

ans =
    2.5000
    2.5000
```

14.29 RANK Calculate the Rank of a Matrix

14.29.1 Usage

Returns the rank of a matrix. There are two ways to use the `rank` function is

```
y = rank(A,tol)
```

where `tol` is the tolerance to use when computing the rank. The second form is

```
y = rank(A)
```

in which case the tolerance `tol` is chosen as

```
tol = max(size(A))*max(s)*eps,
```

where `s` is the vector of singular values of `A`. The rank is computed using the singular value decomposition `svd`.

14.29.2 Examples

Some examples of matrix rank calculations

```
--> rank([1,3,2;4,5,6])

ans =
 2
```

```
--> rank([1,2,3;2,4,6])

ans =
 1
```

Here we construct an ill-conditioned matrix, and show the use of the `tol` argument.

```
--> A = [1,0;0,eps/2]

A =
    1.0000         0
         0    0.0000

--> rank(A)

ans =
 1

--> rank(A,eps/8)

ans =
 2
```

14.30 RCOND Reciprocal Condition Number Estimate

14.30.1 Usage

The `rcond` function is a FreeMat wrapper around LAPACKs function `XGECON`, which estimates the 1-norm condition number (reciprocal). For the details of the algorithm see the LAPACK documentation. The syntax for its use is

```
   x = rcond(A)
```

where `A` is a matrix.

14.30.2 Example

Here is the reciprocal condition number for a random square matrix

```
--> A = rand(30);
--> rcond(A)

ans =
 4.4279e-04
```

And here we calculate the same value using the definition of (reciprocal) condition number

```
--> 1/(norm(A,1)*norm(inv(A),1))

ans =
 4.3595e-04
```

Note that the values are very similar. LAPACKs `rcond` function is far more efficient than the explicit calculation (which is also used by the `cond` function.

14.31 REPMAT Array Replication Function

14.31.1 Usage

The `repmat` function replicates an array the specified number of times. The source and destination arrays may be multidimensional. There are three distinct syntaxes for the `repmap` function. The first form:

```
y = repmat(x,n)
```

replicates the array `x` on an `n-times-n` tiling, to create a matrix `y` that has `n` times as many rows and columns as `x`. The output `y` will match `x` in all remaining dimensions. The second form is

```
y = repmat(x,m,n)
```

And creates a tiling of `x` with `m` copies of `x` in the row direction, and `n` copies of `x` in the column direction. The final form is the most general

```
y = repmat(x,[m n p...])
```

where the supplied vector indicates the replication factor in each dimension.

14.31.2 Example

Here is an example of using the `repmat` function to replicate a row 5 times. Note that the same effect can be accomplished (although somewhat less efficiently) by a multiplication.

```
--> x = [1 2 3 4]

x =
 1 2 3 4

--> y = repmat(x,[5,1])

y =
 1 2 3 4
 1 2 3 4
 1 2 3 4
 1 2 3 4
 1 2 3 4
```

The `repmat` function can also be used to create a matrix of scalars or to provide replication in arbitrary dimensions. Here we use it to replicate a 2D matrix into a 3D volume.

```
--> x = [1 2;3 4]

x =
 1 2
 3 4

--> y = repmat(x,[1,1,3])

y =

(:,:,1) =
 1 2
 3 4

(:,:,2) =
```

```
 1 2
 3 4

(:,:,3) =
 1 2
 3 4
```

14.32 RESHAPE Reshape An Array

14.32.1 Usage

Reshapes an array from one size to another. Two seperate syntaxes are possible. The first syntax specifies the array dimensions as a sequence of scalar dimensions:

```
y = reshape(x,d1,d2,...,dn).
```

The resulting array has the given dimensions, and is filled with the contents of `x`. The type of `y` is the same as `x`. The second syntax specifies the array dimensions as a vector, where each element in the vector specifies a dimension length:

```
y = reshape(x,[d1,d2,...,dn]).
```

This syntax is more convenient for calling `reshape` using a variable for the argument. The `reshape` function requires that the length of `x` equal the product of the `di` values. Note that arrays are stored in column format, which means that elements in `x` are transferred to the new array `y` starting with the first column first element, then proceeding to the last element of the first column, then the first element of the second column, etc.

14.32.2 Example

Here are several examples of the use of `reshape` applied to various arrays. The first example reshapes a row vector into a matrix.

```
--> a = uint8(1:6)

a =
 1 2 3 4 5 6

--> reshape(a,2,3)

ans =
 1 3 5
 2 4 6
```

The second example reshapes a longer row vector into a volume with two planes.

```
--> a = uint8(1:12)

a =
  1  2  3  4  5  6  7  8  9 10 11 12

--> reshape(a,[2,3,2])

ans =

(:,:,1) =
```

```
  1  3  5
  2  4  6

(:,:,2) =
  7  9 11
  8 10 12
```

The third example reshapes a matrix into another matrix.

```
--> a = [1,6,7;3,4,2]

a =
 1 6 7
 3 4 2

--> reshape(a,3,2)

ans =
 1 4
 3 7
 6 2
```

14.33 RREF Reduced Row Echelon Form of a Matrix

14.33.1 Usage

Calculates the reduced row echelon form of a matrix using Gauss Jordan elimination with partial pivoting. The generic syntax for `rref` is

```
R = rref(A)
```

A default tolerance of `max(size(A))*eps*norm(A,inf)` is used to detect negligible column elements. The second form of `rref` returns a vector `k` as well as `R`

```
[R,k] = rref(A)
```

where `k` is a vector that correponds to the columns of `A` used as pivot columns. If you want to control the tolerance used to identify negligible elements, you can use the form

```
[R,k] = rref(A, tolerance)
```

This implementation of `rref` is based on the one from the matcompat lib for octave. It is copyright Paul Kienzle, and distributed under the GNU GPL.

14.34 SHIFTDIM Shift Array Dimensions Function

14.34.1 Usage

The `shiftdim` function is used to shift the dimensions of an array. The general syntax for the `shiftdim` function is

```
y = shiftdim(x,n)
```

where `x` is a multidimensional array, and `n` is an integer. If `n` is a positive integer, then `shiftdim` circularly shifts the dimensions of `x` to the left, wrapping the dimensions around as necessary. If `n` is a negative integer, then `shiftdim` shifts the dimensions of `x` to the right, introducing singleton dimensions as necessary. In its second form:

```
  [y,n] = shiftdim(x)
```

the `shiftdim` function will shift away (to the left) the leading singleton dimensions of `x` until the leading dimension is not a singleton dimension (recall that a singleton dimension `p` is one for which `size(x,p) == 1`).

14.34.2 Example

Here are some simple examples of using `shiftdim` to remove the singleton dimensions of an array, and then restore them:

```
--> x = uint8(10*randn(1,1,1,3,2));
--> [y,n] = shiftdim(x);
--> n

ans =
 3

--> size(y)

ans =
 3 2

--> c = shiftdim(y,-n);
--> size(c)

ans =
 1 1 1 3 2

--> any(c~=x)

ans =

(:,:,1,1,1) =
 0

(:,:,1,1,2) =
 0
```

Note that these operations (where shifting involves only singleton dimensions) do not actually cause data to be resorted, only the size of the arrays change. This is not true for the following example, which triggers a call to `permute`:

```
--> z = shiftdim(x,4);
```

Note that `z` is now the transpose of `x`

```
--> squeeze(x)

ans =
 11  1
  0  0
  0  8

--> squeeze(z)

ans =
 11  0  0
  1  0  8
```

14.35 SORT Sort

14.35.1 Usage

Sorts an n-dimensional array along the specified dimensional. The first form sorts the array along the first non-singular dimension.

```
B = sort(A)
```

Alternately, the dimension along which to sort can be explicitly specified

```
B = sort(A,dim)
```

FreeMat does not support vector arguments for `dim` - if you need `A` to be sorted along multiple dimensions (i.e., row first, then columns), make multiple calls to `sort`. Also, the direction of the sort can be specified using the `mode` argument

```
B = sort(A,dim,mode)
```

where `mode = 'ascend'` means to sort the data in ascending order (the default), and `mode = 'descend'` means to sort the data into descending order.

When two outputs are requested from `sort`, the indexes are also returned. Thus, for

```
[B,IX] = sort(A)
[B,IX] = sort(A,dim)
[B,IX] = sort(A,dim,mode)
```

an array `IX` of the same size as `A`, where `IX` records the indices of `A` (along the sorting dimension) corresponding to the output array `B`.

Two additional issues worth noting. First, a cell array can be sorted if each cell contains a `string`, in which case the strings are sorted by lexical order. The second issue is that FreeMat uses the same method as MATLAB to sort complex numbers. In particular, a complex number `a` is less than another complex number `b` if `abs(a) < abs(b)`. If the magnitudes are the same then we test the angle of `a`, i.e. `angle(a) < angle(b)`, where `angle(a)` is the phase of `a` between `-pi,pi`.

14.35.2 Example

Here are some examples of sorting on numerical arrays.

```
--> A = int32(10*rand(4,3))

A =
 8 2 8
 0 5 5
 2 5 7
 3 7 1

--> [B,IX] = sort(A)
B =
 0 2 1
 2 5 5
 3 5 7
 8 7 8

IX =
 2 1 4
 3 2 2
 4 3 3
```

```
 1 4 1

--> [B,IX] = sort(A,2)
B =
 2 8 8
 0 5 5
 2 5 7
 1 3 7

IX =
 2 1 3
 1 2 3
 1 2 3
 3 1 2

--> [B,IX] = sort(A,1,'descend')
B =
 8 7 8
 3 5 7
 2 5 5
 0 2 1

IX =
 1 4 1
 4 2 3
 3 3 2
 2 1 4
```

Here we sort a cell array of strings.

```
--> a = {'hello','abba','goodbye','jockey','cake'}

a =
 [hello] [abba] [goodbye] [jockey] [cake]

--> b = sort(a)

b =
 [abba] [cake] [goodbye] [hello] [jockey]
```

14.36 SQUEEZE Remove Singleton Dimensions of an Array

14.36.1 Usage

This function removes the singleton dimensions of an array. The syntax for its use is

```
y = squeeze(x)
```

where `x` is a multidimensional array. Generally speaking, if `x` is of size `d1 x 1 x d2 x ...`, then `squeeze(x)` is of size `d1 x d2 x ...`, i.e., each dimension of `x` that was singular (size 1) is squeezed out.

14.36.2 Example

Here is a many dimensioned, ungainly array, both before and after squeezing;

```
--> x = zeros(1,4,3,1,1,2);
--> size(x)

ans =
 1 4 3 1 1 2

--> y = squeeze(x);
--> size(y)

ans =
 4 3 2
```

14.37 SUBSREF Array Dereferencing

14.37.1 Usage

This function can be used to index into basic array types (or structures). It provides a functional interface to execute complex indexing expressions such as `a.b(3){5}` at run time (i.e. while executing a script or a function) without resorting to using `eval`. Note that this function should be overloaded for use with user defined classes, and that it cannot be overloaeded for base types. The basic syntax of the function is:

```
b = subsref(a,s)
```

where `s` is a structure array with two fields. The first field is

- `type` is a string containing either `'()'` or `'{}'` or `'.'` depending on the form of the call.
- `subs` is a cell array or string containing the the subscript information.

When multiple indexing experssions are combined together such as `b = a(5).foo{:}`, the `s` array should contain the following entries

```
s(1).type = '()'  s(1).subs = {5}
s(2).type = '.'   s(2).subs = 'foo'
s(3).type = '{}'  s(3).subs = ':'
```

14.38 TRACE Sum Diagonal Elements of an Array

14.38.1 Usage

Returns the sum of the diagonal elements of a square matrix. The general syntax for its use is

```
y = trace(x)
```

where x is a square matrix.

14.39 TRANSPOSE Matrix Transpose

14.39.1 Usage

Performs a (nonconjugate) transpose of a matrix. The syntax for its use is

```
y = transpose(x)
```

and is a synonym for `y = x.'`.

14.39.2 Example

Here is an example of the transpose of a complex matrix. Note that the entries are not conjugated.

```
--> A = [1+i,2+i;3-2*i,4+2*i]

A =
   1.0000 +  1.0000i   2.0000 +  1.0000i
   3.0000 -  2.0000i   4.0000 +  2.0000i

--> transpose(A)

ans =
   1.0000 +  1.0000i   3.0000 -  2.0000i
   2.0000 +  1.0000i   4.0000 +  2.0000i
```

14.40 TRIL Lower Triangular Matrix Function

14.40.1 Usage

Returns the lower triangular matrix of a square matrix. The general syntax for its use is

```
y = tril(x)
```

where —x— is a square matrix. This returns the lower triangular matrix (i.e.: all cells on or above the diagonal are set to 0). You can also specify a different diagonal using the alternate form

```
y = tril(x,n)
```

where n is the diagonal offset. In this mode, the diagonal specified is not set to zero in the returned matrix (e.g.: tril(x) and tril(x,-1)) will return the same value.

14.41 TRIU Upper Triangular Matrix Function

14.41.1 Usage

Returns the upper triangular matrix of a square matrix. The general syntax for its use is

```
y = triu(x)
```

where x is a square matrix. This returns the upper triangular matrix (i.e.: all elements on or below the diagonal are set to 0). You can also specify a different diagonal using the alternate form

```
y = triu(x,n)
```

where n is the diagonal offset. In this mode, the diagonal specified is not set to zero in the returned matrix (e.g.: tril(x) and tril(x,1)) will return the same value.

14.42 UNIQUE Unique

14.42.1 Usage

Returns a vector containing the unique elements of an array. The first form is simply

```
y = unique(x)
```

where `x` is either a numerical array or a cell-array of strings. The result is sorted in increasing order. You can also retrieve two sets of index vectors

```
[y, m, n] = unique(x)
```

such that `y = x(m)` and `x = y(n)`. If the argument `x` is a matrix, you can also indicate that FreeMat should look for unique rows in the matrix via

```
y = unique(x,'rows')
```

and

```
[y, m, n] = unique(x,'rows')
```

14.42.2 Example

Here is an example in row mode

```
--> A = randi(1,3*ones(15,3))

A =
 2 3 2
 2 1 1
 2 2 3
 2 1 3
 2 2 3
 2 1 2
 1 2 2
 1 1 1
 3 1 3
 2 2 2
 1 3 3
 1 2 3
 3 1 1
 3 3 1
 2 3 3

--> unique(A,'rows')

ans =
 1 1 1
 1 2 2
 1 2 3
 1 3 3
 2 1 1
 2 1 2
 2 1 3
 2 2 2
 2 2 3
 2 3 2
 2 3 3
 3 1 1
 3 1 3
 3 3 1

--> [b,m,n] = unique(A,'rows');
--> b

ans =
```

```
 1 1 1
 1 2 2
 1 2 3
 1 3 3
 2 1 1
 2 1 2
 2 1 3
 2 2 2
 2 2 3
 2 3 2
 2 3 3
 3 1 1
 3 1 3
 3 3 1

--> A(m,:)

ans =
 1 1 1
 1 2 2
 1 2 3
 1 3 3
 2 1 1
 2 1 2
 2 1 3
 2 2 2
 2 2 3
 2 3 2
 2 3 3
 3 1 1
 3 1 3
 3 3 1

--> b(n,:)

ans =
 2 3 2
 2 1 1
 2 2 3
 2 1 3
 2 2 3
 2 1 2
 1 2 2
 1 1 1
 3 1 3
 2 2 2
 1 3 3
 1 2 3
 3 1 1
 3 3 1
 2 3 3
```

Here is an example in vector mode

```
--> A = randi(1,5*ones(10,1))
```

```
A =
 5
 5
 5
 3
 5
 3
 4
 1
 3
 2

--> unique(A)

ans =
 1
 2
 3
 4
 5

--> [b,m,n] = unique(A,'rows');
--> b

ans =
 1
 2
 3
 4
 5

--> A(m)

ans =
 1
 2
 3
 4
 5

--> b(n)

ans =
 5
 5
 5
 3
 5
 3
 4
 1
 3
 2
```

For cell arrays of strings.

```
--> A = {'hi','bye','good','tell','hi','bye'}

A =
 [hi] [bye] [good] [tell] [hi] [bye]

--> unique(A)

ans =
 [bye] [good] [hi] [tell]
```

14.43 XNRM2 BLAS Norm Calculation

14.43.1 Usage

Calculates the 2-norm of a vector. The syntax for its use is

```
y = xnrm2(A)
```

where `A` is the n-dimensional array to analyze. This form uses the underlying BLAS implementation to compute the 2-norm.

14.44 ZEROS Array of Zeros

14.44.1 Usage

Creates an array of zeros of the specified size. Two seperate syntaxes are possible. The first syntax specifies the array dimensions as a sequence of scalar dimensions:

```
y = zeros(d1,d2,...,dn).
```

The resulting array has the given dimensions, and is filled with all zeros. The type of `y` is `double`, a 64-bit floating point array. To get arrays of other types, use the typecast functions (e.g., `uint8`, `int8`, etc.). An alternative syntax is to use the following notation:

```
y = zeros(d1,d2,...,dn,classname)
```

where `classname` is one of 'double', 'single', 'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'int64', 'uint64', 'float', 'logical'.

The second syntax specifies the array dimensions as a vector, where each element in the vector specifies a dimension length:

```
y = zeros([d1,d2,...,dn]),
```

or

```
y = zeros([d1,d2,...,dn],classname).
```

This syntax is more convenient for calling `zeros` using a variable for the argument. In both cases, specifying only one dimension results in a square matrix output.

14.44.2 Example

The following examples demonstrate generation of some zero arrays using the first form.

```
--> zeros(2,3,2)

ans =

(:,:,1) =
 0 0 0
 0 0 0

(:,:,2) =
 0 0 0
 0 0 0

--> zeros(1,3)

ans =
 0 0 0
```

The same expressions, using the second form.

```
--> zeros([2,6])

ans =
 0 0 0 0 0 0
 0 0 0 0 0 0

--> zeros([1,3])

ans =
 0 0 0
```

Finally, an example of using the type casting function `uint16` to generate an array of 16-bit unsigned integers with zero values.

```
--> uint16(zeros(3))

ans =
 0 0 0
 0 0 0
 0 0 0
```

Here we use the second syntax where the class of the output is specified explicitly

```
--> zeros(3,'int16')

ans =
 0 0 0
 0 0 0
 0 0 0
```

Chapter 15

Random Number Generation

15.1 RAND Uniform Random Number Generator

15.1.1 Usage

Creates an array of pseudo-random numbers of the specified size. The numbers are uniformly distributed on `[0,1)`. Two seperate syntaxes are possible. The first syntax specifies the array dimensions as a sequence of scalar dimensions:

```
y = rand(d1,d2,...,dn).
```

The resulting array has the given dimensions, and is filled with random numbers. The type of `y` is `double`, a 64-bit floating point array. To get arrays of other types, use the typecast functions.

The second syntax specifies the array dimensions as a vector, where each element in the vector specifies a dimension length:

```
y = rand([d1,d2,...,dn]).
```

This syntax is more convenient for calling `rand` using a variable for the argument.

Finally, `rand` supports two additional forms that allow you to manipulate the state of the random number generator. The first retrieves the state

```
y = rand('state')
```

which is a 625 length integer vector. The second form sets the state

```
rand('state',y)
```

or alternately, you can reset the random number generator with

```
rand('state',0)
```

15.1.2 Example

The following example demonstrates an example of using the first form of the `rand` function.

```
--> rand(2,2,2)

ans =

(:,:,1) =
    0.8539    0.1733
    0.0415    0.1300
```

```
(:,:,2) =
    0.7163    0.5752
    0.5953    0.3728
```

The second example demonstrates the second form of the `rand` function.

```
--> rand([2,2,2])

ans =

(:,:,1) =
    0.4992    0.2797
    0.6513    0.3209

(:,:,2) =
    0.6244    0.7774
    0.0934    0.1820
```

The third example computes the mean and variance of a large number of uniform random numbers. Recall that the mean should be `1/2`, and the variance should be `1/12 ~ 0.083`.

```
--> x = rand(1,10000);
--> mean(x)

ans =
    0.4952

--> var(x)

ans =
    0.0845
```

Now, we use the state manipulation functions of `rand` to exactly reproduce a random sequence. Note that unlike using `seed`, we can exactly control where the random number generator starts by saving the state.

```
--> rand('state',0)    % restores us to startup conditions
--> a = rand(1,3)      % random sequence 1

a =
    0.3759    0.0183    0.9134

--> b = rand('state'); % capture the state vector
--> c = rand(1,3)      % random sequence 2

c =
    0.3580    0.7604    0.8077

--> rand('state',b);   % restart the random generator so...
--> c = rand(1,3)      % we get random sequence 2 again

c =
    0.3580    0.7604    0.8077
```

15.2 RANDBETA Beta Deviate Random Number Generator

15.2.1 Usage

Creates an array of beta random deviates based on the supplied two parameters. The general syntax for `randbeta` is

```
y = randbeta(alpha, beta)
```

where `alpha` and `beta` are the two parameters of the random deviate. There are three forms for calling `randbeta`. The first uses two vectors `alpha` and `beta` of the same size, in which case the output `y` is the same size as both inputs, and each deviate uses the corresponding values of `alpha` and `beta` from the arguments. In the other forms, either `alpha` or `beta` are scalars.

15.2.2 Function Internals

The probability density function (PDF) of a beta random variable is

$$f(x) = x^(a-1) * (1-x)^(b-1)/B(a,b)$$

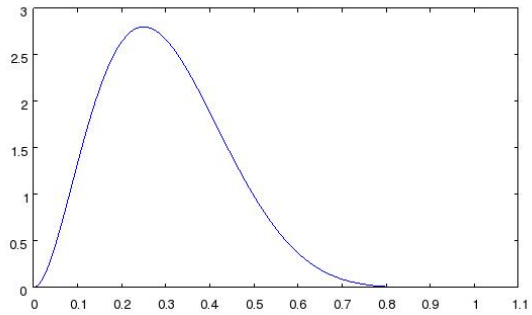
for `x` between 0 and 1. The function `B(a,b)` is defined so that the integral of `f(x)` is 1.

15.2.3 Example

Here is a plot of the PDF of a beta random variable with `a=3`, `b=7`.

```
--> a = 3; b = 7;
--> x = (0:100)/100; t = x.^(a-1).*(1-x).^(b-1);
--> t = t/(sum(t)*.01);
--> plot(x,t);
```

which is plotted as



If we generate a few random deviates with these values, we see they are distributed around the peak of roughly `0.25`.

```
--> randbeta(3*ones(1,5),7*ones(1,5))

ans =
    0.4343    0.4220    0.3992    0.2727    0.2475
```

15.3 RANDBIN Generate Binomial Random Variables

15.3.1 Usage

Generates random variables with a binomial distribution. The general syntax for its use is

```
y = randbin(N,p)
```

where `N` is a vector representing the number of Bernoulli trials, and `p` is the success probability associated with each trial.

15.3.2 Function Internals

A Binomial random variable describes the number of successful outcomes from `N` Bernoulli trials, with the probability of success in each trial being `p`. The probability distribution is

$$P(n) = \frac{N!}{n!(N-n)!} p^n (1-p)^{N-n}$$

15.3.3 Example

Here we generate `10` binomial random variables, corresponding to `N=100` trials, each with probability `p=0.1`, using both `randbin` and then again using `rand` (to simulate the trials):

```
--> randbin(100,.1*ones(1,10))

ans =
  6  7  6  7 13  7  7 10 13 15

--> sum(rand(100,10)<0.1)

ans =
 11  9  8  9 15 16 11 17  4  7
```

15.4 RANDCHI Generate Chi-Square Random Variable

15.4.1 Usage

Generates a vector of chi-square random variables with the given number of degrees of freedom. The general syntax for its use is

```
y = randchi(n)
```

where `n` is an array containing the degrees of freedom for each generated random variable.

15.4.2 Function Internals

A chi-square random variable is essentially distributed as the squared Euclidean norm of a vector of standard Gaussian random variables. The number of degrees of freedom is generally the number of elements in the vector. In general, the PDF of a chi-square random variable is

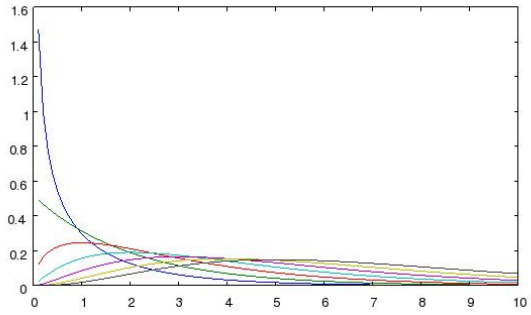
$$f(x) = \frac{x^{r/2-1} e^{-x/2}}{\Gamma(r/2) 2^{r/2}}$$

15.4.3 Example

First, a plot of the PDF for a family of chi-square random variables

```
--> f = zeros(7,100);
--> x = (1:100)/10;
--> for n=1:7;t=x.^(n/2-1).*exp(-x/2);f(n,:)=10*t/sum(t);end
--> plot(x,f');
```

The PDF is below:



Here is an example of using `randchi` and `randn` to compute some chi-square random variables with four degrees of freedom.

```
--> randchi(4*ones(1,6))

ans =
    2.6122    6.2362    0.8717    1.4935    6.0370    5.2771

--> sum(randn(4,6).^2)

ans =
    0.0399    4.6296    0.8697    0.5796    1.5490    5.8538
```

15.5 RANDEXP Generate Exponential Random Variable

15.5.1 Usage

Generates a vector of exponential random variables with the specified parameter. The general syntax for its use is

```
y = randexp(lambda)
```

where `lambda` is a vector containing the parameters for the generated random variables.

15.5.2 Function Internals

The exponential random variable is usually associated with the waiting time between events in a Poisson random process. The PDF of an exponential random variable is:

$$f(x) = \lambda e^{-\lambda x}$$

15.5.3 Example

Here is an example of using the `randexp` function to generate some exponentially distributed random variables

```
--> randexp(ones(1,6))

ans =
    0.7969    0.2401    0.5891    1.5129    0.9998    2.7738
```

15.6 RANDF Generate F-Distributed Random Variable

15.6.1 Usage

Generates random variables with an F-distribution. The general syntax for its use is

```
y = randf(n,m)
```

where `n` and `m` are vectors of the number of degrees of freedom in the numerator and denominator of the chi-square random variables whose ratio defines the statistic.

15.6.2 Function Internals

The statistic `F\_{n,m}` is defined as the ratio of two chi-square random variables:

$$F_{n,m} = \frac{\chi_n^2/n}{\chi_m^2/m}$$

The PDF is given by

$$f_{n,m} = \frac{m^{m/2} n^{n/2} x^{n/2-1}}{(m+nx)^{(n+m)/2} B(n/2, m/2)},$$

where `B(a,b)` is the beta function.

15.6.3 Example

Here we use `randf` to generate some F-distributed random variables, and then again using the `randchi` function:

```
--> randf(5*ones(1,9),7)

ans =
    0.5241    0.8414    0.4859    1.1266    0.4792    2.3743    2.9095    0.5825    0.4244

--> randchi(5*ones(1,9))./randchi(7*ones(1,9))

ans =
    0.3737    0.2363    1.5733    0.7003    1.1385    0.6337    0.4597    0.2691    0.5190
```

15.7 RANDGAMMA Generate Gamma-Distributed Random Variable

15.7.1 Usage

Generates random variables with a gamma distribution. The general syntax for its use is

```
y = randgamma(a,r),
```

where `a` and `r` are vectors describing the parameters of the gamma distribution. Roughly speaking, if `a` is the mean time between changes of a Poisson random process, and we wait for the `r` change, the resulting wait time is Gamma distributed with parameters `a` and `r`.

15.7.2 Function Internals

The Gamma distribution arises in Poisson random processes. It represents the waiting time to the occurance of the `r`-th event in a process with mean time `a` between events. The probability distribution of a Gamma random variable is

$$P(x) = \frac{a^r x^{r-1} e^{-ax}}{\Gamma(r)}.$$

Note also that for integer values of `r` that a Gamma random variable is effectively the sum of `r` exponential random variables with parameter `a`.

15.7.3 Example

Here we use the `randgamma` function to generate Gamma-distributed random variables, and then generate them again using the `randexp` function.

```
--> randgamma(1,15*ones(1,9))

ans =
   10.0227   12.4783   18.0388   21.7056   14.1249   15.9260   22.0177   15.9170   24.3781

--> sum(randexp(ones(15,9)))

ans =
   14.5031   12.8908   10.5201   16.9976    9.8463   12.7479   13.6879   21.7005   11.4172
```

15.8 RANDI Uniformly Distributed Integer

15.8.1 Usage

Generates an array of uniformly distributed integers between the two supplied limits. The general syntax for `randi` is

```
y = randi(low,high)
```

where `low` and `high` are arrays of integers. Scalars can be used for one of the arguments. The output `y` is a uniformly distributed pseudo-random number between `low` and `high` (inclusive).

15.8.2 Example

Here is an example of a set of random integers between zero and 5:

```
--> randi(zeros(1,6),5*ones(1,6))

ans =
 5 5 0 4 4 2
```

15.9 RANDMULTI Generate Multinomial-distributed Random Variables

15.9.1 Usage

This function generates samples from a multinomial distribution given the probability of each outcome. The general syntax for its use is

```
y = randmulti(N,pvec)
```

where `N` is the number of experiments to perform, and `pvec` is the vector of probabilities describing the distribution of outcomes.

15.9.2 Function Internals

A multinomial distribution describes the number of times each of `m` possible outcomes occurs out of `N` trials, where each outcome has a probability `p\_i`. More generally, suppose that the probability of a Bernoulli random variable `X\_i` is `p\_i`, and that

$$\sum_{i=1}^{m} p_i = 1.$$

Then the probability that `X\_i` occurs `x\_i` times is

$$P_N(x_1, x_2, \ldots, x_n) = \frac{N!}{x_1! \cdots x_n!} p_1^{x_1} \cdots p_n^{x_n}.$$

15.9.3 Example

Suppose an experiment has three possible outcomes, say heads, tails and edge, with probabilities `0.4999`, `0.4999` and `0.0002`, respectively. Then if we perform ten thousand coin flips we get

```
--> randmulti(10000,[0.4999,0.4999,0.0002])

ans =
 5051    0 4948
```

15.10 RANDN Gaussian (Normal) Random Number Generator

15.10.1 Usage

Creates an array of pseudo-random numbers of the specified size. The numbers are normally distributed with zero mean and a unit standard deviation (i.e., `mu = 0, sigma = 1`). Two seperate syntaxes are possible. The first syntax specifies the array dimensions as a sequence of scalar dimensions:

```
y = randn(d1,d2,...,dn).
```

The resulting array has the given dimensions, and is filled with random numbers. The type of `y` is `double`, a 64-bit floating point array. To get arrays of other types, use the typecast functions.

The second syntax specifies the array dimensions as a vector, where each element in the vector specifies a dimension length:

```
y = randn([d1,d2,...,dn]).
```

This syntax is more convenient for calling `randn` using a variable for the argument.

Finally, `randn` supports two additional forms that allow you to manipulate the state of the random number generator. The first retrieves the state

```
y = randn('state')
```

which is a 625 length integer vector. The second form sets the state

```
randn('state',y)
```

or alternately, you can reset the random number generator with

```
randn('state',0)
```

15.10.2 Function Internals

Recall that the probability density function (PDF) of a normal random variable is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}.$$

The Gaussian random numbers are generated from pairs of uniform random numbers using a transformation technique.

15.10.3 Example

The following example demonstrates an example of using the first form of the `randn` function.

```
--> randn(2,2,2)

ans =

(:,:,1) =
   -1.3838    0.9091
   -1.1738    0.1705

(:,:,2) =
   -0.0336    0.4572
    0.7566   -1.1720
```

The second example demonstrates the second form of the `randn` function.

```
--> randn([2,2,2])

ans =

(:,:,1) =
    1.2183   -0.5558
    0.1605    0.1819

(:,:,2) =
    0.5727   -0.5929
   -0.3895   -0.2424
```

In the next example, we create a large array of 10000 normally distributed pseudo-random numbers. We then shift the mean to 10, and the variance to 5. We then numerically calculate the mean and variance using `mean` and `var`, respectively.

```
--> x = 10+sqrt(5)*randn(1,10000);
--> mean(x)

ans =
   10.0370

--> var(x)

ans =
    4.9402
```

Now, we use the state manipulation functions of `randn` to exactly reproduce a random sequence. Note that unlike using `seed`, we can exactly control where the random number generator starts by saving the state.

```
--> randn('state',0)    % restores us to startup conditions
--> a = randn(1,3)      % random sequence 1

a =
   -0.0362   -0.1404    0.6934

--> b = randn('state'); % capture the state vector
--> c = randn(1,3)      % random sequence 2
```

```
c =
    0.5998    0.7086   -0.9394

--> randn('state',b);   % restart the random generator so...
--> c = randn(1,3)      % we get random sequence 2 again

c =
    0.5998    0.7086   -0.9394
```

15.11 RANDNBIN Generate Negative Binomial Random Variables

15.11.1 Usage

Generates random variables with a negative binomial distribution. The general syntax for its use is

```
y = randnbin(r,p)
```

where `r` is a vector of integers representing the number of successes, and `p` is the probability of success.

15.11.2 Function Internals

A negative binomial random variable describes the number of failures `x` that occur in `x+r` bernoulli trials, with a success on the `x+r` trial. The pdf is given by

$$P_{r,p}(x) = \binom{x+r-1}{r-1} p^r (1-p)^x.$$

15.11.3 Example

Here we generate some negative binomial random variables:

```
--> randnbin(3*ones(1,4),.01)

ans =
 437 215 199 187

--> randnbin(6*ones(1,4),.01)

ans =
  471 1233  768  338
```

15.12 RANDNCHI Generate Noncentral Chi-Square Random Variable

15.12.1 Usage

Generates a vector of non-central chi-square random variables with the given number of degrees of freedom and the given non-centrality parameters. The general syntax for its use is

```
y = randnchi(n,mu)
```

where `n` is an array containing the degrees of freedom for each generated random variable (with each element of `n` ¿= 1), and `mu` is the non-centrality shift (must be positive).

15.12.2 Function Internals

A non-central chi-square random variable is the sum of a chisquare deviate with `n-1` degrees of freedom plus the square of a normal deviate with mean `mu` and standard deviation 1.

15.12.3 Examples

Here is an example of a non-central chi-square random variable:

```
--> randnchi(5*ones(1,9),0.3)

ans =
   12.8187    1.5030    1.9084    3.6028    1.1185    4.1872    4.5202    3.4539    0.4578
```

15.13 RANDNF Generate Noncentral F-Distribution Random Variable

15.13.1 Usage

Generates a vector of non-central F-distributed random variables with the specified parameters. The general syntax for its use is

```
y = randnf(n,m,c)
```

where `n` is the number of degrees of freedom in the numerator, and `m` is the number of degrees of freedom in the denominator. The vector `c` determines the non-centrality shift of the numerator.

15.13.2 Function Internals

A non-central F-distributed random variable is the ratio of a non-central chi-square random variable and a central chi-square random variable, i.e.,

$$F_{n,m,c} = \frac{\chi^2_{n,c}/n}{\chi^2_m/m}.$$

15.13.3 Example

Here we use the `randf` to generate some non-central F-distributed random variables:

```
--> randnf(5*ones(1,9),7,1.34)

ans =
    0.5880    1.6093    0.4639    0.7857    2.5543    0.5044    3.3383    1.4102    1.1489
```

15.14 RANDP Generate Poisson Random Variable

15.14.1 Usage

Generates a vector Poisson random variables with the given parameters. The general syntax for its use is

```
y = randp(nu),
```

where `nu` is an array containing the rate parameters for the generated random variables.

15.14.2 Function Internals

A Poisson random variable is generally defined by taking the limit of a binomial distribution as the sample size becomes large, with the expected number of successes being fixed (so that the probability of success decreases as `1/N`). The Poisson distribution is given by

$$P_\nu(n) = \frac{\nu^n e^{-nu}}{n!}.$$

15.14.3 Example

Here is an exmaple of using `randp` to generate some Poisson random variables, and also using `randbin` to do the same using `N=1000` trials to approximate the Poisson result.

```
--> randp(33*ones(1,10))

ans =
 39 39 27 27 35 31 29 28 33 25

--> randbin(1000*ones(1,10),33/1000*ones(1,10))

ans =
 31 17 42 19 34 36 34 41 30 30
```

15.15 RANDPERM Random permutation

15.15.1 USAGE

```
   y = randperm(n)
```

`y` is a random permutation of integers from 1 to `n`. `randperm` calls `rand` and changes its state.

15.15.2 Example

```
--> y = randperm(10)

y =
  2  5 10  8  1  6  3  7  9  4
```

15.16 SEED Seed the Random Number Generator

15.16.1 Usage

Seeds the random number generator using the given integer seeds. Changing the seed allows you to choose which pseudo-random sequence is generated. The seed takes two `uint32` values:

```
  seed(s,t)
```

where `s` and `t` are the seed values. Note that due to limitations in `ranlib`, the values of `s,t` must be between `0 <= s,t <= 2\^30`.

15.16.2 Example

Here's an example of how the seed value can be used to reproduce a specific random number sequence.

```
--> seed(32,41);
--> rand(1,5)

ans =
    0.8589    0.3727    0.5551    0.9557    0.7367

--> seed(32,41);
--> rand(1,5)

ans =
    0.8589    0.3727    0.5551    0.9557    0.7367
```

Chapter 16

Input/Ouput Functions

16.1 CSVREAD Read Comma Separated Value (CSV) File

16.1.1 Usage

The `csvread` function reads a text file containing comma separated values (CSV), and returns the resulting numeric matrix (2D). The function supports multiple syntaxes. The first syntax for `csvread` is

```
x = csvread('filename')
```

which attempts to read the entire CSV file into array `x`. The file can contain only numeric values. Each entry in the file should be separated from other entries by a comma. However, FreeMat will attempt to make sense of the entries if the comma is missing (e.g., a space separated file will also parse correctly). For complex values, you must be careful with the spaces). The second form of `csvread` allows you to specify the first row and column (zero-based index)

```
x = csvread('filename',firstrow,firstcol)
```

The last form allows you to specify the range to read also. This form is

```
x = csvread('filename',firstrow,firstcol,readrange)
```

where `readrange` is either a 4-vector of the form `[R1,C1,R2,C2]`, where `R1,C1` is the first row and column to use, and `R2,C2` is the last row and column to use. You can also specify the `readrange` as a spreadsheet range `B12..C34`, in which case the index for the range is 1-based (as in a typical spreadsheet), so that `A1` is the first cell in the upper left corner. Note also that `csvread` is somewhat limited.

16.1.2 Example

Here is an example of a CSV file that we wish to read in

```
    sample_data.csv
10, 12, 13, 00, 45, 16
09, 11, 52, 93, 05, 06
01, 03, 04, 04, 90, -3
14, 17, 13, 67, 30, 43
21, 33, 14, 44, 01, 00
```

We start by reading the entire file

```
--> csvread('sample_data.csv')

ans =
 10 12 13  0 45 16
```

```
  9 11 52 93  5  6
  1  3  4  4 90 -3
 14 17 13 67 30 43
 21 33 14 44  1  0
```

Next, we read everything starting with the second row, and third column

```
--> csvread('sample_data.csv',1,2)

ans =
 52 93  5  6
  4  4 90 -3
 13 67 30 43
 14 44  1  0
```

Finally, we specify that we only want the `3 x 3` submatrix starting with the second row, and third column

```
--> csvread('sample_data.csv',1,2,[1,2,3,4])

ans =
 52 93  5
  4  4 90
 13 67 30
```

16.2 CSVWRITE Write Comma Separated Value (CSV) File

16.2.1 Usage

The `csvwrite` function writes a given matrix to a text file using comma separated value (CSV) notation. Note that you can create CSV files with arbitrary sized matrices, but that `csvread` has limits on line length. If you need to reliably read and write large matrices, use `rawwrite` and `rawread` respectively. The syntax for `csvwrite` is

```
csvwrite('filename',x)
```

where `x` is a numeric array. The contents of `x` are written to `filename` as comma-separated values. You can also specify a row and column offset to `csvwrite` to force `csvwrite` to write the matrix `x` starting at the specified location in the file. This syntax of the function is

```
csvwrite('filename',x,startrow,startcol)
```

where `startrow` and `startcol` are the offsets in zero-based indexing.

16.2.2 Example

Here we create a simple matrix, and write it to a CSV file

```
--> x = [1,2,3;5,6,7]

x =
 1 2 3
 5 6 7

--> csvwrite('csvwrite.csv',x)
--> csvread('csvwrite.csv')

ans =
 1 2 3
 5 6 7
```

Next, we do the same with an offset.

```
--> csvwrite('csvwrite.csv',x,1,2)
--> csvread('csvwrite.csv')

ans =
 0 0 0 0
 0 1 2 3
 0 5 6 7
```

Note the extra zeros.

16.3 DISP Display a Variable or Expression

16.3.1 Usage

Displays the result of a set of expressions. The `disp` function takes a variable number of arguments, each of which is an expression to output:

```
disp(expr1,expr2,...,exprn)
```

This is functionally equivalent to evaluating each of the expressions without a semicolon after each.

16.3.2 Example

Here are some simple examples of using `disp`.

```
--> a = 32;
--> b = 1:4;
--> disp(a,b,pi)
 32

 1 2 3 4

    3.1416
```

16.4 DLMREAD Read ASCII-delimited File

16.4.1 Usage

Loads a matrix from an ASCII-formatted text file with a delimiter between the entries. This function is similar to the `load -ascii` command, except that it can handle complex data, and it allows you to specify the delimiter. Also, you can read only a subset of the data from the file. The general syntax for the `dlmread` function is

```
y = dlmread(filename)
```

where `filename` is a string containing the name of the file to read. In this form, FreeMat will guess at the type of the delimiter in the file. The guess is made by examining the input for common delimiter characters, which are `,;:` or a whitespace (e.g., tab). The text in the file is preprocessed to replace these characters with whitespace and the file is then read in using a whitespace for the delimiter.

If you know the delimiter in the file, you can specify it using this form of the function:

```
y = dlmread(filename, delimiter)
```

where `delimiter` is a string containing the delimiter. If `delimiter` is the empty string, then the delimiter is guessed from the file.

You can also read only a portion of the file by specifying a start row and start column:

```
y = dlmread(filename, delimiter, startrow, startcol)
```

where `startrow` and `startcol` are zero-based. You can also specify the data to read using a range argument:

```
y = dlmread(filename, delimiter, range)
```

where `range` is either a vector `[startrow,startcol,stoprow,stopcol]` or is specified in spreadsheet notation as `B4..ZA5`.

Note that complex numbers can be present in the file if they are encoded without whitespaces inside the number, and use either `i` or `j` as the indicator. Note also that when the delimiter is given, each incidence of the delimiter counts as a separator. Multiple separators generate zeros in the matrix.

16.5 FCLOSE File Close Function

16.5.1 Usage

Closes a file handle, or all open file handles. The general syntax for its use is either

```
fclose(handle)
```

or

```
fclose('all')
```

In the first case a specific file is closed, In the second, all open files are closed. Note that until a file is closed the file buffers are not flushed. Returns a '0' if the close was successful and a '-1' if the close failed for some reason.

16.5.2 Example

A simple example of a file being opened with `fopen` and then closed with `fclose`.

```
--> fp = fopen('test.dat','wb','ieee-le')

fp =
 3

--> fclose(fp)
```

16.6 FEOF End Of File Function

16.6.1 Usage

Check to see if we are at the end of the file. The usage is

```
b = feof(handle)
```

The `handle` argument must be a valid and active file handle. The return is true (logical 1) if the current position is at the end of the file, and false (logical 0) otherwise. Note that simply reading to the end of a file will not cause `feof` to return `true`. You must read past the end of the file (which will cause an error anyway). See the example for more details.

16.6.2 Example

Here, we read to the end of the file to demonstrate how `feof` works. At first pass, we force a read of the contents of the file by specifying `inf` for the dimension of the array to read. We then test the end of file, and somewhat counter-intuitively, the answer is `false`. We then attempt to read past the end of the file, which causes an error. An `feof` test now returns the expected value of `true`.

```
--> fp = fopen('test.dat','rb');
--> x = fread(fp,[512,inf],'float');
--> feof(fp)

ans =
 1

--> x = fread(fp,[1,1],'float');
--> feof(fp)

ans =
 1
```

16.7 FFLUSH Force File Flush

16.7.1 Usage

Flushes any pending output to a given file. The general use of this function is

```
fflush(handle)
```

where `handle` is an active file handle (as returned by `fopen`).

16.8 FGETLINE Read a String from a File

16.8.1 Usage

Reads a string from a file. The general syntax for its use is

```
s = fgetline(handle)
```

This function reads characters from the file `handle` into a `string` array `s` until it encounters the end of the file or a newline. The newline, if any, is retained in the output string. If the file is at its end, (i.e., that `feof` would return true on this handle), `fgetline` returns an empty string.

16.8.2 Example

First we write a couple of strings to a test file.

```
--> fp = fopen('testtext','w');
--> fprintf(fp,'String 1\n');
--> fprintf(fp,'String 2\n');
--> fclose(fp);
```

Next, we read then back.

```
--> fp = fopen('testtext','r')

fp =
 4
```

```
--> fgetline(fp)

ans =
String 1

--> fgetline(fp)

ans =
String 2

--> fclose(fp);
```

16.9 FOPEN File Open Function

16.9.1 Usage

Opens a file and returns a handle which can be used for subsequent file manipulations. The general syntax for its use is

```
fp = fopen(fname,mode,byteorder)
```

Here `fname` is a string containing the name of the file to be opened. `mode` is the mode string for the file open command. The first character of the mode string is one of the following:

- `'r'` Open file for reading. The file pointer is placed at the beginning of the file. The file can be read from, but not written to.
- `'r+'` Open for reading and writing. The file pointer is placed at the beginning of the file. The file can be read from and written to, but must exist at the outset.
- `'w'` Open file for writing. If the file already exists, it is truncated to zero length. Otherwise, a new file is created. The file pointer is placed at the beginning of the file.
- `'w+'` Open for reading and writing. The file is created if it does not exist, otherwise it is truncated to zero length. The file pointer placed at the beginning of the file.
- `'a'` Open for appending (writing at end of file). The file is created if it does not exist. The file pointer is placed at the end of the file.
- `'a+'` Open for reading and appending (writing at end of file). The file is created if it does not exist. The file pointer is placed at the end of the file.

Starting with FreeMat 4, all files are treated as binary files by default. To invoke the operating systems 'CR/LF ¡-¿ CR' translation (on Win32) add a 't' to the mode string, as in 'rt+'.

Also, you can supply a second argument to `fopen` to retrieve error messages if the `fopen` fails.

```
[fp,messages] = fopen(fname,mode,byteorder)
```

Finally, FreeMat has the ability to read and write files of any byte-sex (endian). The third (optional) input indicates the byte-endianness of the file. If it is omitted, the native endian-ness of the machine running FreeMat is used. Otherwise, the third argument should be one of the following strings:

- `'le','ieee-le','little-endian','littleEndian','little','l','ieee-le.l64','s'`
- `'be','ieee-be','big-endian','bigEndian','big','b','ieee-be.l64','a'`

If the file cannot be opened, or the file mode is illegal, then an error occurs. Otherwise, a file handle is returned (which is an integer). This file handle can then be used with `fread`, `fwrite`, or `fclose` for file access.

Note that three handles are assigned at initialization time:

- Handle 0 - is assigned to standard input
- Handle 1 - is assigned to standard output
- Handle 2 - is assigned to standard error

These handles cannot be closed, so that user created file handles start at `3`.

16.9.2 Examples

Here are some examples of how to use `fopen`. First, we create a new file, which we want to be little-endian, regardless of the type of the machine. We also use the `fwrite` function to write some floating point data to the file.

```
--> fp = fopen('test.dat','w','ieee-le')

fp =
 4

--> fwrite(fp,float([1.2,4.3,2.1]))

ans =
 12

--> fclose(fp)
```

Next, we open the file and read the data back

```
--> fp = fopen('test.dat','r','ieee-le')

fp =
 4

--> fread(fp,[1,3],'float')

ans =
    1.2000    4.3000    2.1000

--> fclose(fp)
```

Now, we re-open the file in append mode and add two additional `float`s to the file.

```
--> fp = fopen('test.dat','a+','le')

fp =
 4

--> fwrite(fp,float([pi,e]))

ans =
 8

--> fclose(fp)
```

Finally, we read all 5 `float` values from the file

```
--> fp = fopen('test.dat','r','ieee-le')

fp =
 4

--> fread(fp,[1,5],'float')

ans =
    1.2000    4.3000    2.1000    3.1416    2.7183

--> fclose(fp)
```

16.10 FORMAT Control the Format of Matrix Display

16.10.1 Usage

FreeMat supports several modes for displaying matrices (either through the `disp` function or simply by entering expressions on the command line. There are several options for the format command. The default mode is equivalent to

```
format short
```

which generally displays matrices with 4 decimals, and scales matrices if the entries have magnitudes larger than roughly `1e2` or smaller than `1e-2`. For more information you can use

```
format long
```

which displays roughly 7 decimals for `float` and `complex` arrays, and 14 decimals for `double` and `dcomplex`. You can also use

```
format short e
```

to get exponential format with 4 decimals. Matrices are not scaled for exponential formats. Similarly, you can use

```
format long e
```

which displays the same decimals as `format long`, but in exponential format. You can also use the `format` command to retrieve the current format:

```
s = format
```

where `s` is a string describing the current format.

16.10.2 Example

We start with the short format, and two matrices, one of double precision, and the other of single precision.

```
--> format short
--> a = randn(4)

a =
   -0.3756    0.0920    0.9516    1.8527
    0.5078   -0.2088   -0.3120   -0.2380
    0.5578    0.7695    0.0226    2.9326
   -0.4420   -0.4871   -0.7582   -0.5059
```

```
--> b = float(randn(4))

b =
    0.2010    0.3416    0.1562   -0.5460
    1.2842   -0.3808   -1.2720   -0.3398
   -0.7660   -0.6251    2.4811    0.7956
   -0.1727    0.8577    1.5701   -1.5048
```

Note that in the short format, these two matrices are displayed with the same format. In `long` format, however, they display differently

```
--> format long
--> a

ans =
  -0.37559630424227   0.09196341864118   0.95155934364300   1.85265231634028
   0.50776589164635  -0.20877480315311  -0.31198760445638  -0.23799081322695
   0.55783547335483   0.76954243414671   0.02264031516947   2.93263318869123
  -0.44202929771190  -0.48708606879623  -0.75822963661106  -0.50590405332950

--> b

ans =
   0.2010476   0.3415550   0.1561587  -0.5460028
   1.2841575  -0.3808453  -1.2719837  -0.3397521
  -0.7659672  -0.6251388   2.4811494   0.7956446
  -0.1726678   0.8576548   1.5701485  -1.5048176
```

Note also that we we scale the contents of the matrices, FreeMat rescales the entries with a scale premultiplier.

```
--> format short
--> a*1e4

ans =

   1.0e+04 *
   -0.3756    0.0920    0.9516    1.8527
    0.5078   -0.2088   -0.3120   -0.2380
    0.5578    0.7695    0.0226    2.9326
   -0.4420   -0.4871   -0.7582   -0.5059

--> a*1e-4

ans =

   1.0e-04 *
   -0.3756    0.0920    0.9516    1.8527
    0.5078   -0.2088   -0.3120   -0.2380
    0.5578    0.7695    0.0226    2.9326
   -0.4420   -0.4871   -0.7582   -0.5059

--> b*1e4

ans =
```

```
   1.0e+04 *
    0.2010    0.3416    0.1562   -0.5460
    1.2842   -0.3808   -1.2720   -0.3398
   -0.7660   -0.6251    2.4811    0.7956
   -0.1727    0.8577    1.5701   -1.5048

--> b*1e-4

ans =

   1.0e-04 *
    0.2010    0.3416    0.1562   -0.5460
    1.2842   -0.3808   -1.2720   -0.3398
   -0.7660   -0.6251    2.4811    0.7956
   -0.1727    0.8577    1.5701   -1.5048
```

Next, we use the exponential formats:

```
--> format short e
--> a*1e4

ans =
 -3.7560e+03  9.1963e+02  9.5156e+03  1.8527e+04
  5.0777e+03 -2.0877e+03 -3.1199e+03 -2.3799e+03
  5.5784e+03  7.6954e+03  2.2640e+02  2.9326e+04
 -4.4203e+03 -4.8709e+03 -7.5823e+03 -5.0590e+03

--> a*1e-4

ans =
 -3.7560e-05  9.1963e-06  9.5156e-05  1.8527e-04
  5.0777e-05 -2.0877e-05 -3.1199e-05 -2.3799e-05
  5.5784e-05  7.6954e-05  2.2640e-06  2.9326e-04
 -4.4203e-05 -4.8709e-05 -7.5823e-05 -5.0590e-05

--> b*1e4

ans =
  2.0105e+03  3.4156e+03  1.5616e+03 -5.4600e+03
  1.2842e+04 -3.8085e+03 -1.2720e+04 -3.3975e+03
 -7.6597e+03 -6.2514e+03  2.4811e+04  7.9564e+03
 -1.7267e+03  8.5765e+03  1.5701e+04 -1.5048e+04

--> b*1e-4

ans =
  2.0105e-05  3.4155e-05  1.5616e-05 -5.4600e-05
  1.2842e-04 -3.8085e-05 -1.2720e-04 -3.3975e-05
 -7.6597e-05 -6.2514e-05  2.4811e-04  7.9564e-05
 -1.7267e-05  8.5765e-05  1.5701e-04 -1.5048e-04
```

Finally, if we assign the `format` function to a variable, we can retrieve the current format:

```
--> format short
--> t = format
```

```
t =
short
```

16.11 FPRINTF Formated File Output Function (C-Style)

16.11.1 Usage

Prints values to a file. The general syntax for its use is

```
fprintf(fp,format,a1,a2,...).
```

or,

```
fprintf(format,a1,a2,...).
```

Here `format` is the format string, which is a string that controls the format of the output. The values of the variables `ai` are substituted into the output as required. It is an error if there are not enough variables to satisfy the format string. Note that this `fprintf` command is not vectorized! Each variable must be a scalar. The value `fp` is the file handle. If `fp` is omitted, file handle `1` is assumed, and the behavior of `fprintf` is effectively equivalent to `printf`. For more details on the format string, see `printf`.

16.11.2 Examples

A number of examples are present in the Examples section of the `printf` command.

16.12 FREAD File Read Function

16.12.1 Usage

Reads a block of binary data from the given file handle into a variable of a given shape and precision. The general use of the function is

```
A = fread(handle,size,precision)
```

The `handle` argument must be a valid value returned by the fopen function, and accessable for reading. The `size` argument determines the number of values read from the file. The `size` argument is simply a vector indicating the size of the array `A`. The `size` argument can also contain a single `inf` dimension, indicating that FreeMat should calculate the size of the array along that dimension so as to read as much data as possible from the file (see the examples listed below for more details). The data is stored as columns in the file, not rows.

Alternately, you can specify two return values to the `fread` function, in which case the second value contains the number of elements read

```
[A,count] = fread(...)
```

where `count` is the number of elements in `A`.

The third argument determines the type of the data. Legal values for this argument are listed below:

- 'uint8','uchar','unsigned char' for an unsigned, 8-bit integer.
- 'int8','char','integer*1' for a signed, 8-bit integer.
- 'uint16','unsigned short' for an unsigned, 16-bit integer.
- 'int16','short','integer*2' for a signed, 16-bit integer.
- 'uint32','unsigned int' for an unsigned, 32-bit integer.
- 'int32','int','integer*4' for a signed, 32-bit integer.

- 'single','float32','float','real*4' for a 32-bit floating point.
- 'double','float64','real*8' for a 64-bit floating point.

Starting with FreeMat 4, the format for the third argument has changed. If you specify only a type, such as `'float'`, the data is read in as single precision, but the output type is always `'double'`. This behavior is consistent with Matlab. If you want the output type to match the input type (as was previous behavior in FreeMat), you must preface the precision string with a `'*'`. Thus, the precision string `'*float'` implies that data is read in as single precision, and the output is also single precision.

The third option is to specify the input and output types explicitly. You can do this by specifiying a precision string of the form `'type1 => type2'`, where `'type1'` is the input type and `'type2'` is the output type. For example, if the input type is `'double'` and the output type is to be a `'float'`, then a type spec of `'double => float'` should be used.

16.12.2 Example

First, we create an array of `512 x 512` Gaussian-distributed `float` random variables, and then writing them to a file called `test.dat`.

```
--> A = float(randn(512));
--> fp = fopen('test.dat','w');
--> fwrite(fp,A);
--> fclose(fp);
```

Read as many floats as possible into a row vector

```
--> fp = fopen('test.dat','r');
--> x = fread(fp,[1,inf],'float');
--> fclose(fp);
--> who x
  Variable Name       Type   Flags             Size
              x    double                   [1x262144]
```

Note that `x` is a `double` array. This behavior is new to FreeMat 4. Read the same floats into a 2-D float array.

```
--> fp = fopen('test.dat','r');
--> x = fread(fp,[512,inf],'float');
--> fclose(fp);
--> who x
  Variable Name       Type   Flags             Size
              x    double                   [512x512]
```

To read them as a single precision float array, we can use the following form:

```
--> fp = fopen('test.dat','r');
--> x = fread(fp,[512,inf],'*float');
--> fclose(fp);
--> who x
  Variable Name       Type   Flags             Size
              x    single                   [512x512]
```

16.13 FSCANF Formatted File Input Function (C-Style)

16.13.1 Usage

Reads values from a file. The general syntax for its use is

```
[a,count] = fscanf(handle,format,[size])
```

Here `format` is the format string, which is a string that controls the format of the input, `size` specifies the amount of data to be read. Values that are parsed from the `text` are stored in a. Note that fscanf is vectorized - the format string is reused as long as there are entries in the `text` string. See `printf` for a description of the format. Note that if the file is at the end-of-file, the fscanf will return

16.14 FSEEK Seek File To A Given Position

16.14.1 Usage

Moves the file pointer associated with the given file handle to the specified offset (in bytes). The usage is

```
fseek(handle,offset,style)
```

The `handle` argument must be a value and active file handle. The `offset` parameter indicates the desired seek offset (how much the file pointer is moved in bytes). The `style` parameter determines how the offset is treated. Three values for the `style` parameter are understood:

- string `'bof'` or the value -1, which indicate the seek is relative to the beginning of the file. This is equivalent to `SEEK\_SET` in ANSI C.
- string `'cof'` or the value 0, which indicates the seek is relative to the current position of the file. This is equivalent to `SEEK\_CUR` in ANSI C.
- string `'eof'` or the value 1, which indicates the seek is relative to the end of the file. This is equivalent to `SEEK\_END` in ANSI C.

The offset can be positive or negative.

16.14.2 Example

The first example reads a file and then "rewinds" the file pointer by seeking to the beginning. The next example seeks forward by 2048 bytes from the files current position, and then reads a line of 512 floats.

```
--> % First we create the file
--> fp = fopen('test.dat','wb');
--> fwrite(fp,float(rand(4096,1)));
--> fclose(fp);
--> % Now we open it
--> fp = fopen('test.dat','rb');
--> % Read the whole thing
--> x = fread(fp,[1,inf],'float');
--> % Rewind to the beginning
--> fseek(fp,0,'bof');
--> % Read part of the file
--> y = fread(fp,[1,1024],'float');
--> who x y
  Variable Name       Type   Flags             Size
              x    double                  [1x4096]
              y    double                  [1x1024]
--> % Seek 2048 bytes into the file
--> fseek(fp,2048,'cof');
--> % Read 512 floats from the file
--> x = fread(fp,[512,1],'float');
--> % Close the file
--> fclose(fp);
```

16.15 FTELL File Position Function

16.15.1 Usage

Returns the current file position for a valid file handle. The general use of this function is

```
n = ftell(handle)
```

The `handle` argument must be a valid and active file handle. The return is the offset into the file relative to the start of the file (in bytes).

16.15.2 Example

Here is an example of using `ftell` to determine the current file position. We read 512 4-byte floats, which results in the file pointer being at position 512*4 = 2048.

```
--> fp = fopen('test.dat','wb');
--> fwrite(fp,randn(512,1));
--> fclose(fp);
--> fp = fopen('test.dat','rb');
--> x = fread(fp,[512,1],'float');
--> ftell(fp)

ans =
 2048
```

16.16 FWRITE File Write Function

16.16.1 Usage

Writes an array to a given file handle as a block of binary (raw) data. The general use of the function is

```
n = fwrite(handle,A)
```

The `handle` argument must be a valid value returned by the fopen function, and accessable for writing. The array `A` is written to the file a column at a time. The form of the output data depends on (and is inferred from) the precision of the array `A`. If the write fails (because we ran out of disk space, etc.) then an error is returned. The output `n` indicates the number of elements successfully written.

Note that unlike MATLAB, FreeMat 4 does not default to `uint8` for writing arrays to files. Alternately, the type of the data to be written to the file can be specified with the syntax

```
n = fwrite(handle,A,type)
```

where `type` is one of the following legal values:

- 'uint8','uchar','unsigned char' for an unsigned, 8-bit integer.
- 'int8','char','integer*1' for a signed, 8-bit integer.
- 'uint16','unsigned short' for an unsigned, 16-bit integer.
- 'int16','short','integer*2' for a signed, 16-bit integer.
- 'uint32','unsigned int' for an unsigned, 32-bit integer.
- 'int32','int','integer*4' for a signed, 32-bit integer.
- 'single','float32','float','real*4' for a 32-bit floating point.
- 'double','float64','real*8' for a 64-bit floating point.

16.16.2 Example

Heres an example of writing an array of `512 x 512` Gaussian-distributed `float` random variables, and then writing them to a file called `test.dat`.

```
--> A = float(randn(512));
--> fp = fopen('test.dat','w');
--> fwrite(fp,A,'single');
--> fclose(fp);
```

16.17 GETLINE Get a Line of Input from User

16.17.1 Usage

Reads a line (as a string) from the user. This function has two syntaxes. The first is

```
a = getline(prompt)
```

where `prompt` is a prompt supplied to the user for the query. The second syntax omits the `prompt` argument:

```
a = getline
```

Note that this function requires command line input, i.e., it will only operate correctly for programs or scripts written to run inside the FreeMat GUI environment or from the X11 terminal. If you build a stand-alone application and expect it to operate cross-platform, do not use this function (unless you include the FreeMat console in the final application).

16.18 GETPRINTLIMIT Get Limit For Printing Of Arrays

16.18.1 Usage

Returns the limit on how many elements of an array are printed using either the `disp` function or using expressions on the command line without a semi-colon. The default is set to one thousand elements. You can increase or decrease this limit by calling `setprintlimit`. This function is provided primarily so that you can temporarily change the output truncation and then restore it to the previous value (see the examples).

```
n=getprintlimit
```

where `n` is the current limit in use.

16.18.2 Example

Here is an example of using `getprintlimit` along with `setprintlimit` to temporarily change the output behavior of FreeMat.

```
--> A = randn(100,1);
--> n = getprintlimit

n =
 1000

--> setprintlimit(5);
--> A

ans =
   -0.6933
   -0.9500
```

```
    0.0824
   -1.1740
   -0.3467
Print limit has been reached.  Use setprintlimit function to enable longer printouts
--> setprintlimit(n)
```

16.19 HTMLREAD Read an HTML Document into FreeMat

16.19.1 Usage

Given a filename, reads an HTML document, (attempts to) parse it, and returns the result as a FreeMat data structure. The syntax for its use is:

```
p = htmlread(filename)
```

where `filename` is a `string`. The resulting object `p` is a data structure containing the information in the document. Note that this function works by internally converting the HTML document into something closer to XHTML, and then using the XML parser to parse it. In some cases, the converted HTML cannot be properly parsed. In such cases, a third party tool such as "tidy" will probably do a better job.

16.20 IMREAD Read Image File To Matrix

16.20.1 Usage

Reads the image data from the given file into a matrix. Note that FreeMat's support for `imread` is not complete. Only some of the formats specified in the MATLAB API are implemented. The syntax for its use is

```
[A,map,alpha] = imread(filename)
```

where `filename` is the name of the file to read from. The returned arrays `A` contain the image data, `map` contains the colormap information (for indexed images), and `alpha` contains the alphamap (transparency). The returned values will depend on the type of the original image. Generally you can read images in the `jpg,png,xpm,ppm` and some other formats.

16.21 IMWRITE Write Matrix to Image File

16.21.1 Usage

Write the image data from the matrix into a given file. Note that FreeMat's support for `imwrite` is not complete. You can write images in the `jpg,png,xpm,ppm` and some other formats. The syntax for its use is

```
  imwrite(A, filename)
  imwrite(A, map, filename)
  imwrite(A, map, filename, 'Alpha', alpha)

or Octave-style syntax:
  imwrite(filename, A)
  imwrite(filename, A, map)
  imwrite(filename, A, map, alpha)
```

where `filename` is the name of the file to write to. The input array `A` contains the image data (2D for gray or indexed, and 3D for color). If `A` is an integer array (int8, uint8, int16, uint16, int32, uint32), the values of its elements should be within 0-255. If `A` is a floating-point array (float or double), the value of its elements should be in the range [0,1]. `map` contains the colormap information (for indexed images), and `alpha` the alphamap (transparency).

16.21.2 Example

Here is a simple example of `imread/imwrite`. First, we generate a grayscale image and save it to an image file.

```
--> a =  uint8(255*rand(64));
--> figure(1), image(a), colormap(gray)
--> title('image to save')
--> imwrite(a, 'test.bmp')
```

Then, we read image file and show it:

```
--> b = imread('test.bmp');
--> figure(2), image(b), colormap(gray)
--> title('loaded image')
```

16.22 INPUT Get Input From User

16.22.1 Usage

The `input` function is used to obtain input from the user. There are two syntaxes for its use. The first is

```
r = input('prompt')
```

in which case, the prompt is presented, and the user is allowed to enter an expression. The expression is evaluated in the current workspace or context (so it can use any defined variables or functions), and returned for assignment to the variable (`r` in this case). In the second form of the `input` function, the syntax is

```
r = input('prompt','s')
```

in which case the text entered by the user is copied verbatim to the output.

16.23 LOAD Load Variables From A File

16.23.1 Usage

Loads a set of variables from a file in a machine independent format. The `load` function takes one argument:

```
load filename,
```

or alternately,

```
load('filename')
```

This command is the companion to `save`. It loads the contents of the file generated by `save` back into the current context. Global and persistent variables are also loaded and flagged appropriately. By default, FreeMat assumes that files that end in a `.mat` or `.MAT` extension are MATLAB-formatted files. Also, FreeMat assumes that files that end in `.txt` or `.TXT` are ASCII files. For other filenames, FreeMat first tries to open the file as a FreeMat binary format file (as created by the `save` function). If the file fails to open as a FreeMat binary file, then FreeMat attempts to read it as an ASCII file.

You can force FreeMat to assume a particular format for the file by using alternate forms of the `load` command. In particular,

```
load -ascii filename
```

will load the data in file `filename` as an ASCII file (space delimited numeric text) loaded into a single variable in the current workspace with the name `filename` (without the extension).

For MATLAB-formatted data files, you can use

```
load -mat filename
```

which forces FreeMat to assume that `filename` is a MAT-file, regardless of the extension on the filename.

You can also specify which variables to load from a file (not from an ASCII file - only single 2-D variables can be successfully saved and retrieved from ASCII files) using the additional syntaxes of the `load` command. In particular, you can specify a set of variables to load by name

```
load filename Var_1 Var_2 Var_3 ...
```

where `Var\_n` is the name of a variable to load from the file. Alternately, you can use the regular expression syntax

```
load filename -regexp expr_1 expr_2 expr_3 ...
```

where `expr\_n` is a regular expression (roughly as expected by `regexp`). Note that a simpler regular expression mechanism is used for this syntax than the full mechanism used by the `regexp` command.

Finally, you can use `load` to create a variable containing the contents of the file, instead of automatically inserting the variables into the curent workspace. For this form of `load` you must use the function syntax, and capture the output:

```
V = load('arg1','arg2',...)
```

which returns a structure `V` with one field for each variable retrieved from the file. For ASCII files, `V` is a double precision matrix.

16.23.2 Example

Here is a simple example of `save`/`load`. First, we save some variables to a file.

```
--> D = {1,5,'hello'};
--> s = 'test string';
--> x = randn(512,1);
--> z = zeros(512);
--> who
  Variable Name       Type   Flags             Size
             D       cell                    [1x3]
             s       char                   [1x11]
             x     double                 [512x1]
             z     double                 [512x512]
--> save loadsave.dat
```

Next, we clear the variables, and then load them back from the file.

```
--> clear D s x z
--> who
  Variable Name       Type   Flags             Size
           ans     double                   [0x0]
--> load loadsave.dat
--> who
  Variable Name       Type   Flags             Size
             D       cell                    [1x3]
           ans     double                   [0x0]
             s       char                   [1x11]
             x     double                 [512x1]
             z     double                 [512x512]
```

16.24 PAUSE Pause Script Execution

16.24.1 Usage

The `pause` function can be used to pause execution of FreeMat scripts. There are several syntaxes for its use. The first form is

```
pause
```

This form of the `pause` function pauses FreeMat until you press any key. The second form of the `pause` function takes an argument

```
pause(p)
```

where `p` is the number of seconds to pause FreeMat for. The pause argument should be accurate to a millisecond on all supported platforms. Alternately, you can control all `pause` statements using:

```
pause on
```

which enables pauses and

```
pause off
```

which disables all `pause` statements, both with and without arguments.

16.25 PRINTF Formated Output Function (C-Style)

16.25.1 Usage

Prints values to the output. The general syntax for its use is

```
printf(format,a1,a2,...)
```

Here `format` is the format string, which is a string that controls the format of the output. The values of the variables `a\_i` are substituted into the output as required. It is an error if there are not enough variables to satisfy the format string. Note that this `printf` command is not vectorized! Each variable must be a scalar.

It is important to point out that the `printf` function does not add a newline (or carriage return) to the output by default. That can lead to some confusing behavior if you do not know what to expect. For example, the command `printf('Hello')` does not appear to produce any output. In fact, it does produce the text, but it then gets overwritten by the prompt. To see the text, you need `printf('Hello\\n')`. This seems odd, but allows you to assemble a line using multiple `printf` commands, including the `'\\n'` when you are done with the line. You can also use the `'\r'` character as an explicit carriage return (with no line feed). This allows you to write to the same line many times (to show a progress string, for example).

16.25.2 Format of the format string

The format string is a character string, beginning and ending in its initial shift state, if any. The format string is composed of zero or more directives: ordinary characters (not unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character conversion specifier. In between there may be (in this order) zero or more flags, an optional minimum field width, and an optional precision.

The arguments must correspond properly (after type promotion) with the conversion specifier, and are used in the order given.

16.25.3 The flag characters

The character % is followed by zero or more of the following flags:

- \# The value should be converted to an "alternate form". For o conversions, the first character of the output string is made zero (by prefixing a 0 if it was not zero already). For x and X conversions, a nonzero result has the string '0x' (or '0X' for X conversions) prepended to it. For a, A, e, E, f, F, g, and G conversions, the result will always contain a decimal point, even if no digits follow it (normally, a decimal point appears in the results of those conversions only if a digit follows). For g and G conversions, trailing zeros are not removed from the result as they would otherwise be. For other conversions, the result is undefined.
- 0 The value should be zero padded. For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversions, the converted value is padded on the left with zeros rather than blanks. If the 0 and - flags both appear, the 0 flag is ignored. If a precision is given with a numeric conversion (d, i, o, u, x, and X), the 0 flag is ignored. For other conversions, the behavior is undefined.
- - The converted value is to be left adjusted on the field boundary. (The default is right justification.) Except for n conversions, the converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A - overrides a 0 if both are given.
- ' ' (a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.
- + A sign (+ or -) always be placed before a number produced by a signed conversion. By default a sign is used only for negative numbers. A + overrides a space if both are used.

16.25.4 The field width

An optional decimal digit string (with nonzero first digit) specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left-adjustment flag has been given). A negative field width is taken as a '-' flag followed by a positive field width. In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

16.25.5 The precision

An optional precision, in the form of a period ('.') followed by an optional decimal digit string. If the precision is given as just '.', or the precision is negative, the precision is taken to be zero. This gives the minimum number of digits to appear for d, i, o, u, x, and X conversions, the number of digits to appear after the radix character for a, A, e, E, f, and F conversions, the maximum number of significant digits for g and G conversions, or the maximum number of characters to be printed from a string for s conversions.

16.25.6 The conversion specifier

A character that specifies the type of conversion to be applied. The conversion specifiers and their meanings are:

- d,i The int argument is converted to signed decimal notation. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1. When 0 is printed with an explicit precision 0, the output is empty.
- o,u,x,X The unsigned int argument is converted to unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal (x and X) notation. The letters abcdef are used for x conversions; the letters ABCDEF are used for X conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1. When 0 is printed with an explicit precision 0, the output is empty.

- `e,E` The double argument is rounded and converted in the style `[-]d.ddde dd` where there is one digit before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as `6`; if the precision is zero, no decimal-point character appears. An `E` conversion uses the letter `E` (rather than `e`) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is `00`.
- `f,F` The double argument is rounded and converted to decimal notation in the style `[-]ddd.ddd`, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as `6`; if the precision is explicitly zero, no decimal-point character appears. If a decimal point appears, at least one digit appears before it.
- `g,G` The double argument is converted in style `f` or `e` (or `F` or `E` for `G` conversions). The precision specifies the number of significant digits. If the precision is missing, `6` digits are given; if the precision is zero, it is treated as `1`. Style e is used if the exponent from its conversion is less than `-4` or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal point appears only if it is followed by at least one digit.
- `c` The int argument is converted to an unsigned char, and the resulting character is written.
- `s` The string argument is printed.
- `%` A `'%'` is written. No argument is converted. The complete conversion specification is `'%%'`.

16.25.7 Example

Here are some examples of the use of `printf` with various arguments. First we print out an integer and double value.

```
--> printf('intvalue is %d, floatvalue is %f\n',3,1.53);
intvalue is 3, floatvalue is 1.530000
```

Next, we print out a string value.

```
--> printf('string value is %s\n','hello');
string value is hello
```

Now, we print out an integer using 12 digits, zeros up front.

```
--> printf('integer padded is %012d\n',32);
integer padded is 000000000032
```

Print out a double precision value with a sign, a total of 18 characters (zero prepended if necessary), a decimal point, and 12 digit precision.

```
--> printf('float value is %+018.12f\n',pi);
float value is +0003.141592653590
```

16.26 RAWREAD Read N-dimensional Array From File

16.26.1 Usage

The syntax for `rawread` is

```
function x = rawread(fname,size,precision,byteorder)
```

where `fname` is the name of the file to read from, and `size` is an n-dimensional vector that stores the size of the array in each dimension. The argument `precision` is the type of the data to read in:

- 'uint8','uchar','unsigned char' for unsigned, 8-bit integers

- 'int8','char','integer*1' for signed, 8-bit integers
- 'uint16','unsigned short' for unsigned, 16-bit integers
- 'int16','short','integer*2' for signed, 16-bit integers
- 'uint32','unsigned int' for unsigned, 32-bit integers
- 'int32','int','integer*4' for signed, 32-bit integers
- 'uint64','unsigned int' for unsigned, 64-bit integers
- 'int64','int','integer*8' for signed, 64-bit integers
- 'single','float32','float','real*4' for 32-bit floating point
- 'double','float64','real*8' for 64-bit floating point
- 'complex','complex*8' for 64-bit complex floating point (32 bits for the real and imaginary part).
- 'dcomplex','complex*16' for 128-bit complex floating point (64 bits for the real and imaginary part).

As a special feature, one of the size elements can be 'inf', in which case, the largest possible array is read in. If `byteorder` is left unspecified, the file is assumed to be of the same byte-order as the machine `FreeMat` is running on. If you wish to force a particular byte order, specify the `byteorder` argument as

- `'le','ieee-le','little-endian','littleEndian','little'`
- `'be','ieee-be','big-endian','bigEndian','big'`

16.27 RAWWRITE Write N-dimensional Array From File

16.27.1 Usage

The syntax for `rawwrite` is

```
function rawwrite(fname,x,byteorder)
```

where `fname` is the name of the file to write to, and the (numeric) array `x` is writen to the file in its native type (e.g. if `x` is of type `int16`, then it will be written to the file as 16-bit signed integers. If `byteorder` is left unspecified, the file is assumed to be of the same byte-order as the machine `FreeMat` is running on. If you wish to force a particular byte order, specify the `byteorder` argument as

- `'le','ieee-le','little-endian','littleEndian','little'`
- `'be','ieee-be','big-endian','bigEndian','big'`

16.28 SAVE Save Variables To A File

16.28.1 Usage

Saves a set of variables to a file in a machine independent format. There are two formats for the function call. The first is the explicit form, in which a list of variables are provided to write to the file:

```
save filename a1 a2 ...
```

In the second form,

```
save filename
```

all variables in the current context are written to the file. The format of the file is a simple binary encoding (raw) of the data with enough information to restore the variables with the `load` command. The endianness of the machine is encoded in the file, and the resulting file should be portable between machines of similar types (in particular, machines that support IEEE floating point representation).

You can also specify both the filename as a string, in which case you also have to specify the names of the variables to save. In particular

```
save('filename','a1','a2')
```

will save variables `a1` and `a2` to the file.

Starting with version 2.0, FreeMat can also read and write MAT files (the file format used by MATLAB) thanks to substantial work by Thomas Beutlich. Support for MAT files in version 2.1 has improved over previous versions. In particular, classes should be saved properly, as well as a broader range of sparse matrices. Compression is supported for both reading and writing to MAT files. MAT file support is still in the alpha stages, so please be cautious with using it to store critical data. The file format is triggered by the extension. To save files with a MAT format, simply use a filename with a ".mat" ending.

The `save` function also supports ASCII output. This is a very limited form of the save command - it can only save numeric arrays that are 2-dimensional. This form of the `save` command is triggered using

```
save -ascii filename var1 var 2
```

although where `-ascii` appears on the command line is arbitrary (provided it comes after the `save` command, of course). Be default, the `save` command uses an 8-digit exponential format notation to save the values to the file. You can specify that you want 16-digits using the

```
save -ascii -double filename var1 var2
```

form of the command. Also, by default, `save` uses spaces as the delimiters between the entries in the matrix. If you want tabs instead, you can use

```
save -ascii -tabs filename var1 var2
```

(you can also use both the `-tabs` and `-double` flags simultaneously).

Finally, you can specify that `save` should only save variables that match a particular regular expression. Any of the above forms can be combined with the `-regexp` flag:

```
save filename -regexp pattern1 pattern2
```

in which case variables that match any of the patterns will be saved.

16.28.2 Example

Here is a simple example of `save`/`load`. First, we save some variables to a file.

```
--> D = {1,5,'hello'};
--> s = 'test string';
--> x = randn(512,1);
--> z = zeros(512);
--> who
  Variable Name       Type   Flags             Size
              D       cell                    [1x3]
              s       char                    [1x11]
              x     double                    [512x1]
              z     double                    [512x512]
--> save loadsave.dat
```

Next, we clear the variables, and then load them back from the file.

```
--> clear D s x z
--> who
  Variable Name       Type   Flags             Size
           ans    double                    [0x0]
--> load loadsave.dat
--> who
  Variable Name       Type   Flags             Size
             D      cell                     [1x3]
           ans    double                     [0x0]
             s      char                     [1x11]
             x    double                     [512x1]
             z    double                     [512x512]
```

16.29 SETPRINTLIMIT Set Limit For Printing Of Arrays

16.29.1 Usage

Changes the limit on how many elements of an array are printed using either the `disp` function or using expressions on the command line without a semi-colon. The default is set to one thousand elements. You can increase or decrease this limit by calling

```
setprintlimit(n)
```

where `n` is the new limit to use.

16.29.2 Example

Setting a smaller print limit avoids pages of output when you forget the semicolon on an expression.

```
--> A = randn(512);
--> setprintlimit(10)
--> A

ans =

 Columns 1 to 12

   -0.2514   -0.1353   -1.3148    1.7915   -0.4740   -1.6836   -0.2605    0.1477   -0.1555    0.8534
Print limit has been reached.  Use setprintlimit function to enable longer printouts
--> setprintlimit(1000)
```

16.30 SPRINTF Formated String Output Function (C-Style)

16.30.1 Usage

Prints values to a string. The general syntax for its use is

```
y = sprintf(format,a1,a2,...).
```

Here `format` is the format string, which is a string that controls the format of the output. The values of the variables `a\_i` are substituted into the output as required. It is an error if there are not enough variables to satisfy the format string. Note that this `sprintf` command is not vectorized! Each variable must be a scalar. The returned value `y` contains the string that would normally have been printed. For more details on the format string, see `printf`.

16.30.2 Examples

Here is an example of a loop that generates a sequence of files based on a template name, and stores them in a cell array.

```
--> l = {}; for i = 1:5; s = sprintf('file_%d.dat',i); l(i) = {s}; end;
--> l

ans =
 [file_1.dat] [file_2.dat] [file_3.dat] [file_4.dat] [file_5.dat]
```

16.31 SSCANF Formated String Input Function (C-Style)

16.31.1 Usage

Reads values from a string. The general syntax for its use is

```
[a, count, errmsg, nextind] = sscanf(text,format,[size])
```

Here `format` is the format string, which is a string that controls the format of the input, `size` specifies the amount of data to be read. Values that are parsed from the `text` are stored in a. Note that sscanf is vectorized - the format string is reused as long as there are entries in the `text` string. See `printf` for a description of the format.

16.32 STR2NUM Convert a String to a Number

16.32.1 Usage

Converts a string to a number. The general syntax for its use is

```
x = str2num(string)
```

Here `string` is the data string, which contains the data to be converted into a number. The output is in double precision, and must be typecasted to the appropriate type based on what you need. Note that by definition, `str2num` is entirely equivalent to `eval(['[' string ']'],[])` with all of the associated problems where `string` contains text that causes side effects.

16.33 TYPE Type Contents of File or Function

16.33.1 Usage

Displays the contents of a file or a function to the screen or console. The syntax for its use is

```
type filename
type('filename')
```

or

```
type function
type('function')
```

in which case the function named `'function.m'` will be displayed.

16.33.2 Example

Here we use `type` to display the contents of itself

```
--> type('type')
%!
%@Module TYPE Type Contents of File or Function
%@@Section IO
%@@Usage
%Displays the contents of a file or a function to the screen
%or console.  The syntax for its use is
%@[
%   type filename
%   type('filename')
%@]
%or
%@[
%   type function
%   type('function')
%@]
%in which case the function named @|'function.m'| will be displayed.
%@@Example
%Here we use @|type| to display the contents of itself
%@<
%type('type')
%@>
%!

function type(filename)
fp = fopen(filename,'r');
if (fp == -1)
  f = which(filename);
  if (isempty(f)), return; end
  filename = f;
  fp = fopen(filename,'r');
end
if (fp == -1), return; end
while (~feof(fp))
  printf('%s',fgetline(fp));
end
fclose(fp);
```

16.34 URLWRITE Retrieve a URL into a File

16.34.1 Usage

Given a URL and a timeout, attempts to retrieve the URL and write the contents to a file. The syntax is

```
f = urlwrite(url,filename,timeout)
```

The `timeout` is in milliseconds. Note that the URL must be a complete spec (i.e., including the name of the resource you wish to retrieve). So for example, you cannot use `http://www.google.com` as a URL, but must instead use `http://www.google.com/index.html`.

16.35 WAVPLAY

16.35.1 Usage

Plays a linear PCM set of samples through the audio system. This function is only available if the `portaudio` library was available when FreeMat was built. The syntax for the command is one of:

```
wavplay(y)
wavplay(y,sampling_rate)
wavplay(...,mode)
```

where `y` is a matrix of audio samples. If `y` has two columns, then the audio playback is in stereo. The `y` input can be of types `float, double, int32, int16, int8, uint8`. For `float` and `double` types, the sample values in `y` must be between `-1` and `1`. The `sampling\_rate` specifies the rate at which the data is recorded. If not specified, the `sampling\_rate` defaults to `11025Hz`. Finally, you can specify a playback mode of `'sync'` which is synchronous playback or a playback mode of `'async'` which is asynchronous playback. For `'sync'` playback, the wavplay function returns when the playback is complete. For `'async'` playback, the function returns immediately (unless a former playback is still issuing).

16.36 WAVREAD Read a WAV Audio File

16.36.1 Usage

The `wavread` function (attempts) to read the contents of a linear PCM audio WAV file. This function could definitely use improvements - it is based on a very simplistic notion of a WAV file. The simplest form for its use is

```
y = wavread(filename)
```

where `filename` is the name of the WAV file to read. If no extension is provided, FreeMat will add a '.wav' extension. This loads the data from the WAV file into `y`, and returns it in `double` precision, normalized format. If you want additional information on, for example, the WAV sampling rate or bit depth, you can request it via

```
[y, SamplingRate, BitDepth] = wavread(filename)
```

where `SamplingRate` and `BitDepth` are the sampling rate (in Hz) and the bit depth of the original data in the WAV file. If you only want to load part of the WAV file, you can use

```
[...] = wavread(filename, N)
```

where `N` indicates the number of samples to read from the file. Alternately, you can indicate a range of samples to load via

```
[...] = wavread(filename, [N1 N2])
```

which returns only the indicated samples from each channel in the file. By default, the output format is `double` precision. You can cntrol the format of the output by indicating

```
[...] = wavread(filename, format)
```

where `format` is either `'double'` for double precision output, or `'native'` for native precision output (meaning whatever bitdepth that was present in the original file). Finally, you can use the `'size'` flag

```
y_siz = wavread(filename,'size')
```

which returns a vector `[samples channels]` indicating the size of the data present in the WAV file.

16.37 WAVRECORD

16.37.1 Usage

Records linear PCM sound from the audio system. This function is only available if the `portaudio` library was available when FreeMat was built. The syntax for this command is one of:

```
y = wavrecord(samples,rate)
y = wavrecord(...,channels)
y = wavrecord(...,'datatype')
```

where `samples` is the number of samples to record, and `rate` is the sampling rate. If not specified, the `rate` defaults to `11025Hz`. If you want to record in stero, specify `channels = 2`. Finally, you can specify the type of the recorded data (defaults to `FM\_DOUBLE`). Valid choices are `float, double, int32, int16, int8, uint8`.

16.38 WAVWRITE Write a WAV Audio File

16.38.1 Usage

The `wavwrite` funtion writes an audio signal to a linear PCM WAV file. The simplest form for its use is

```
wavwrite(y,filename)
```

which writes the data stored in `y` to a WAV file with the name `filename`. By default, the output data is assumed to be sampled at a rate of 8 KHz, and is output using 16 bit integer format. Each column of `y` is written as a separate channel. The data are clipped to the range `[-1,1]` prior to writing them out. If you want the data to be written with a different sampling frequency, you can use the following form of the `wavwrite` command:

```
wavwrite(y,SampleRate,filename)
```

where `SampleRate` is in Hz. Finally, you can specify the number of bits to use in the output form of the file using the form

```
wavwrite(y,SampleRate,NBits,filename)
```

where `NBits` is the number of bits to use. Legal values include `8,16,24,32`. For less than 32 bit output format, the data is truncated to the range `[-1,1]`, and an integer output format is used (type 1 PCM in WAV-speak). For 32 bit output format, the data is written in type 3 PCM as floating point data.

16.39 XMLREAD Read an XML Document into FreeMat

16.39.1 Usage

Given a filename, reads an XML document, parses it, and returns the result as a FreeMat data structure. The syntax for its use is:

```
p = xmlread(filename)
```

where `filename` is a `string`. The resulting object `p` is a data structure containing the information in the document. Note that the returned object `p` is not the same object as the one returned by MATLAB's `xmlread`, although the information content is the same. The output is largely compatible with the output of the parseXML example in the `xmlread` documentation of the MATLAB API.

Chapter 17

String Functions

17.1 BLANKS Create a blank string

17.1.1 Usage

```
str = blanks(n)
```

Create a string `str` containing `n` blank charaters.

17.1.2 Example

A simple example:

```
--> sprintf(['x0123456789y\n','x',blanks(10),'y\n'])

ans =
x0123456789y
x          y
```

17.2 CELLSTR Convert character array to cell array of strings

17.2.1 Usage

The `cellstr` converts a character array matrix into a a cell array of individual strings. Each string in the matrix is placed in a different cell, and extra spaces are removed. The syntax for the command is

```
y = cellstr(x)
```

where `x` is an `N x M` array of characters as a string.

17.2.2 Example

Here is an example of how to use `cellstr`

```
--> a = ['quick';'brown';'fox  ';'is   ']

a =
quick
brown
fox
is
--> cellstr(a)
```

```
ans =
 [quick]
 [brown]
 [fox]
 [is]
```

17.3 DEBLANK Remove trailing blanks from a string

17.3.1 Usage

The `deblank` function removes spaces at the end of a string when used with the syntax

```
y = deblank(x)
```

where `x` is a string, in which case, all of the extra spaces in `x` are stripped from the end of the string. Alternately, you can call `deblank` with a cell array of strings

```
y = deblank(c)
```

in which case each string in the cell array is deblanked.

17.3.2 Example

A simple example

```
--> deblank('hello   ')

ans =
hello
```

and a more complex example with a cell array of strings

```
--> deblank({'hello  ','there ','  is  ','  sign  '})

ans =
 [hello] [there] [  is] [  sign]
```

17.4 ISALPHA Test for Alpha Characters in a String

17.4.1 Usage

The `isalpha` functions returns a logical array that is 1 for characters in the argument string that are letters, and is a logical 0 for characters in the argument that are not letters. The syntax for its use is

```
x = isalpha(s)
```

where `s` is a `string`. Note that this function is not locale sensitive, and returns a logical 1 for letters in the classic ASCII sense (a through z, and A through Z).

17.4.2 Example

A simple example of `isalpha`:

```
--> isalpha('numb3r5')

ans =
 1 1 1 1 0 1 0
```

17.5 ISDIGIT Test for Digit Characters in a String

17.5.1 Usage

The `isdigit` functions returns a logical array that is 1 for characters in the argument string that are digits, and is a logical 0 for characters in the argument that are not digits. The syntax for its use is

```
x = isdigit(s)
```

where `s` is a `string`.

17.5.2 Example

A simple example of `isdigit`:

```
--> isdigit('numb3r5')

ans =
 0 0 0 0 1 0 1
```

17.6 ISSPACE Test for Space Characters in a String

17.6.1 Usage

The `isspace` functions returns a logical array that is 1 for characters in the argument string that are spaces, and is a logical 0 for characters in the argument that are not spaces. The syntax for its use is

```
x = isspace(s)
```

where `s` is a `string`. A blank character is considered a space, newline, tab, carriage return, formfeed, and vertical tab.

17.6.2 Example

A simple example of `isspace`:

```
--> isspace('  hello there world ')

ans =
 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1
```

17.7 LOWER Convert strings to lower case

17.7.1 Usage

The `lower` function converts a string to lower case with the syntax

```
y = lower(x)
```

where `x` is a string, in which case all of the upper case characters in `x` (defined as the range `'A'-'Z'`) are converted to lower case. Alternately, you can call `lower` with a cell array of strings

```
y = lower(c)
```

in which case each string in the cell array is converted to lower case.

17.7.2 Example

A simple example:

```
--> lower('this Is Strange CAPitalizaTion')

ans =
this is strange capitalization
```

and a more complex example with a cell array of strings

```
--> lower({'This','Is','Strange','CAPitalizaTion'})

ans =
 [this] [is] [strange] [capitalization]
```

17.8 REGEXP Regular Expression Matching Function

17.8.1 Usage

Matches regular expressions in the provided string. This function is complicated, and compatibility with MATLABs syntax is not perfect. The syntax for its use is

```
regexp('str','expr')
```

which returns a row vector containing the starting index of each substring of `str` that matches the regular expression described by `expr`. The second form of `regexp` returns six outputs in the following order:

```
[start stop tokenExtents match tokens names] = regexp('str','expr')
```

where the meaning of each of the outputs is defined below.

- `start` is a row vector containing the starting index of each substring that matches the regular expression.
- `stop` is a row vector containing the ending index of each substring that matches the regular expression.
- `tokenExtents` is a cell array containing the starting and ending indices of each substring that matches the `tokens` in the regular expression. A token is a captured part of the regular expression. If the `'once'` mode is used, then this output is a `double` array.
- `match` is a cell array containing the text for each substring that matches the regular expression. In `'once'` mode, this is a string.
- `tokens` is a cell array of cell arrays of strings that correspond to the tokens in the regular expression. In `'once'` mode, this is a cell array of strings.
- `named` is a structure array containing the named tokens captured in a regular expression. Each named token is assigned a field in the resulting structure array, and each element of the array corresponds to a different match.

If you want only some of the the outputs, you can use the following variant of `regexp`:

```
[o1 o2 ...] = regexp('str','expr', 'p1', 'p2', ...)
```

where `p1` etc. are the names of the outputs (and the order we want the outputs in). As a final variant, you can supply some mode flags to `regexp`

```
[o1 o2 ...] = regexp('str','expr', p1, p2, ..., 'mode1', 'mode2')
```

where acceptable `mode` flags are:

- 'once' - only the first match is returned.
- 'matchcase' - letter case must match (selected by default for regexp)
- 'ignorecase' - letter case is ignored (selected by default for regexpi)
- 'dotall' - the '.' operator matches any character (default)
- 'dotexceptnewline' - the '.' operator does not match the newline character
- 'stringanchors' - the \^ and $ operators match at the beginning and end (respectively) of a string.
- 'lineanchors' - the \^ and $ operators match at the beginning and end (respectively) of a line.
- 'literalspacing' - the space characters and comment characters \# are matched as literals, just like any other ordinary character (default).
- 'freespacing' - all spaces and comments are ignored in the regular expression. You must use ' ' and '#' to match spaces and comment characters, respectively.

Note the following behavior differences between MATLABs regexp and FreeMats:

- If you have an old version of pcre installed, then named tokens must use the older <?P<name> syntax, instead of the new <?<name> syntax.
- The pcre library is pickier about named tokens and their appearance in expressions. So, for example, the regexp from the MATLAB manual '(?<first>\\w+)\\s+(?<last>\\w+)(?¡last¿ w+), s+(?¡first¿ w+)'— does not work correctly (as of this writing) because the same named tokens appear multiple times. The workaround is to assign different names to each token, and then collapse the results later.

17.8.2 Example

Some examples of using the regexp function

```
--> [start,stop,tokenExtents,match,tokens,named] = regexp('quick down town zoo','(.)own')
start =
  7 12

stop =
 10 15

tokenExtents =
 [1x2 double array] [1x2 double array]

match =
 [down] [town]

tokens =
 [1x1 cell array] [1x1 cell array]

named =
  []
```

17.9 REGEXPREP Regular Expression Replacement Function

17.9.1 Usage

Replaces regular expressions in the provided string. The syntax for its use is

```
outstring = regexprep(instring,pattern,replacement,modes)
```

Here `instring` is the string to be operated on. And `pattern` is a regular expression of the type accepted by `regexp`. For each match, the contents of the matched string are replaced with the replacement text. Tokens in the regular expression can be used in the replacement text using `$N` where `N` is the number of the token to use. You can also specify the same `mode` flags that are used by `regexp`.

17.10 STRCMP String Compare Function

17.10.1 USAGE

Compares two strings for equality. The general syntax for its use is

```
p = strcmp(x,y)
```

where `x` and `y` are two strings. Returns `true` if `x` and `y` are the same size, and are equal (as strings). Otherwise, it returns `false`. In the second form, `strcmp` can be applied to a cell array of strings. The syntax for this form is

```
p = strcmp(cellstra,cellstrb)
```

where `cellstra` and `cellstrb` are cell arrays of a strings to compare. Also, you can also supply a character matrix as an argument to `strcmp`, in which case it will be converted via `cellstr` (so that trailing spaces are removed), before being compared.

17.10.2 Example

The following piece of code compares two strings:

```
--> x1 = 'astring';
--> x2 = 'bstring';
--> x3 = 'astring';
--> strcmp(x1,x2)

ans =
 0

--> strcmp(x1,x3)

ans =
 1
```

Here we use a cell array strings

```
--> x = {'astring','bstring',43,'astring'}

x =
 [astring] [bstring] [43] [astring]

--> p = strcmp(x,'astring')

p =
 1 0 0 1
```

Here we compare two cell arrays of strings

```
--> strcmp({'this','is','a','pickle'},{'what','is','to','pickle'})

ans =
 0 1 0 1
```

Finally, the case where one of the arguments is a matrix string

```
--> strcmp({'this','is','a','pickle'},['peter ';'piper ';'hated ';'pickle'])

ans =
 0 0 0 0
```

17.11 STRCMPI String Compare Case Insensitive Function

17.11.1 Usage

Compares two strings for equality ignoring case. The general syntax for its use is

```
p = strcmpi(x,y)
```

where `x` and `y` are two strings, or cell arrays of strings. See `strcmp` for more help.

17.12 STRFIND Find Substring in a String

17.12.1 Usage

Searches through a string for a pattern, and returns the starting positions of the pattern in an array. There are two forms for the `strfind` function. The first is for single strings

```
ndx = strfind(string, pattern)
```

the resulting array `ndx` contains the starting indices in `string` for the pattern `pattern`. The second form takes a cell array of strings

```
ndx = strfind(cells, pattern)
```

and applies the search operation to each string in the cell array.

17.12.2 Example

Here we apply `strfind` to a simple string

```
--> a = 'how now brown cow?'

a =
how now brown cow?
--> b = strfind(a,'ow')

b =
  2  6 11 16
```

Here we search over multiple strings contained in a cell array.

```
--> a = {'how now brown cow','quick brown fox','coffee anyone?'}

a =
 [how now brown cow] [quick brown fox] [coffee anyone?]

--> b = strfind(a,'ow')

b =
 [1x4 double array] [9] []
```

17.13 STRNCMP String Compare Function To Length N

17.13.1 USAGE

Compares two strings for equality, but only looks at the first N characters from each string. The general syntax for its use is

```
  p = strncmp(x,y,n)
```

where `x` and `y` are two strings. Returns `true` if `x` and `y` are each at least `n` characters long, and if the first `n` characters from each string are the same. Otherwise, it returns `false`. In the second form, `strncmp` can be applied to a cell array of strings. The syntax for this form is

```
  p = strncmp(cellstra,cellstrb,n)
```

where `cellstra` and `cellstrb` are cell arrays of a strings to compare. Also, you can also supply a character matrix as an argument to `strcmp`, in which case it will be converted via `cellstr` (so that trailing spaces are removed), before being compared.

17.13.2 Example

The following piece of code compares two strings:

```
--> x1 = 'astring';
--> x2 = 'bstring';
--> x3 = 'astring';
--> strncmp(x1,x2,4)

ans =
 0

--> strncmp(x1,x3,4)

ans =
 1
```

Here we use a cell array strings

```
--> x = {'ast','bst',43,'astr'}

x =
 [ast] [bst] [43] [astr]

--> p = strncmp(x,'ast',3)

p =
 1 0 0 1
```

Here we compare two cell arrays of strings

```
--> strncmp({'this','is','a','pickle'},{'think','is','to','pickle'},3)

ans =
 1 0 0 1
```

Finally, the case where one of the arguments is a matrix string

```
--> strncmp({'this','is','a','pickle'},['peter ';'piper ';'hated ';'pickle'],4);
```

17.14 STRREP String Replace Function

17.14.1 Usage

Replace every occurance of one string with another. The general syntax for its use is

```
p = strrep(source,find,replace)
```

Every instance of the string `find` in the string `source` is replaced with the string `replace`. Any of `source`, `find` and `replace` can be a cell array of strings, in which case each entry has the replace operation applied.

17.14.2 Example

Here are some examples of the use of `strrep`. First the case where are the arguments are simple strings

```
--> strrep('Matlab is great','Matlab','FreeMat')

ans =
FreeMat is great
```

And here we have the replace operation for a number of strings:

```
--> strrep({'time is money';'A stitch in time';'No time for games'},'time','money')

ans =
 [money is money]
 [A stitch in money]
 [No money for games]
```

17.15 STRSTR String Search Function

17.15.1 Usage

Searches for the first occurance of one string inside another. The general syntax for its use is

```
p = strstr(x,y)
```

where `x` and `y` are two strings. The returned integer `p` indicates the index into the string `x` where the substring `y` occurs. If no instance of `y` is found, then `p` is set to zero.

17.15.2 Example

Some examples of `strstr` in action

```
--> strstr('hello','lo')

ans =
 4

--> strstr('quick brown fox','own')

ans =
 9

--> strstr('free stuff','lunch')

ans =
 0
```

17.16 STRTRIM Trim Spaces from a String

17.16.1 Usage

Removes the white-spaces at the beginning and end of a string (or a cell array of strings). See `isspace` for a definition of a white-space. There are two forms for the `strtrim` function. The first is for single strings

```
y = strtrim(strng)
```

where `strng` is a string. The second form operates on a cell array of strings

```
y = strtrim(cellstr)
```

and trims each string in the cell array.

17.16.2 Example

Here we apply `strtrim` to a simple string

```
--> strtrim('  lot of blank spaces    ');
```

and here we apply it to a cell array

```
--> strtrim({'  space','enough ',' for ',''})

ans =
 [space] [enough] [for] []
```

17.17 UPPER Convert strings to upper case

17.17.1 Usage

The `upper` function converts a string to upper case with the syntax

```
y = upper(x)
```

where `x` is a string, in which case all of the lower case characters in `x` (defined as the range `'a'-'z'`) are converted to upper case. Alternately, you can call `upper` with a cell array of strings

```
y = upper(c)
```

in which case each string in the cell array is converted to upper case.

17.17.2 Example

A simple example:

```
--> upper('this Is Strange CAPitalizaTion')

ans =
THIS IS STRANGE CAPITALIZATION
```

and a more complex example with a cell array of strings

```
--> upper({'This','Is','Strange','CAPitalizaTion'})

ans =
 [THIS] [IS] [STRANGE] [CAPITALIZATION]
```

Chapter 18

Transforms/Decompositions

18.1 EIG Eigendecomposition of a Matrix

18.1.1 Usage

Computes the eigendecomposition of a square matrix. The `eig` function has several forms. The first returns only the eigenvalues of the matrix:

```
s = eig(A)
```

The second form returns both the eigenvectors and eigenvalues as two matrices (the eigenvalues are stored in a diagonal matrix):

```
[V,D] = eig(A)
```

where `D` is the diagonal matrix of eigenvalues, and `V` is the matrix of eigenvectors.

Eigenvalues and eigenvectors for asymmetric matrices `A` normally are computed with balancing applied. Balancing is a scaling step that normaly improves the quality of the eigenvalues and eigenvectors. In some instances (see the Function Internals section for more details) it is necessary to disable balancing. For these cases, two additional forms of `eig` are available:

```
s = eig(A,'nobalance'),
```

which computes the eigenvalues of `A` only, and does not balance the matrix prior to computation. Similarly,

```
[V,D] = eig(A,'nobalance')
```

recovers both the eigenvectors and eigenvalues of `A` without balancing. Note that the 'nobalance' option has no affect on symmetric matrices.

FreeMat also provides the ability to calculate generalized eigenvalues and eigenvectors. Similarly to the regular case, there are two forms for `eig` when computing generalized eigenvector (see the Function Internals section for a description of what a generalized eigenvector is). The first returns only the generalized eigenvalues of the matrix pair `A,B`

```
s = eig(A,B)
```

The second form also computes the generalized eigenvectors, and is accessible via

```
[V,D] = eig(A,B)
```

18.1.2 Function Internals

Recall that `v` is an eigenvector of `A` with associated eigenvalue `d` if

$$Av = dv.$$

This decomposition can be written in matrix form as

$$AV = VD$$

where

$$V = [v_1, v_2, \ldots, v_n], D = \mathrm{diag}(d_1, d_2, \ldots, d_n).$$

The `eig` function uses the `LAPACK` class of functions `GEEVX` to compute the eigenvalue decomposition for non-symmetric (or non-Hermitian) matrices `A`. For symmetric matrices, `SSYEV` and `DSYEV` are used for `float` and `double` matrices (respectively). For Hermitian matrices, `CHEEV` and `ZHEEV` are used for `complex` and `dcomplex` matrices.

For some matrices, the process of balancing (in which the rows and columns of the matrix are pre-scaled to facilitate the search for eigenvalues) is detrimental to the quality of the final solution. This is particularly true if the matrix contains some elements on the order of round off error. See the Example section for an example.

A generalized eigenvector of the matrix pair `A,B` is simply a vector `v` with associated eigenvalue `d` such that

$$Av = dBv,$$

where `B` is a square matrix of the same size as `A`. This decomposition can be written in matrix form as

$$AV = BVD$$

where

$$V = [v_1, v_2, \ldots, v_n], D = \mathrm{diag}(d_1, d_2, \ldots, d_n).$$

For general matrices `A` and `B`, the `GGEV` class of routines are used to compute the generalized eigendecomposition. If howevever, `A` and `B` are both symmetric (or Hermitian, as appropriate), Then FreeMat first attempts to use `SSYGV` and `DSYGV` for `float` and `double` arguments and `CHEGV` and `ZHEGV` for `complex` and `dcomplex` arguments (respectively). These routines requires that `B` also be positive definite, and if it fails to be, FreeMat will revert to the routines used for general arguments.

18.1.3 Example

Some examples of eigenvalue decompositions. First, for a diagonal matrix, the eigenvalues are the diagonal elements of the matrix.

```
--> A = diag([1.02f,3.04f,1.53f])

A =
    1.0200         0         0
         0    3.0400         0
         0         0    1.5300

--> eig(A)

ans =
    1.0200
    1.5300
    3.0400
```

Next, we compute the eigenvalues of an upper triangular matrix, where the eigenvalues are again the diagonal elements.

```
--> A = [1.0f,3.0f,4.0f;0,2.0f,6.7f;0.0f,0.0f,1.0f]

A =
    1.0000    3.0000    4.0000
         0    2.0000    6.7000
         0         0    1.0000

--> eig(A)

ans =
 1
 2
 1
```

Next, we compute the complete eigenvalue decomposition of a random matrix, and then demonstrate the accuracy of the solution

```
--> A = float(randn(2))

A =
    0.3747   -1.5129
   -0.6283   -1.1096

--> [V,D] = eig(A)
V =
    0.9526    0.6096
   -0.3042    0.7927

D =
    0.8578         0
         0   -1.5928

--> A*V - V*D

ans =

   1.0e-08 *
   -5.9605         0
   -2.9802         0
```

Now, we consider a matrix that requires the nobalance option to compute the eigenvalues and eigenvectors properly. Here is an example from MATLAB's manual.

```
--> B = [3,-2,-.9,2*eps;-2,4,1,-eps;-eps/4,eps/2,-1,0;-.5,-.5,.1,1]

B =
    3.0000   -2.0000   -0.9000    0.0000
   -2.0000    4.0000    1.0000   -0.0000
   -0.0000    0.0000   -1.0000         0
   -0.5000   -0.5000    0.1000    1.0000

--> [VB,DB] = eig(B)
VB =
    0.6153   -0.4176   -0.0000   -0.1495
   -0.7881   -0.3261   -0.0000    0.1316
   -0.0000   -0.0000    0.0000   -0.9570
```

```
    0.0189    0.8481    1.0000   -0.2110

DB =
    5.5616         0         0         0
         0    1.4384         0         0
         0         0    1.0000         0
         0         0         0   -1.0000

--> B*VB - VB*DB

ans =
         0         0    0.0000   -0.0000
         0    0.0000   -0.0000    0.0000
   -0.0000   -0.0000   -0.0000   -0.0000
    0.0000    0.0000    0.0000   -0.5088

--> [VN,DN] = eig(B,'nobalance')
VN =
    0.6153   -0.4176   -0.0000   -0.1528
   -0.7881   -0.3261         0    0.1345
   -0.0000   -0.0000   -0.0000   -0.9781
    0.0189    0.8481   -1.0000    0.0443

DN =
    5.5616         0         0         0
         0    1.4384         0         0
         0         0    1.0000         0
         0         0         0   -1.0000

--> B*VN - VN*DN

ans =

   1.0e-15 *
   -1.7764   -0.1110   -0.5587   -0.1665
    3.5527    1.0547    0.3364   -0.1943
    0.0172    0.0015    0.0066         0
    0.1527   -0.2220    0.2220    0.0833
```

18.2 FFT (Inverse) Fast Fourier Transform Function

18.2.1 Usage

Computes the Discrete Fourier Transform (DFT) of a vector using the Fast Fourier Transform technique. The general syntax for its use is

```
y = fft(x,n,d)
```

where `x` is an `n`-dimensional array of numerical type. Integer types are promoted to the `double` type prior to calculation of the DFT. The argument `n` is the length of the FFT, and `d` is the dimension along which to take the DFT. If —n— is larger than the length of `x` along dimension `d`, then `x` is zero-padded (by appending zeros) prior to calculation of the DFT. If `n` is smaller than the length of `x` along the given dimension, then `x` is truncated (by removing elements at the end) to length `n`.

If `d` is omitted, then the DFT is taken along the first non-singleton dimension of `x`. If `n` is omitted, then the DFT length is chosen to match of the length of `x` along dimension `d`.

Note that FFT support on Linux builds requires availability of the FFTW libraries at compile time. On Windows and Mac OS X, single and double precision FFTs are available by default.

18.2.2 Function Internals

The output is computed via

$$y(m_1,\ldots,m_{d-1},l,m_{d+1},\ldots,m_p) = \sum_{k=1}^{n} x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p)e^{-\frac{2\pi(k-1)l}{n}}.$$

For the inverse DFT, the calculation is similar, and the arguments have the same meanings as the DFT:

$$y(m_1,\ldots,m_{d-1},l,m_{d+1},\ldots,m_p) = \frac{1}{n}\sum_{k=1}^{n} x(m_1,\ldots,m_{d-1},k,m_{d+1},\ldots,m_p)e^{\frac{2\pi(k-1)l}{n}}.$$

The FFT is computed using the FFTPack library, available from netlib at `http://www.netlib.org`. Generally speaking, the computational cost for a FFT is (in worst case) `O(n\^2)`. However, if `n` is composite, and can be factored as

$$n = \prod_{k=1}^{p} m_k,$$

then the DFT can be computed in

$$O(n\sum_{k=1}^{p} m_k)$$

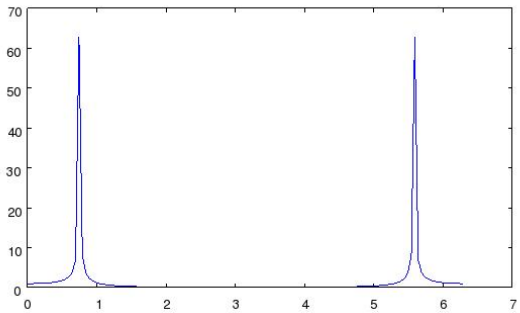
operations. If `n` is a power of 2, then the FFT can be calculated in `O(n log\_2 n)`. The calculations for the inverse FFT are identical.

18.2.3 Example

The following piece of code plots the FFT for a sinusoidal signal:

```
--> t = linspace(0,2*pi,128);
--> x = cos(15*t);
--> y = fft(x);
--> plot(t,abs(y));
```

The resulting plot is:



The FFT can also be taken along different dimensions, and with padding and/or truncation. The following example demonstrates the Fourier Transform being computed along each column, and then along each row.

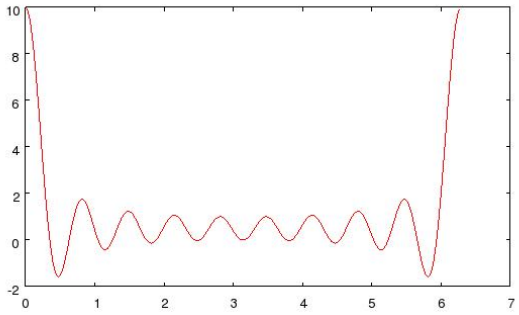
```
--> A = [2,5;3,6]

A =
 2 5
```

```
 3 6

--> real(fft(A,[],1))

ans =
  5 11
 -1 -1

--> real(fft(A,[],2))

ans =
  7 -3
  9 -3
```

Fourier transforms can also be padded using the `n` argument. This pads the signal with zeros prior to taking the Fourier transform. Zero padding in the time domain results in frequency interpolation. The following example demonstrates the FFT of a pulse (consisting of 10 ones) with (red line) and without (green circles) padding.

```
--> delta(1:10) = 1;
--> plot((0:255)/256*pi*2,real(fft(delta,256)),'r-');
--> hold on
--> plot((0:9)/10*pi*2,real(fft(delta)),'go');
```

The resulting plot is:



18.3 FFTN N-Dimensional Forward FFT

18.3.1 Usage

Computes the DFT of an N-dimensional numerical array along all dimensions. The general syntax for its use is

```
y = fftn(x)
```

which computes the same-size FFTs for each dimension of `x`. Alternately, you can specify the size vector

```
y = fftn(x,dims)
```

where `dims` is a vector of sizes. The array `x` is zero padded or truncated as necessary in each dimension so that the output is of size `dims`. The `fftn` function is implemented by a sequence of calls to `fft`.

18.4 FFTSHIFT Shift FFT Output

18.4.1 Usage

The `fftshift` function shifts the DC component (zero-frequency) of the output from an FFT to the center of the array. For vectors this means swapping the two halves of the vector. For matrices, the first and third quadrants are swapped. So on for N-dimensional arrays. The syntax for its use is

```
y = fftshift(x).
```

Alternately, you can specify that only one dimension be shifted

```
y = fftshift(x,dim).
```

18.5 HILBERT Hilbert Transform

18.5.1 Usage

The `hilbert` function computes the hilbert transform of the argument vector or matrix. The FreeMat `hilbert` function is compatible with the one from the MATLAB API. This means that the output of the `hilbert` function is the sum of the original function and an imaginary signal containing the Hilbert transform of it. There are two syntaxes for the hilbert function. The first is

```
y = hilbert(x)
```

where `x` is real vector or matrix. If `x` is a matrix, then he Hilbert transform is computed along the columns of `x`. The second syntax provides a dimension along which to take the transform.

```
y = hilbert(x,n)
```

where `n` is the dimension along which to apply the transformation.

18.6 IFFTN N-Dimensional Inverse FFT

18.6.1 Usage

Computes the inverse DFT of an N-dimensional numerical array along all dimensions. The general syntax for its use is

```
y = ifftn(x)
```

which computes the same-size inverse FFTs for each dimension of `x`. Alternately, you can specify the size vector

```
y = ifftn(x,dims)
```

where `dims` is a vector of sizes. The array `x` is zero padded or truncated as necessary in each dimension so that the output is of size `dims`. The `ifftn` function is implemented by a sequence of calls to `ifft`.

18.7 IFFTSHIFT Inverse Shift FFT Output

18.7.1 Usage

The `ifftshift` function shifts the DC component (zero-frequency) of the output from the center of the array back to the first position and iseffectively the inverse of `fftshift`. For vectors this means swapping the two halves of the vector. For matrices, the first and third quadrants are swapped. So on for N-dimensional arrays. The syntax for its use is

```
y = ifftshift(x).
```

Alternately, you can specify that only one dimension be shifted

```
y = ifftshift(x,dim).
```

18.8 INV Invert Matrix

18.8.1 Usage

Inverts the argument matrix, provided it is square and invertible. The syntax for its use is

```
y = inv(x)
```

Internally, the `inv` function uses the matrix divide operators. For sparse matrices, a sparse matrix solver is used.

18.8.2 Example

Here we invert some simple matrices

```
--> a = randi(zeros(3),5*ones(3))

a =
 5 3 3
 4 1 3
 5 2 5

--> b = inv(a)

b =
    0.0909    0.8182   -0.5455
    0.4545   -0.9091    0.2727
   -0.2727   -0.4545    0.6364

--> a*b

ans =
    1.0000    0.0000   -0.0000
    0.0000    1.0000         0
    0.0000    0.0000    1.0000

--> b*a

ans =
    1.0000    0.0000         0
    0.0000    1.0000         0
    0.0000   -0.0000    1.0000
```

18.9 LU LU Decomposition for Matrices

18.9.1 Usage

Computes the LU decomposition for a matrix. The form of the command depends on the type of the argument. For full (non-sparse) matrices, the primary form for `lu` is

```
[L,U,P] = lu(A),
```

where `L` is lower triangular, `U` is upper triangular, and `P` is a permutation matrix such that `L*U = P*A`. The second form is

```
[V,U] = lu(A),
```

where V is P'*L (a row-permuted lower triangular matrix), and U is upper triangular. For sparse, square matrices, the LU decomposition has the following form:

```
[L,U,P,Q,R] = lu(A),
```

where A is a sparse matrix of either double or dcomplex type. The matrices are such that L*U=P*R*A*Q, where L is a lower triangular matrix, U is upper triangular, P and Q are permutation vectors and R is a diagonal matrix of row scaling factors. The decomposition is computed using UMFPACK for sparse matrices, and LAPACK for dense matrices.

18.9.2 Example

First, we compute the LU decomposition of a dense matrix.

```
--> a = float([1,2,3;4,5,8;10,12,3])

a =
  1  2  3
  4  5  8
 10 12  3

--> [l,u,p] = lu(a)
l =
    1.0000         0         0
    0.1000    1.0000         0
    0.4000    0.2500    1.0000

u =
   10.0000   12.0000    3.0000
         0    0.8000    2.7000
         0         0    6.1250

p =
 0 0 1
 1 0 0
 0 1 0

--> l*u

ans =
 10 12  3
  1  2  3
  4  5  8

--> p*a

ans =
 10 12  3
  1  2  3
  4  5  8
```

Now we repeat the exercise with a sparse matrix, and demonstrate the use of the permutation vectors.

```
--> a = sparse([1,0,0,4;3,2,0,0;0,0,0,1;4,3,2,4])

a =
```

```
 1 1 1
 2 1 3
 4 1 4
 2 2 2
 4 2 3
 4 3 2
 1 4 4
 3 4 1
 4 4 4
--> [l,u,p,q,r] = lu(a)
l =
 1 1 1
 2 2 1
 3 3 1
 4 4 1
u =
 1 1 0.153846
 1 2 0.230769
 2 2 0.4
 1 3 0.307692
 2 3 0.6
 3 3 0.2
 1 4 0.307692
 3 4 0.8
 4 4 1
p =
 4
 2
 1
 3

q =
 3
 2
 1
 4

r =
 1 1 0.2
 2 2 0.2
 3 3 1
 4 4 0.0769231
--> full(l*a)

ans =
 1 0 0 4
 3 2 0 0
 0 0 0 1
 4 3 2 4

--> b = r*a

b =
 1 1 0.2
```

```
 2 1 0.6
 3 1 0
 4 1 0.307692
 1 2 0
 2 2 0.4
 3 2 0
 4 2 0.230769
 1 3 0
 2 3 0
 3 3 0
 4 3 0.153846
 1 4 0.8
 2 4 0
 3 4 1
 4 4 0.307692
--> full(b(p,q))

ans =
    0.1538    0.2308    0.3077    0.3077
         0    0.4000    0.6000         0
         0         0    0.2000    0.8000
         0         0         0    1.0000
```

18.10 QR QR Decomposition of a Matrix

18.10.1 Usage

Computes the QR factorization of a matrix. The `qr` function has multiple forms, with and without pivoting. The non-pivot version has two forms, a compact version and a full-blown decomposition version. The compact version of the decomposition of a matrix of size `M x N` is

```
[q,r] = qr(a,0)
```

where `q` is a matrix of size `M x L` and `r` is a matrix of size `L x N` and `L = min(N,M)`, and `q*r = a`. The QR decomposition is such that the columns of `Q` are orthonormal, and `R` is upper triangular. The decomposition is computed using the LAPACK routine `xgeqrf`, where `x` is the precision of the matrix. FreeMat supports decompositions of `single` and `double` types.

The second form of the non-pivot decomposition omits the second `0` argument:

```
[q,r] = qr(a)
```

This second form differs from the previous form only for matrices with more rows than columns (`M > N`). For these matrices, the full decomposition is of a matrix `Q` of size `M x M` and a matrix `R` of size `M x N`. The full decomposition is computed using the same LAPACK routines as the compact decomposition, but on an augmented matrix `[a 0]`, where enough columns are added to form a square matrix.

Generally, the QR decomposition will not return a matrix `R` with diagonal elements in any specific order. The remaining two forms of the `qr` command utilize permutations of the columns of `a` so that the diagonal elements of `r` are in decreasing magnitude. To trigger this form of the decomposition, a third argument is required, which records the permutation applied to the argument `a`. The compact version is

```
[q,r,e] = qr(a,0)
```

where `e` is an integer vector that describes the permutation of the columns of `a` necessary to reorder the diagonal elements of `r`. This result is computed using the LAPACK routines `(s,d)geqp3`. In the non-compact version of the QR decomposition with pivoting,

```
[q,r,e] = qr(a)
```

the returned matrix `e` is a permutation matrix, such that `q*r*e' = a`.

18.11 SVD Singular Value Decomposition of a Matrix

18.11.1 Usage

Computes the singular value decomposition (SVD) of a matrix. The `svd` function has three forms. The first returns only the singular values of the matrix:

```
s = svd(A)
```

The second form returns both the singular values in a diagonal matrix `S`, as well as the left and right eigenvectors.

```
[U,S,V] = svd(A)
```

The third form returns a more compact decomposition, with the left and right singular vectors corresponding to zero singular values being eliminated. The syntax is

```
[U,S,V] = svd(A,0)
```

18.11.2 Function Internals

Recall that `sigma\_i` is a singular value of an `M x N` matrix `A` if there exists two vectors `u\_i, v\_i` where `u\_i` is of length `M`, and `v\_i` is of length `u\_i` and

$$Av_i = \sigma_i u_i$$

and generally

$$A = \sum_{i=1}^{K} \sigma_i u_i * v_i',$$

where `K` is the rank of `A`. In matrix form, the left singular vectors `u\_i` are stored in the matrix `U` as

$$U = [u_1, \ldots, u_m], V = [v_1, \ldots, v_n]$$

The matrix `S` is then of size `M x N` with the singular values along the diagonal. The SVD is computed using the `LAPACK` class of functions `GESVD` (Note that this has changed. Previous versions of FreeMat used `GESDD`, which yields a valid, but slightly different choice of the decomposition. Starting in version 4, it was changed to `GESVD` to improve compatibility.

18.11.3 Examples

Here is an example of a partial and complete singular value decomposition.

```
--> A = float(randn(2,3))

A =
    0.1962   -1.7828   -1.0621
   -0.6022   -0.6335    0.5810

--> [U,S,V] = svd(A)
U =
   -0.9929   -0.1189
   -0.1189    0.9929

S =
    2.0957         0         0
         0    1.0268         0
```

```
V =
   -0.0588   -0.6051    0.7940
    0.8806   -0.4061   -0.2443
    0.4702    0.6848    0.5567

--> U*S*V'

ans =
    0.1962   -1.7828   -1.0621
   -0.6022   -0.6335    0.5810

--> svd(A)

ans =
    2.0957
    1.0268
```

Chapter 19

Signal Processing Functions

19.1 CONV Convolution Function

19.1.1 Usage

The `conv` function performs a one-dimensional convolution of two vector arguments. The syntax for its use is

```
z = conv(x,y)
```

where `x` and `y` are vectors. The output is of length `nx + ny -1`. The `conv` function calls `conv2` to do the calculation. See its help for more details.

19.2 CONV2 Matrix Convolution

19.2.1 Usage

The `conv2` function performs a two-dimensional convolution of matrix arguments. The syntax for its use is

```
Z = conv2(X,Y)
```

which performs the full 2-D convolution of `X` and `Y`. If the input matrices are of size `[xm,xn]` and `[ym,yn]` respectively, then the output is of size `[xm+ym-1,xn+yn-1]`. Another form is

```
Z = conv2(hcol,hrow,X)
```

where `hcol` and `hrow` are vectors. In this form, `conv2` first convolves `Y` along the columns with `hcol`, and then convolves `Y` along the rows with `hrow`. This is equivalent to `conv2(hcol(:)*hrow(:)',Y)`.

You can also provide an optional `shape` argument to `conv2` via either

```
Z = conv2(X,Y,'shape')
Z = conv2(hcol,hrow,X,'shape')
```

where `shape` is one of the following strings

- `'full'` - compute the full convolution result - this is the default if no `shape` argument is provided.
- `'same'` - returns the central part of the result that is the same size as `X`.
- `'valid'` - returns the portion of the convolution that is computed without the zero-padded edges. In this situation, `Z` has size `[xm-ym+1,xn-yn+1]` when `xm>=ym` and `xn>=yn`. Otherwise `conv2` returns an empty matrix.

19.2.2 Function Internals

The convolution is computed explicitly using the definition:

$$Z(m,n) = \sum_k \sum_j X(k,j)Y(m-k,n-j)$$

If the full output is requested, then `m` ranges over `0 <= m < xm+ym-1` and `n` ranges over `0 <= n < xn+yn-1`.
For the case where `shape` is `'same'`, the output ranges over `(ym-1)/2 <= m < xm + (ym-1)/2` and `(yn-1)/2 <= n < xn + (`

Chapter 20

Numerical Methods

20.1 CUMTRAPZ Trapezoidal Rule Cumulative Integration

20.1.1 Usage

The cumtrapz routine has the following syntax @[[z] = cumtrapz(x,y) @] where —x— is a dependent vector and —y— an m-by-n matrix equal in at least one dimension to x. (e.g.:x = time samples, y = f(t))

Alternatively, you can enter

```
[z] = cumtrapz(y)
```

for a unit integration of —y—.

If —y— is a matrix, m must be equal to length(x) (e.g.: y must have the same number of rows as x has elements). In this case, integrals are taken for each row, returned in a resulting vector z of dimension (1,n)

20.2 ODE45 Numerical Solution of ODEs

20.2.1 Usage

function [t,y] = ode45(f,tspan,y0,options,varargin) function SOL = ode45(f,tspan,y0,options,varargin)

ode45 is a solver for ordinary differential equations and initial value problems. To solve the ODE

```
y'(t) =  f(t,y)
y(0)  =  y0
```

over the interval tspan=[t0 t1], you can use ode45. For example, to solve the ode

y' = y y(0) = 1

whose exact solution is y(t)=exp(t), over the interval t0=0, t1=3, do

```
-->       [t,y]=ode45(@(t,y) y,[0 3],1)
t =

 Columns 1 to 12

         0    0.0030    0.0180    0.0930    0.3930    0.6930    0.9930    1.2930    1.5930    1.8930

 Columns 13 to 14

    2.7930    3.0000

y =
    1.0000
```

```
    1.0030
    1.0182
    1.0975
    1.4814
    1.9997
    2.6993
    3.6437
    4.9185
    6.6392
    8.9620
   12.0975
   16.3299
   20.0854
```

If you want a dense output (i.e., an output that also contains an interpolating spline), use instead

```
-->       SOL=ode45(@(t,y) y,[0 3],1)

SOL =
    x: 1x14 double array
    y: 1x14 double array
    xe:
    ye:
    ie:
    solver: generic_ode_solver
    interpolant: 1x1 functionpointer array
    idata: 1x1 struct array
```

You can view the result using

```
      plot(0:0.01:3,deval(SOL,0:0.01:3))
```

You will notice that this function is available for "every" value of t, while
plot(t,y,'o-')
is only available at a few points.

The optional argument 'options' is a structure. It may contain any of the following fields:

'AbsTol' - Absolute tolerance, default is 1e-6. 'RelTol' - Relative tolerance, default is 1e-3. 'MaxStep' - Maximum step size, default is (tspan(2)-tspan(1))/10 'InitialStep' - Initial step size, default is maxstep/100 'Stepper' - To override the default Fehlberg integrator 'Events' - To provide an event function 'Projection' - To provide a projection function

The varargin is ignored by this function, but is passed to all your callbacks, i.e., f, the event function and the projection function.

==Event Function==

The event function can be used to detect situations where the integrator should stop, possibly because the right-hand-side has changed, because of a collision, etc...

An event function should look like

function [val,isterminal,direction]=event(t,y,...)

The return values are:

val - the value of the event function. isterminal - whether or not this event should cause termination of the integrator. direction - 1=upcrossings only matter, -1=downcrossings only, 0=both.

== Projection function ==

For geometric integration, you can provide a projection function which will be called after each time step. The projection function has the following signature:

function yn=project(t,yn,...);

If the output yn is very different from the input yn, the quality of interpolation may decrease.

20.3 TRAPZ Trapezoidal Rule Integration

20.3.1 Usage

The trapz routine has the following syntax @[z = trapz(x,y) @] where —x— is a dependent vector and —y— an m-by-n matrix equal in at least one dimension to x. (e.g.:—x— = time samples, y = f(t))

Alternatively, you can enter

```
z = trapz(y)
```

for a unit integration of y.

If y is a matrix, m must be equal to length(x) (e.g.: y must have the same number of rows as x has elements). In this case, integrals are taken for each row, returned in a resulting vector z of dimension (1,n)

Chapter 21

Operating System Functions

21.1 CD Change Working Directory Function

21.1.1 Usage

Changes the current working directory to the one specified as the argument. The general syntax for its use is

```
cd('dirname')
```

but this can also be expressed as

```
cd 'dirname'
```

or

```
cd dirname
```

Examples of all three usages are given below. Generally speaking, `dirname` is any string that would be accepted by the underlying OS as a valid directory name. For example, on most systems, `'.'` refers to the current directory, and `'..'` refers to the parent directory. Also, depending on the OS, it may be necessary to "escape" the directory seperators. In particular, if directories are seperated with the backwards-slash character `'\\'`, then the path specification must use double-slashes `'\\\\'`. Note: to get file-name completion to work at this time, you must use one of the first two forms of the command.

21.1.2 Example

The `pwd` command returns the current directory location. First, we use the simplest form of the `cd` command, in which the directory name argument is given unquoted.

```
--> pwd

ans =
/home/sbasu/Devel/FreeMat/help/tmp
--> cd ..
--> pwd

ans =
/home/sbasu/Devel/FreeMat/help
```

Next, we use the "traditional" form of the function call, using both the parenthesis and a variable to store the quoted string.

```
--> a = pwd;
--> cd(a)
--> pwd

ans =
/home/sbasu/Devel/FreeMat/help/tmp
```

21.2 COPYFILE Copy Files

21.2.1 Usage

Copies a file or files from one location to another. There are several syntaxes for this function that are acceptable:

```
copyfile(file_in,file_out)
```

copies the file from `file\_in` to `file\_out`. Also, the second argument can be a directory name:

```
copyfile(file_in,directory_out)
```

in which case `file\_in` is copied into the directory specified by `directory\_out`. You can also use `copyfile` to copy entire directories as in

```
copyfile(dir_in,dir_out)
```

in which case the directory contents are copied to the destination directory (which is created if necessary). Finally, the first argument to `copyfile` can contain wildcards

```
copyfile(pattern,directory_out)
```

in which case all files that match the given pattern are copied to the output directory. Note that to remain compatible with the MATLAB API, this function will delete/replace destination files that already exist, unless they are marked as read-only. If you want to force the copy to succeed, you can append a `'f'` argument to the `copyfile` function:

```
copyfile(arg1,arg2,'f')
```

or equivalently

```
copyfile arg1 arg2 f
```

21.3 DELETE Delete a File

21.3.1 Usage

Deletes a file. The general syntax for its use is

```
delete('filename')
```

or alternately

```
delete filename
```

which removes the file described by `filename` which must be relative to the current path.

21.4 DIR List Files Function

21.4.1 Usage

In some versions of FreeMat (pre 3.1), the `dir` function was aliased to the `ls` function. Starting with version `3.1`, the `dir` function has been rewritten to provide compatibility with MATLAB. The general syntax for its use is

```
dir
```

in which case, a listing of the files in the current directory are output to the console. Alternately, you can specify a target via

```
dir('name')
```

or using the string syntax

```
dir name
```

If you want to capture the output of the `dir` command, you can assign the output to an array

```
result = dir('name')
```

(or you can omit `'name'` to get a directory listing of the current directory. The resulting array `result` is a structure array containing the fields:

- `name` the filename as a string
- `date` the modification date and time stamp as a string
- `bytes` the size of the file in bytes as a `uint64`
- `isdir` a logical that is `1` if the file corresponds to a directory.

Note that `'name'` can also contain wildcards (e.g., `dir *.m` to get a listing of all FreeMat scripts in the current directory.

21.5 DIRSEP Director Seperator

21.5.1 Usage

Returns the directory seperator character for the current platform. The general syntax for its use is

```
y = dirsep
```

This function can be used to build up paths (or see `fullfile` for another way to do this.

21.6 FILEATTRIB Get and Set File or Directory Attributes

21.6.1 Usage

Retrieves information about a file or directory. The first version uses the syntax

```
y = fileattrib(filename)
```

where `filename` is the name of a file or directory. The returned structure contains several entries, corresponding to the attributes of the file. Here is a list of the entries, and their meaning:

- `Name` - the full pathname for the file
- `archive` - not used, set to `0`

- `system` - not used, set to `0`
- `hidden` - set to `1` for a hidden file, and `0` else.
- `directory` - set to `1` for a directory, and `0` for a file.
- `UserRead` - set to `1` if the user has read permission, `0` otherwise.
- `UserWrite` - set to `1` if the user has write permission, `0` otherwise.
- `UserExecute` - set to `1` if the user has execute permission, `0` otherwise.
- `GroupRead` - set to `1` if the group has read permission, `0` otherwise.
- `GroupWrite` - set to `1` if the group has write permission, `0` otherwise.
- `GroupExecute` - set to `1` if the group has execute permission, `0` otherwise.
- `OtherRead` - set to `1` if the world has read permission, `0` otherwise.
- `OtherWrite` - set to `1` if the world has write permission, `0` otherwise.
- `OtherExecute` - set to `1` if the world has execute permission, `0` otherwise.

You can also provide a wildcard filename to get the attributes for a set of files e.g.,

```
y = fileattrib('foo*')
```

You can also use `fileattrib` to change the attributes of a file and/or directories. To change attributes, use one of the following syntaxes

```
y = fileattrib(filename,attributelist)
y = fileattrib(filename,attributelist,userlist)
y = fileattrib(filename,attributelist,userlist,'s')
```

where `attributelist` is a string that consists of a list of attributes, each preceeded by a `+` to enable the attribute, and `-` to disable the attribute. The valid list of attributes that can be changed are

- `'w'` - change write permissions
- `'r'` - change read permissions
- `'x'` - change execute permissions

for example, `'-w +r'` would indicate removal of write permissions and addition of read permissions. The `userlist` is a string that lists the realm of the permission changes. If it is not specified, it defaults to `'u'`.

- `'u'` - user or owner permissions
- `'g'` - group permissions
- `'o'` - other permissions ("world" in normal Unix terminology)
- `'a'` - equivalent to 'ugo'.

Finally, if you specify a `'s'` for the last argument, the attribute change is applied recursively, so that setting the attributes for a directory will apply to all the entries within the directory.

21.7 FILEPARTS Extract Filename Parts

21.7.1 Usage

The `fileparts` takes a filename, and returns the path, filename, extension, and (for MATLAB-compatibility) an empty version number of the file. The syntax for its use is

```
[path,name,extension,version] = fileparts(filename)
```

where `filename` is a string containing the description of the file, and `path` is the `path` to the file,

21.8 FULLFILE Build a Full Filename From Pieces

21.8.1 Usage

The `fullfile` routine constructs a full filename from a set of pieces, namely, directory names and a filename. The syntax is:

```
x = fullfile(dir1,dir2,...,dirn,filename)
```

where each of the arguments are strings. The `fullfile` function is equivalent to `[dir1 dirsep dir2 dirsep ... dirn dirs`

21.8.2 Example

```
--> fullfile('path','to','my','file.m')

ans =
path/to/my/file.m
```

21.9 GETENV Get the Value of an Environment Variable

21.9.1 Usage

The `getenv` function returns the value for an environment variable from the underlying OS. The syntax for the `getenv` function is

```
y = getenv(environment_variable)
```

where `environment\_variable` is the name of the environment variable to return. The return is a string.

21.9.2 Example

Here is an example of using the `getenv` function to get the value for the `HOME` variable

```
--> getenv('HOME')

ans =
/home/sbasu
```

21.10 GETPATH Get Current Search Path

21.10.1 Usage

Returns a `string` containing the current FreeMat search path. The general syntax for its use is

```
y = getpath
```

The delimiter between the paths depends on the system being used. For Win32, the delimiter is a semicolon. For all other systems, the delimiter is a colon.

21.10.2 Example

The `getpath` function is straightforward.

```
--> getpath

ans =
/usr/local/FreeMat/MFiles:/localhome/basu/MFiles
```

21.11 LS List Files Function

21.11.1 Usage

Lists the files in a directory or directories. The general syntax for its use is

```
ls('dirname1','dirname2',...,'dirnameN')
```

but this can also be expressed as

```
ls 'dirname1' 'dirname2' ... 'dirnameN'
```

or

```
ls dirname1 dirname2 ... dirnameN
```

For compatibility with some environments, the function `dir` can also be used instead of `ls`. Generally speaking, `dirname` is any string that would be accepted by the underlying OS as a valid directory name. For example, on most systems, `'.'` refers to the current directory, and `'..'` refers to the parent directory. Also, depending on the OS, it may be necessary to "escape" the directory seperators. In particular, if directories are seperated with the backwards-slash character `'\\'`, then the path specification must use double-slashes `'\\\\'`. Two points worth mentioning about the `ls` function:

- To get file-name completion to work at this time, you must use one of the first two forms of the command.
- If you want to capture the output of the `ls` command, use the `system` function instead.

21.11.2 Example

First, we use the simplest form of the `ls` command, in which the directory name argument is given unquoted.

```
--> ls m*.m
```

Next, we use the "traditional" form of the function call, using both the parenthesis and the quoted string.

```
--> ls('m*.m')
```

In the third version, we use only the quoted string argument without parenthesis.

```
--> ls 'm*.m'
```

21.12 MKDIR Make Directory

21.12.1 Usage

Creates a directory. The general syntax for its use is

```
mkdir('dirname')
```

which creates the directory `dirname` if it does not exist. The argument `dirname` can be either a relative path or an absolute path. For compatibility with MATLAB, the following syntax is also allowed

```
mkdir('parentdir','dirname')
```

which attempts to create a directory `dirname` in the directory given by `parentdir`. However, this simply calls `mkdir([parentdir dirsep dirname])`, and if this is not the required behavior, please file an enhancement request to have it changed. Note that `mkdir` returns a logical `1` if the call succeeded, and a logical `0` if not.

21.13 PWD Print Working Directory Function

21.13.1 Usage

Returns a `string` describing the current working directory. The general syntax for its use is

```
y = pwd
```

21.13.2 Example

The `pwd` function is fairly straightforward.

```
--> pwd

ans =
/home/sbasu/Devel/FreeMat/help/tmp
```

21.14 RMDIR Remove Directory

21.14.1 Usage

Deletes a directory. The general syntax for its use is

```
rmdir('dirname')
```

which removes the directory `dirname` if it is empty. If you want to delete the directory and all subdirectories and files in it, use the syntax

```
rmdir('dirname','s')
```

21.15 SETPATH Set Current Search Path

21.15.1 Usage

Changes the current FreeMat search path. The general syntax for its use is

```
setpath(y)
```

where `y` is a `string` containing a delimited list of directories to be searched for M files and libraries. The delimiter between the paths depends on the system being used. For Win32, the delimiter is a semicolon. For all other systems, the delimiter is a colon.

@Example The `setpath` function is straightforward.

```
--> getpath

ans =
/usr/local/FreeMat/MFiles:/localhome/basu/MFiles
--> setpath('/usr/local/FreeMat/MFiles:/localhome/basu/MFiles')
--> getpath

ans =
/usr/local/FreeMat/MFiles:/localhome/basu/MFiles
```

21.16 SYSTEM Call an External Program

21.16.1 Usage

The `system` function allows you to call an external program from within FreeMat, and capture the output. The syntax of the `system` function is

```
y = system(cmd)
```

where `cmd` is the command to execute. The return array `y` is of type `cell-array`, where each entry in the array corresponds to a line from the output.

21.16.2 Example

Here is an example of calling the `ls` function (the list files function under Un*x-like operating system).

```
--> y = system('ls a*.m')

y =
 [addtest2.m] [addtest3.m] [addtest.m]

--> y{1}

ans =
addtest2.m
```

Chapter 22

Optimization and Curve Fitting

22.1 FITFUN Fit a Function

22.1.1 Usage

Fits `n` (non-linear) functions of `m` variables using least squares and the Levenberg-Marquardt algorithm. The general syntax for its usage is

```
[xopt,yopt] = fitfun(fcn,xinit,y,weights,tol,params...)
```

Where `fcn` is the name of the function to be fit, `xinit` is the initial guess for the solution (required), `y` is the right hand side, i.e., the vector `y` such that:

$$xopt = \arg\min_x \|\mathrm{diag}(weights) * (f(x) - y)\|_2^2,$$

the output `yopt` is the function `fcn` evaluated at `xopt`. The vector `weights` must be the same size as `y`, and contains the relative weight to assign to an error in each output value. Generally, the ith weight should reflect your confidence in the ith measurement. The parameter `tol` is the tolerance used for convergence. The function `fcn` must return a vector of the same size as `y`, and `params` are passed to `fcn` after the argument `x`, i.e.,

$$y = fcn(x, param1, param2, ...).$$

Note that both `x` and `y` (and the output of the function) must all be real variables. Complex variables are not handled yet.

22.2 GAUSFIT Gaussian Curve Fit

22.2.1 Usage

The `gausfit` routine has the following syntax

```
[mu,sigma,dc,gain,yhat] = gausfit(t,y,w,mug,sigmag,dcg,gaing).
```

where the required inputs are

- `t` - the values of the independant variable (e.g., time samples)
- `y` - the values of the dependant variable (e.g., f(t))

The following inputs are all optional, and default values are available for each of them.

- `w` - the weights to use in the fitting (set to ones if omitted)
- `mug` - initial estimate of the mean

- `sigmag` - initial estimate of the sigma (standard deviation)
- `dcg` - initial estimate of the DC value
- `gaing` - initial estimate of the gain

The fit is of the form `yhat=gain*exp((t-mu).\^2/(2*sigma\^2))+dc`. The outputs are

- `mu` - the mean of the fit
- `sigma` - the sigma of the fit
- `dc` - the dc term of the fit
- `gain` - the gain of the gaussian fit
- `yhat` - the output samples (the Gaussian fits)

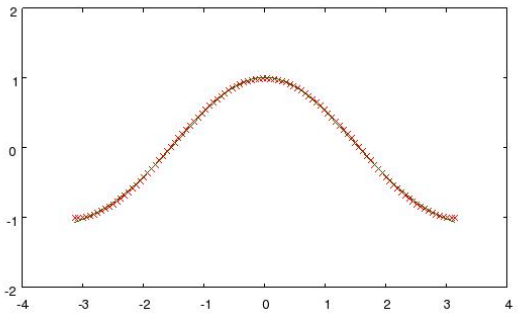
Because the fit is nonlinear, a good initial guess is critical to convergence of the solution. Thus, you can supply initial guesses for each of the parameters using the `mug`, `sigmag`, `dcg`, `gaing` arguments. Any arguments not supplied are estimated using a simple algorithm. In particular, the DC value is estimated by taking the minimum value from the vector `y`. The gain is estimated from the range of `y`. The mean and standard deviation are estimated using the first and second order moments of `y`. This function uses `fitfun`.

22.2.2 Example

Suppose we want to fit a cycle of a cosine using a Gaussian shape.

```
--> t = linspace(-pi,pi);
--> y = cos(t);
--> [mu,sigma,dc,gain,yhat] = gausfit(t,y);
--> plot(t,y,'rx',t,yhat,'g-');
```

Which results in the following plot



22.3 INTERP2 2-D Interpolation

22.3.1 Usage

Given a set of monotonically increasing `x` coordinates and a corresponding set of `y` values, performs simple linear interpolation to a new set of `x` coordinates. The general syntax for its usage is

```
zi = interp2(z,xi,yi)
```

where `xi` and `yi` are vectors of the same length. The output vector `zi` is the same size as the input vector `xi`. For each element of `xi`, the values in `zi` are linearly interpolated by default. Interpolation method can be selected as:

```
zi = interp2(z,xi,yi,method)
```

Default interpolation method is `'linear'`. Other methods are `'nearest'`, and `'cubic'`. For values in `xi, yi` that are outside the size of `z`, the default value returned is NaN. To change this behavior, you can specify the extrapolation value:

```
zi = interp2(z,xi,yi,method,extrapval)
```

The `z` and `xi,yi` vectors must be real, although complex types are allowed for `z`.

22.4 INTERPLIN1 Linear 1-D Interpolation

22.4.1 Usage

Given a set of monotonically increasing `x` coordinates and a corresponding set of `y` values, performs simple linear interpolation to a new set of `x` coordinates. The general syntax for its usage is

```
yi = interplin1(x1,y1,xi)
```

where `x1` and `y1` are vectors of the same length, and the entries in `x1` are monotoniccally increasing. The output vector `yi` is the same size as the input vector `xi`. For each element of `xi`, the values in `y1` are linearly interpolated. For values in `xi` that are outside the range of `x1` the default value returned is `nan`. To change this behavior, you can specify the extrapolation flag:

```
yi = interplin1(x1,y1,xi,extrapflag)
```

Valid options for `extrapflag` are:

- `'nan'` - extrapolated values are tagged with `nan`s
- `'zero'` - extrapolated values are set to zero
- `'endpoint'` - extrapolated values are set to the endpoint values
- `'extrap'` - linear extrapolation is performed

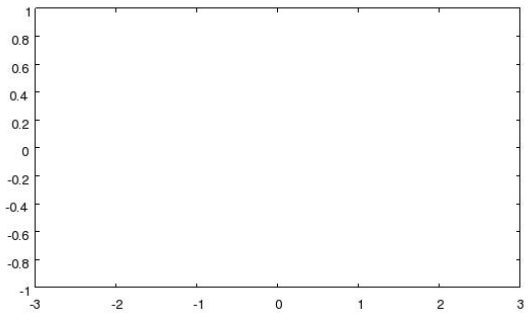
The `x1` and `xi` vectors must be real, although complex types are allowed for `y1`.

22.4.2 Example

Here is an example of simple linear interpolation with the different extrapolation modes. We start with a fairly coarse sampling of a cosine.

```
--> x = linspace(-pi*7/8,pi*7/8,15);
--> y = cos(x);
--> plot(x,y,'ro');
```

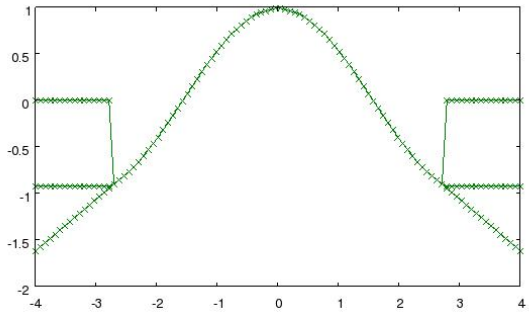
which is shown here



Next, we generate a finer sampling over a slightly broader range (in this case `[-pi,pi]`). First, we demonstrate the `'nan'` extrapolation method

```
--> xi = linspace(-4,4,100);
--> yi_nan = interplin1(x,y,xi,'nan');
--> yi_zero = interplin1(x,y,xi,'zero');
--> yi_endpoint = interplin1(x,y,xi,'endpoint');
--> yi_extrap = interplin1(x,y,xi,'extrap');
--> plot(x,y,'ro',xi,yi_nan,'g-x',xi,yi_zero,'g-x',xi,yi_endpoint,'g-x',xi,yi_extrap,'g-x');
```

which is shown here



22.5 POLY Convert Roots To Polynomial Coefficients

22.5.1 Usage

This function calculates the polynomial coefficients for given roots

```
p = poly(r)
```

when `r` is a vector, is a vector whose elements are the coefficients of the polynomial whose roots are the elements of `r`. Alternately, you can provide a matrix

```
p = poly(A)
```

when `A` is an `N x N` square matrix, is a row vector with `N+1` elements which are the coefficients of the characteristic polynomial, `det(lambda*eye(size(A))-A)`.

Contributed by Paulo Xavier Candeias under GPL.

22.5.2 Example

Here are some examples of the use of `poly`

```
--> A = [1,2,3;4,5,6;7,8,0]

A =
 1 2 3
 4 5 6
 7 8 0

--> p = poly(A)

p =
    1.0000   -6.0000  -72.0000  -27.0000

--> r = roots(p)

r =
```

```
   12.1229
   -5.7345
   -0.3884
```

22.6 POLYDER Polynomial Coefficient Differentiation

22.6.1 Usage

The `polyder` function returns the polynomial coefficients resulting from differentiation of polynomial `p`. The syntax for its use is either

```
 pder = polyder(p)
```

for the derivitave of polynomial p, or

```
 convp1p2der = polyder(p1,p2)
```

for the derivitave of polynomial conv(p1,p2), or still

```
 [nder,dder] = polyder(n,d)
```

for the derivative of polynomial `n/d` (`nder` is the numerator and `dder` is the denominator). In all cases the polynomial coefficients are assumed to be in decreasing degree. Contributed by Paulo Xavier Candeias under GPL

22.6.2 Example

Here are some examples of the use of `polyder`

```
--> polyder([2,3,4])

ans =
 4 3

--> polyder([2,3,4],7)

ans =
 28 21

--> [n,d] = polyder([2,3,4],5)
n =
 -20 -15

d =
 25
```

22.7 POLYFIT Fit Polynomial To Data

22.7.1 Usage

The `polyfit` routine has the following syntax

```
  p = polyfit(x,y,n)
```

where `x` and `y` are vectors of the same size, and `n` is the degree of the approximating polynomial. The resulting vector `p` forms the coefficients of the optimal polynomial (in descending degree) that fit `y` with `x`.

22.7.2 Function Internals

The `polyfit` routine finds the approximating polynomial

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

such that

$$\sum_i (p(x_i) - y_i)^2$$

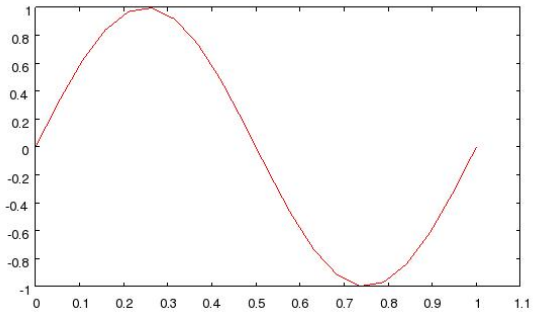
is minimized. It does so by forming the Vandermonde matrix and solving the resulting set of equations using the backslash operator. Note that the Vandermonde matrix can become poorly conditioned with large `n` quite rapidly.

22.7.3 Example

A classic example from Edwards and Penny, consider the problem of approximating a sinusoid with a polynomial. We start with a vector of points evenly spaced on the unit interval, along with a vector of the sine of these points.

```
--> x = linspace(0,1,20);
--> y = sin(2*pi*x);
--> plot(x,y,'r-')
```

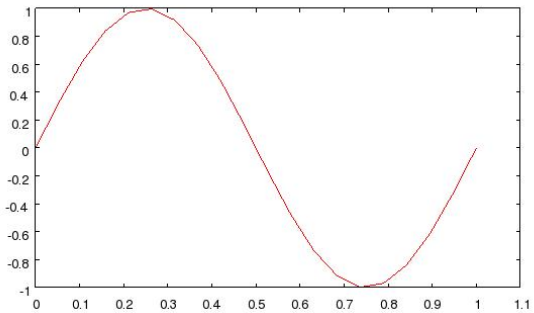
The resulting plot is shown here



Next, we fit a third degree polynomial to the sine, and use `polyval` to plot it

```
--> p = polyfit(x,y,3)

p =
   21.9170  -32.8756   11.1897   -0.1156

--> f = polyval(p,x);
--> plot(x,y,'r-',x,f,'ko');
```

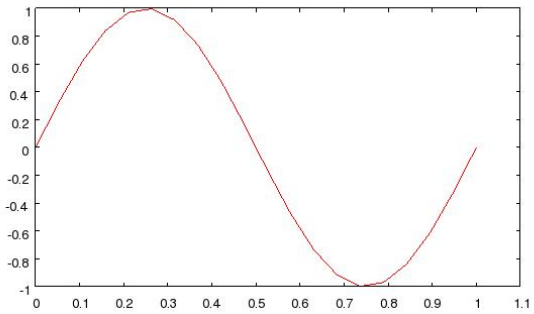
The resulting plot is shown here



Increasing the order improves the fit, as

```
--> p = polyfit(x,y,11)

p =
   12.4644  -68.5541  130.0555  -71.0940  -38.2814  -14.1222   85.1018   -0.5642  -41.2861   -0.0029

--> f = polyval(p,x);
--> plot(x,y,'r-',x,f,'ko');
```

The resulting plot is shown here



22.8 POLYINT Polynomial Coefficient Integration

22.8.1 Usage

The polyint function returns the polynomial coefficients resulting from integration of polynomial p. The syntax for its use is either

```
 pint = polyint(p,k)
```

or, for a default `k = 0`,

```
 pint = polyint(p);
```

where `p` is a vector of polynomial coefficients assumed to be in decreasing degree and `k` is the integration constant. Contributed by Paulo Xavier Candeias under GPL

22.8.2 Example

Here is are some examples of the use of `polyint`.

```
--> polyint([2,3,4])

ans =
    0.6667    1.5000    4.0000         0
```

And

```
--> polyint([2,3,4],5)

ans =
    0.6667    1.5000    4.0000    5.0000
```

22.9 POLYVAL Evaluate Polynomial Fit at Selected Points

22.9.1 Usage

The `polyval` routine has the following syntax

```
y = polyval(p,x)
```

where `p` is a vector of polynomial coefficients, in decreasing degree (as generated by `polyfit`, for example). If `x` is a matrix, the polynomial is evaluated in the matrix sense (in which case `x` must be square).

22.9.2 Function Internals

The polynomial is evaluated using a recursion method. If the polynomial is

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

then the calculation is performed as

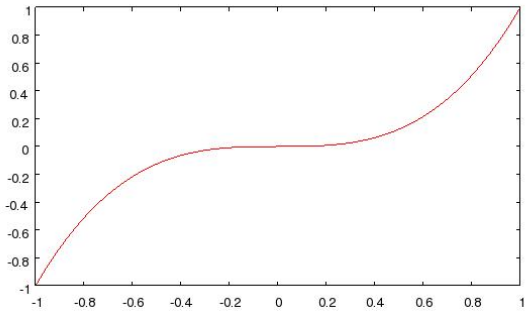
$$p(x) = ((p_1)x + p_2)x + p_3$$

22.9.3 Example

Here is a plot of `x\^3` generated using polyval

```
--> p = [1 0 0 0]

p =
 1 0 0 0

--> x = linspace(-1,1);
--> y = polyval(p,x);
--> plot(x,y,'r-')
```

Here is the resulting plot



22.10 ROOTS Find Roots of Polynomial

22.10.1 Usage

The `roots` routine will return a column vector containing the roots of a polynomial. The general syntax is

```
z = roots(p)
```

where `p` is a vector containing the coefficients of the polynomial ordered in descending powers.

22.10.2 Function Internals

Given a vector

$$[p_1, p_2, \ldots p_n]$$

which describes a polynomial

$$p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_n$$

we construct the companion matrix (which has a characteristic polynomial matching the polynomial described by `p`), and then find the eigenvalues of it (which are the roots of its characteristic polynomial), and which are also the roots of the polynomial of interest. This technique for finding the roots is described in the help page for `roots` on the Mathworks website.

22.10.3 Example

Here is an example of finding the roots to the polynomial

$$x^3 - 6x^2 - 72x - 27$$

```
--> roots([1 -6 -72 -27])

ans =
   12.1229
   -5.7345
   -0.3884
```

Chapter 23

Handle-Based Graphics

23.1 AXES Create Handle Axes

23.1.1 Usage

This function has three different syntaxes. The first takes no arguments,

```
h = axes
```

and creates a new set of axes that are parented to the current figure (see `gcf`). The newly created axes are made the current axes (see `gca`) and are added to the end of the list of children for the current figure. The second form takes a set of property names and values

```
h = axes(propertyname,value,propertyname,value,...)
```

Creates a new set of axes, and then sets the specified properties to the given value. This is a shortcut for calling `set(h,propertyname,value)` for each pair. The third form takes a handle as an argument

```
axes(handle)
```

and makes `handle` the current axes, placing it at the head of the list of children for the current figure.

23.2 AXIS Setup Axis Behavior

23.2.1 Usage

Control the axis behavior. There are several versions of the axis command based on what you would like the axis command to do. The first versions set scalings for the current plot. The general syntax for its use is

```
axis([xmin xmax ymin ymax zmin zmax cmin cmax])
```

which sets the limits in the X, Y, Z and color axes. You can also set only the X, Y and Z axes:

```
axis([xmin xmax ymin ymax zmin zmax])
```

or only the X and Y axes:

```
axis([xmin xmax ymin ymax])
```

To retrieve the current axis limits, use the syntax

```
x = axis
```

where `x` is a 4-vector for 2D plots, and a 6-vector for 3D plots.

There are a number of axis options supported by FreeMat. The first version sets the axis limits to be automatically selected by FreeMat for each dimension. This state is the default one for new axes created by FreeMat.

```
axis auto
```

The next option sets all of the axis limits to `manual` mode. This state turns off automatic scaling of the axis based on the children of the current axis object.

```
axis manual
```

The next option sets the axis limits to fit tightly around the data.

```
axis tight
```

The next option adjusts the axis limits and plotbox aspect ratio so that the axis fills the position rectangle.

```
axis fill
```

The next option puts the axis in matrix mode. This mode is equivalent to the standard mode, but with the vertical axis reversed. Thus, the origin of the coordinate system is at the top left corner of the plot. This mode makes plots of matrix elements look normal (i.e., an identity matrix goes from upper left to lower right).

```
axis ij
```

The next option puts the axis in normal mode, with the origin at the lower left corner.

```
axis xy
```

The next option sets the axis parameters (specifically the data aspect ratio) so that equal ticks on each axis represent equal length. In this mode, spheres look spherical insteal of ellipsoidal.

```
axis equal
```

The next option is the same as `axis equal`, but sets the plot box to fit tightly around the data (so no background shows through). It is the best option to use when displaying images.

```
axis image
```

The next option makes the axis box square.

```
axis square
```

The next option restores many of the normal characteristics of the axis. In particular, it undoes the effects of `square` `image` and `equal` modes.

```
axis normal
```

The next mode freezes axis properties so that 3D objects can be rotated properly.

```
axis vis3d
```

The next mode turns off all labels, tick marks and background.

```
axis on
```

The next mode turns on all labels, tick marks and background.

```
axis off
```

The next mode is similar to `axis off`, but also repacks the figure as tightly as possible. The result is a plot box that takes up the entire `outerposition` vector.

```
axis maximal
```

The `axis` command can also be applied to a particular axis (as opposed to the current axis as returned by `gca`) handle

```
axis(M,...)
```

23.3 AXISPROPERTIES Axis Object Properties

23.3.1 Usage

Below is a summary of the properties for the axis.

- `activepositionproperty` - `four vector` - Not used.
- `alim` - `two vector` - Controls the mapping of transparency. The vector `[a\_1,a\_2]@` defines the scale for transparency. Plots then map `a\_1` to a completely opaque value, and `a\_2` to a completely transparent value. This mapping is applied to the alpha data of the plot data.
- `alimmode` - `{'auto','manual'}` - For `auto` mode, we map the alpha ranges of all objects in the plot to a full scale. For `manual` mode, we use the `alim` vector.
- `ambientlightcolor` - `colorspec` - Not used.
- `box` - `On/Off` - Not used.
- `cameraposition` - `three vector` - Set the position for the camera in axis space.
- `camerapositionmode` - `{'auto','manual'}` - For `manual` mode, the camera position is picked up from the `cameraposition` vector. For `auto` mode, the camera position is set to be centered on the `x` and `y` axis limits, and beyond the `z` maximum limit.
- `cameratarget` - `three vector` - Defines the point in axis space that the camera is targetted at.
- `cameratargetmode` - `{'auto','manual'}` - For `manual` mode the camera target is picked up from the `cameratarget` vector. For `auto` mode, the camera target is chosen to be the center of the three axes.
- `cameraupvector` - `three vector` - Defines the upwards vector for the camera (what is ultimately mapped to the vertical axis of the plot or screen). This vector must not be parallel to the vector that is defined by the optical axis (i.e., the one connecting the target to the camera position).
- `cameraupvectormode` - `{'auto','manual'}` - For `manual` mode, the camera up vector is picked up from the `cameraupvector`. The `auto` mode chooses the up vector to point along the positive `y` axis.
- `cameraviewangle` - `scalar` - Not used.
- `cameraviewanglemode` - `{'auto','manual'}` - Not used.
- `children` - `vector of handles` - A vector containing handles to children of the current axis. Be careful as to how you manipulate this vector. FreeMat uses a reference counting mechanism for graphics objects, so if you remove a handle from the `children` property of an axis, and you have not added it to the `children` property of another object, it will be deleted.
- `clim` - `two vector` - The color range vector. This vector contains two values that dictate how children of this axis get mapped to the colormap. Values between the two endpoints of this vector are mapped to the extremes of the colormap.
- `climmode` - `{'auto','manual'}` - For `auto` mode, the color limits are chosen to span the colordata for all of the children objects. For `manual` mode, the color mapping is based on `clim`.
- `clipping` - `{'on','off'}` - Not used.
- `color` - `colorspec` - The color used to draw the background box for the axes. Defaults to white.
- `colororder` - `color vector` - A vector of color specs (in RGB) that are cycled between when drawing line plots into this axis. The default is order blue,green,red,cyan,magenta,yellow,black.
- `datalimits` - `six vector` - A vector that contains the x, y and z limits of the data for children of the current axis. Changes to this property are ignored - it is calculated by FreeMat based on the datasets.

- `dataaspectratio` - `three vector` - A vector that describes the aspect ratio of the data. You can think of this as the relative scaling of units for each axis. For example, if one unit along the x axis is twice as long as one unit along the y axis, you would specify a data aspect ratio of `[2,1,1]`.
- `dataaspectratiomode` - `{'auto','manual'}` - When the data aspect ratio is set to `manual`, the data is scaled by the data aspect ratio before being plotted. When the data aspect ratio mode is `auto` a complex set of rules are applied to determine how the data should be scaled. If `dataaspectratio` mode is `auto` and `plotboxaspectratio` is `auto`, then the default data aspect ratio is set to `[1,1,1]` and the default plot box aspect ratio is chosen proportional to `[xrange,yrange,zrange]`, where `xrange` is the span of data along the `x` axis, and similarly for `yrange` and `zrange`. If `plotboxaspectratio` is set to `[px,py,pz]`, then the `dataaspectratio` is set to `[xrange/px,yrange/py,zrange/pz]`. If one of the axes has been specified manually, then the data will be scaled to fit the axes as well as possible.
- `fontangle` - `{'normal','italic','oblique'}` - The angle of the fonts used for text labels (e.g., tick labels).
- `fontsize` - `scalar` - The size of fonts used for text labels (tick labels).
- `fontunits` - Not used.
- `fontweight` - `{'normal','bold','light','demi'}` - The weight of the font used for tick labels.
- `gridlinestyle` - `{'-','--',':','-.','none'}` - The line style to use for drawing the grid lines. Defaults to `':'`.
- `handlevisibility` - Not used.
- `hittest` - Not used.
- `interruptible` - Not used.
- `layer` - Not used.
- `linestyleorder` - `linestyle vector` - A vector of linestyles that are cycled through when plotted line series.
- `linewidth` - `scalar` - The width of line used to draw grid lines, axis lines, and other lines.
- `minorgridlinestyle` - `{'-','--',':','-.','none'}` - The line style used for drawing grid lines through minor ticks.
- `nextplot` - `{'add','replace','replacechildren'}` - Controls how the next plot interacts with the axis. If it is set to `'add'` the next plot will be added to the current axis. If it is set to `'replace'` the new plot replaces all of the previous children.
- `outerposition` - `four vector` - Specifies the coordinates of the outermost box that contains the axis relative to the containing figure. This vector is in normalized coordinates and corresponds to the `x, y, width, height` coordinates of the box.
- `parent` - `handle` - The handle for the containing object (a figure).
- `plotboxaspectratio` - `three vector` - Controls the aspect ratio of the plot box. See the entry under `dataaspectratio` for details on how FreeMat uses this vector in combination with the axis limits and the `plotboxaspectratio` to determine how to scale the data.
- `plotboxaspectratiomode` - `{'auto','manual'}` - The plot box aspect ratio mode interacts with the `dataaspectratiomode` and the axis limits.
- `position` - `fourvector` - The normalized coordinates of the plot box space. Should be inside the rectable defined by `outerposition`.

- `positionmode` - `{'auto','manual'}` - the position mode is normally `'auto'` which means that FreeMat computes the position vector to fit the plot inside the `outerposition` vector. If you set the `position` vector manually (using a `set` command), this `mode` flag becomes `'manual'` and remains that way until reset to @—'auto'.
- `projection` - Not used.
- `selected` - Not used.
- `selectionhighlight` - Not used.
- `tag` - A string that can be set to tag the axes with a name.
- `textheight` - `scalar` - This value is set by FreeMat to the height of the current font in pixels.
- `tickdir` - `{'in','out'}` - The direction of ticks. Defaults to `'in'` for 2D plots, and `'out'` for 3D plots if `tickdirmode` is `auto`.
- `tickdirmode` - `{'auto','manual'}` - When set to `'auto'` the `tickdir` defaults to `'in'` for 2D plots, and `'out'` for 3D plots.
- `ticklength` - `two vector` - The first element is the length of the tick in 2D plots, and the second is the length of the tick in the 3D plots. The lengths are described as fractions of the longer dimension (width or height).
- `tightinset` - Not used.
- `title` - `handle` - The handle of the label used to represent the title of the plot.
- `type` - `string` - Takes the value of `'axes'` for objects of the axes type.
- `units` - Not used.
- `userdata` - `array` - An arbitrary array you can set to anything you want.
- `visible` - `{'on','off'}` - If set to `'on'` the axes are drawn as normal. If set to `'off'`, only the children of the axes are drawn. The plot box, axis lines, and tick labels are not drawn.
- `xaxislocation` - `{'top','bottom'}` - Controls placement of the x axis.
- `yaxislocation` - `{'left','right'}` - Controls placement of the y axis.
- `xcolor` - `colorspec` - The color of the x elements including the the x axis line, ticks, grid lines and tick labels
- `ycolor` - `colorspec` - The color of the y elements including the the y axis line, ticks, grid lines and tick labels.
- `zcolor` - `colorspec` - The color of the z elements including the the z axis line, ticks, grid lines and tick labels.
- `xdir` - `{'normal','reverse'}` - For `normal`, axes are drawn as you would expect (e.g, in default 2D mode, the x axis has values increasing from left to right. For `reverse`, the x axis has values increasing from right to left.
- `ydir` - `{'normal','reverse'}` - For `normal`, axes are drawn as you would expect (e.g, in default 2D mode, the y axis has values increasing from bottom to top. For `reverse`, the y axis has values increasing from top to bottom.
- `zdir` - `{'normal','reverse'}` - For `normal`, axes are drawn as you would expect. In default 3D mode, the z axis has values increasing in some direction (usually up). For `reverse` the z axis increases in the opposite direction.

- `xgrid` - `{'on','off'}` - Set to `on` to draw grid lines from ticks on the x axis.
- `ygrid` - `{'on','off'}` - Set to `on` to draw grid lines from ticks on the y axis.
- `zgrid` - `{'on','off'}` - Set to `on` to draw grid lines from ticks on the z axis.
- `xlabel` - `handle` - The handle of the text label attached to the x axis. The position of that label and the rotation angle is computed automatically by FreeMat.
- `ylabel` - `handle` - The handle of the text label attached to the y axis. The position of that label and the rotation angle is computed automatically by FreeMat.
- `zlabel` - `handle` - The handle of the text label attached to the z axis. The position of that label and the rotation angle is computed automatically by FreeMat.
- `xlim` - `two vector` - Contains the limits of the data along the x axis. These are set automatically for `xlimmode`. When manually set it allows you to zoom into the data. The first element of this vector should be the smallest x value you want mapped to the axis, and the second element should be the largest.
- `ylim` - `two vector` - Contains the limits of the data along the y axis. These are set automatically for `ylimmode`. When manually set it allows you to zoom into the data. The first element of this vector should be the smallest y value you want mapped to the axis, and the second element should be the largest.
- `zlim` - `two vector` - Contains the limits of the data along the z axis. These are set automatically for `zlimmode`. When manually set it allows you to zoom into the data. The first element of this vector should be the smallest z value you want mapped to the axis, and the second element should be the largest.
- `xlimmode` - `{'auto','manual'}` - Determines if `xlim` is determined automatically or if it is determined manually. When determined automatically, it is chosen to span the data range (at least).
- `ylimmode` - `{'auto','manual'}` - Determines if `ylim` is determined automatically or if it is determined manually. When determined automatically, it is chosen to span the data range (at least).
- `zlimmode` - `{'auto','manual'}` - Determines if `zlim` is determined automatically or if it is determined manually. When determined automatically, it is chosen to span the data range (at least).
- `xminorgrid` - `{'on','off'}` - Set to `on` to draw grid lines from minor ticks on the x axis.
- `yminorgrid` - `{'on','off'}` - Set to `on` to draw grid lines from minor ticks on the y axis.
- `zminorgrid` - `{'on','off'}` - Set to `on` to draw grid lines from minor ticks on the z axis.
- `xscale` - `{'linear','log'}` - Determines if the data on the x axis is linear or logarithmically scaled.
- `yscale` - `{'linear','log'}` - Determines if the data on the y axis is linear or logarithmically scaled.
- `zscale` - `{'linear','log'}` - Determines if the data on the z axis is linear or logarithmically scaled.
- `xtick` - `vector` - A vector of x coordinates where ticks are placed on the x axis. Setting this vector allows you complete control over the placement of ticks on the axis.
- `ytick` - `vector` - A vector of y coordinates where ticks are placed on the y axis. Setting this vector allows you complete control over the placement of ticks on the axis.
- `ztick` - `vector` - A vector of z coordinates where ticks are placed on the z axis. Setting this vector allows you complete control over the placement of ticks on the axis.

- `xticklabel` - `string vector` - A string vector, of the form `'string`string—string'— that contains labels to assign to the labels on the axis. If this vector is shorter than `xtick`, then FreeMat will cycle through the elements of this vector to fill out the labels.
- `yticklabel` - `string vector` - A string vector, of the form `'string`string—string'— that contains labels to assign to the labels on the axis. If this vector is shorter than `ytick`, then FreeMat will cycle through the elements of this vector to fill out the labels.
- `zticklabel` - `string vector` - A string vector, of the form `'string`string—string'— that contains labels to assign to the labels on the axis. If this vector is shorter than `ztick`, then FreeMat will cycle through the elements of this vector to fill out the labels.
- `xtickmode` - `{'auto','manual'}` - Set to `'auto'` if you want FreeMat to calculate the tick locations. Setting `'xtick'` will cause this property to switch to `'manual'`.
- `ytickmode` - `{'auto','manual'}` - Set to `'auto'` if you want FreeMat to calculate the tick locations. Setting `'ytick'` will cause this property to switch to `'manual'`.
- `ztickmode` - `{'auto','manual'}` - Set to `'auto'` if you want FreeMat to calculate the tick locations. Setting `'ztick'` will cause this property to switch to `'manual'`.
- `xticklabelmode` - `{'auto','manual'}` - Set to `'auto'` if you want FreeMat to set the tick labels. This will be based on the vector `xtick`.
- `yticklabelmode` - `{'auto','manual'}` - Set to `'auto'` if you want FreeMat to set the tick labels. This will be based on the vector `ytick`.
- `zticklabelmode` - `{'auto','manual'}` - Set to `'auto'` if you want FreeMat to set the tick labels. This will be based on the vector `ztick`.

23.4 CLA Clear Current Axis

23.4.1 Usage

Clears the current axes. The syntax for its use is

```
cla
```

23.5 CLABEL Add Labels To Contour Plot

23.5.1 Usage

The `clabel` function adds labels to a contour plot Generate contour labels for a contour plot. The syntax for its use is either:

```
handles = clabel(contourhandle,property,value,property,value,...)
```

which labels all of the contours in the plot, or

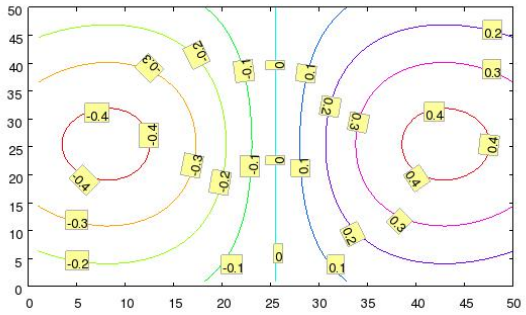
```
handles = clabel(contourhandle,vals,property,value,property,value,...)
```

which only labels those contours indicated by the vector `vals`. The `contourhandle` must be the handle to a contour plot object. The remaining property/value pairs are passed to the `text` function to control the parameters of the generated text labels. See the `text properties` for the details on what can be used in those labels.

23.5.2 Example

```
--> [x,y] = meshgrid(linspace(-1,1,50));
--> z = x.*exp(-(x.^2+y.^2));
--> h = contour(z);
--> clabel(h,'backgroundcolor',[1,1,.6],'edgecolor',[.7,.7,.7]);
```

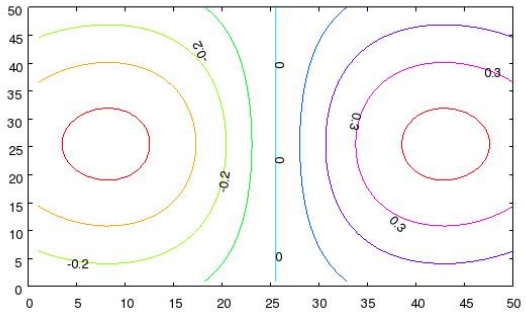
which results in



Alternately, we can just label a subset of the contours

```
--> h = contour(z);
--> clabel(h,[-.2,0,.3]);
```

which results in



23.6 CLF Clear Figure

23.6.1 Usage

This function clears the contents of the current figure. The syntax for its use is

```
clf
```

23.7 CLIM Adjust Color limits of plot

23.7.1 Usage

There are several ways to use `clim` to adjust the color limits of a plot. The various syntaxes are

```
clim
clim([lo,hi])
clim('auto')
clim('manual')
clim('mode')
clim(handle,...)
```

The first form (without arguments), returns a 2-vector containing the current limits. The second form sets the limits on the plot to `[lo,hi]`. The third and fourth form set the mode for the limit to `auto` and `manual` respectively. In `auto` mode, FreeMat chooses the range for the axis automatically. The `clim('mode')` form returns the current mode for the axis (either `'auto'` or `'manual'`).

Switching to `manual` mode does not change the limits, it simply allows you to modify them (and disables the automatic adjustment of the limits as more objects are added to the plot). Also, if you specify a set of limits explicitly, the mode is set to `manual`

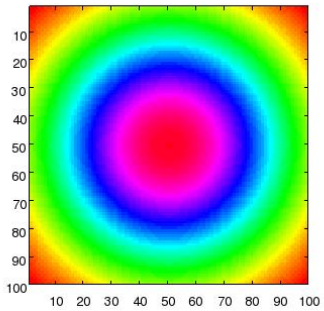
Finally, you can specify the handle of an axis to manipulate instead of using the current one.

23.7.2 Example

Here is an example of using `clim` to change the effective window and level onto an image. First, the image with default limits

```
--> x = repmat(linspace(-1,1),[100,1]); y = x';
--> z = exp(-x.^2-y.^2);
--> image(z);
--> min(z(:))

ans =
    0.1353

--> max(z(:))

ans =
    0.9998
```

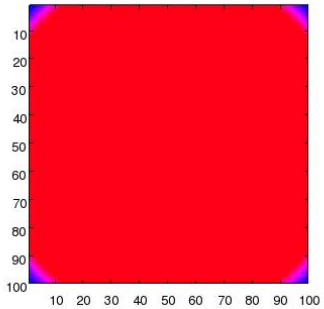
which results in



Next, we change the colorscale of the image using the `clim` function

```
--> image(z);
--> clim([0,0.2]);
```

which results in



23.8 CLOSE Close Figure Window

23.8.1 Usage

Closes a figure window, either the currently active window, a window with a specific handle, or all figure windows. The general syntax for its use is

```
close(handle)
```

in which case the figure window with the speicified `handle` is closed. Alternately, issuing the command with no argument

```
close
```

is equivalent to closing the currently active figure window. Finally the command

```
close('all')
```

closes all figure windows currently open.

23.9 COLORBAR Add Colorbar to Current Plot

23.9.1 Usage

There are a number of syntaxes for the `colorbar` command. The first takes no arguments, and adds a vertical colorbar to the right of the current axes.

```
colorbar
```

You can also pass properties to the newly created axes object using the second syntax for colorbar

```
colorbar(properties...)
```

23.10 COLORMAP Image Colormap Function

23.10.1 Usage

Changes the colormap for the current figure. The generic syntax for its use is

```
colormap(map)
```

where `map` is a an array organized as `3 \times N`), which defines the RGB (Red Green Blue) coordinates for each color in the colormap. You can also use the function with no arguments to recover the current colormap

```
map = colormap
```

23.10.2 Function Internals

Assuming that the contents of the colormap function argument `c` are labeled as:

$$c = \begin{bmatrix} r_1 & g_1 & b_1 \\ r_1 & g_2 & b_2 \\ r_1 & g_3 & b_3 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

then these columns for the RGB coordinates of pixel in the mapped image. Assume that the image occupies the range $[a, b]$. Then the RGB color of each pixel depends on the value x via the following integer

$$k = 1 + \lfloor 256 \frac{x - a}{b - a} \rfloor,$$

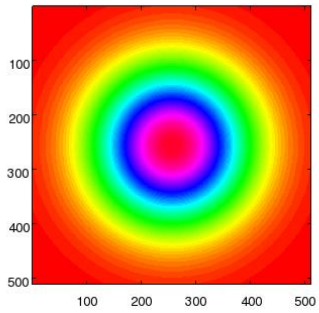
so that a pixel corresponding to image value x will receive RGB color $[r_k, g_k, b_k]$. Colormaps are generally used to pseudo color images to enhance visibility of features, etc.

23.10.3 Examples

We start by creating a smoothly varying image of a 2D Gaussian pulse.

```
--> x = linspace(-1,1,512)'*ones(1,512);
--> y = x';
--> Z = exp(-(x.^2+y.^2)/0.3);
--> image(Z);
```

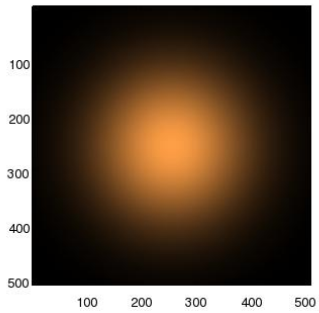
which we display with the default (grayscale) colormap here.



Next we switch to the `copper` colormap, and redisplay the image.

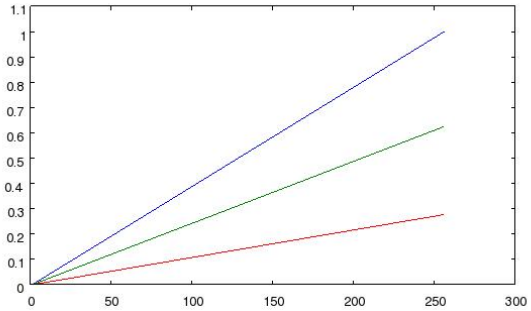
```
--> colormap(copper);
--> image(Z);
```

which results in the following image.



If we capture the output of the `copper` command and plot it, we obtain the following result:

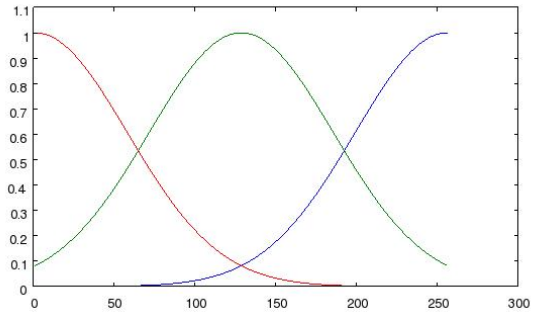
```
--> a = copper;
--> plot(a);
```



Note that in the output that each of the color components are linear functions of the index, with the ratio between the red, blue and green components remaining constant as a function of index. The result is

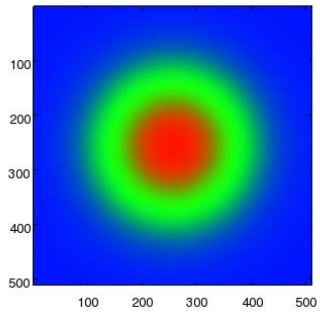
an intensity map with a copper tint. We can similarly construct a colormap of our own by defining the three components seperately. For example, suppose we take three gaussian curves, one for each color, centered on different parts of the index space:

```
--> t = linspace(0,1,256);
--> A = [exp(-(t-1.0).^2/0.1);exp(-(t-0.5).^2/0.1);exp(-t.^2/0.1)]';
--> plot(A);
```



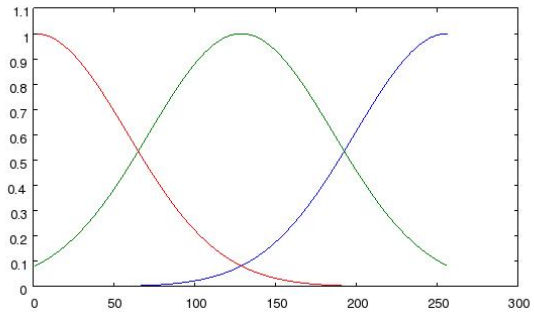
The resulting image has dark bands in it near the color transitions.

```
--> image(Z);
--> colormap(A);
```



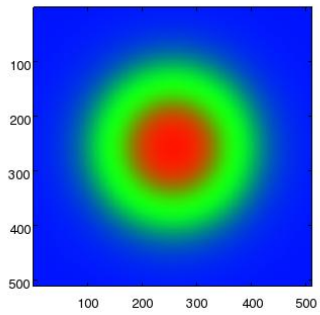
These dark bands are a result of the nonuniform color intensity, which we can correct for by renormalizing each color to have the same norm.

```
--> w = sqrt(sum(A'.^2));
--> sA = diag(1./w)*A;
--> plot(A);
```



The resulting image has no more dark bands.

```
--> image(Z);
--> colormap(A);
```



23.11 COLORSPEC Color Property Description

23.11.1 Usage

There are a number of ways of specifying a color value for a color-based property. Examples include line colors, marker colors, and the like. One option is to specify color as an RGB triplet

```
set(h,'color',[r,g,b])
```

where `r,g,b` are between @[0,1]@. Alternately, you can use color names to specify a color.

- `'none'` - No color.
- `'y','yellow'` - The color @[1,1,0]@ in RGB space.
- `'m','magenta'` - The color @[1,0,1]@ in RGB space.
- `'c','cyan'` - The color @[0,1,1]@ in RGB space.
- `'r','red'` - The color @[1,0,0]@ in RGB space.
- `'g','green'` - The color @[0,1,0]@ in RGB space.
- `'b','blue'` - The color @[0,0,1]@ in RGB space.
- `'w','white'` - The color @[1,1,1]@ in RGB space.
- `'k','black'` - The color @[0,0,0]@ in RGB space.

23.12 CONTOUR Contour Plot Function

23.12.1 Usage

This command generates contour plots. There are several syntaxes for the command. The simplest is

```
contour(Z)
```

which generates a contour plot of the data in matrix `Z`, and will automatically select the contour levels. The `x,y` coordinates of the contour default to `1:n` and `1:m`, where `n` is the number of columns and `m` is the number of rows in the `Z` matrix. Alternately, you can specify a scalar `n`

```
contour(Z,n)
```

which indicates that you want `n` contour levels. For more control, you can provide a vector `v` containing the levels to contour. If you want to generate a contour for a particular level, you must pass a vector `[t,t]` where `t` is the level you want to contour. If you have data that lies on a particular `X,Y` grid, you can pass either vectors `x,y` or matrices `X,Y` to the contour function via

```
contour(X,Y,Z)
contour(X,Y,Z,n)
contour(X,Y,Z,v)
```

Each form of `contour` can optionally take a line spec to indicate the color and linestyle of the contours to draw:

```
contour(...,linespec)
```

or any of the other forms of `contour`. Furthermore, you can supply an axis to target the `contour` plot to (so that it does not get added to the current axis, which is the default):

```
contour(axis_handle,...)
```

Finally, the `contour` command returns a handle to the newly returned contour plot.

```
handle = contour(...)
```

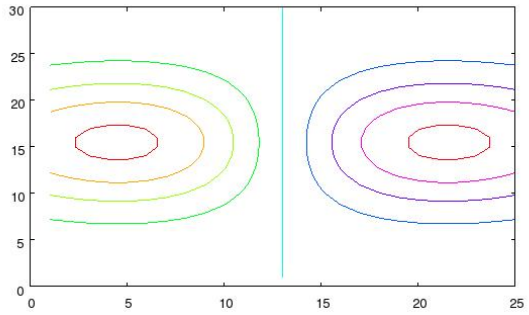
To place labels on the contour plot, use the `clabel` function.

23.12.2 Example

Here is a simple example of a contour plot with the default `x,y` coordinates:

```
--> [x,y] = meshgrid(linspace(-1,1,25),linspace(-2,2,30));
--> z = x.*exp(-x.^2-y.^2);
--> contour(z)
```

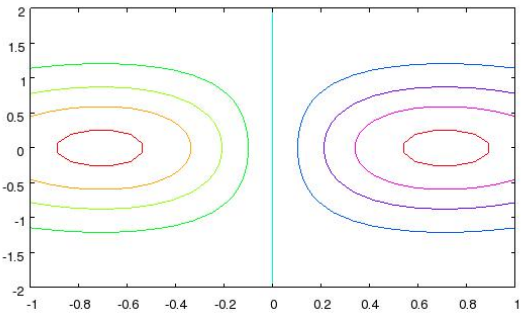
which results in the following plot



Here, we specify the `x` and `y` coordinates explictly

```
--> contour(x,y,z)
```

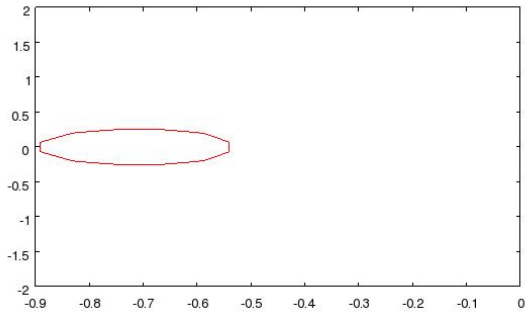
note that the axis limits have changed appropriately



By default, contours are created at values selected by FreeMat. To provide our own set of contour values (asymmetrically about zero in this case), we supply them as

```
--> contour(x,y,z,[-.4,0.,3])
```

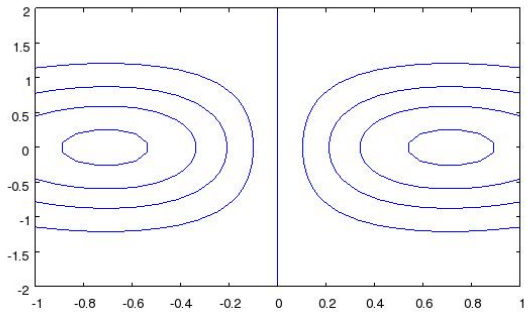
which is here



Also be default, `contour` uses the current color map and `clim` limits to determine the coloring of the contours. Here, we override the color spec so that we have all black contour

```
--> contour(x,y,z,'b-')
```

which is here



23.13 CONTOUR3 3D Contour Plot Function

23.13.1 Usage

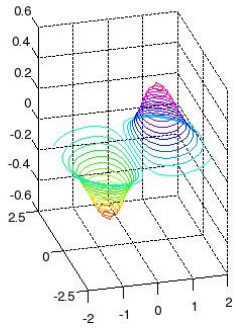
This command generates contour plots where the lines are plotted in 3D. The syntax for its use is identical to the `contour` function. Please see its help for details.

23.13.2 Example

Here is a simple example of a 3D contour plot.

```
--> [x,y] = meshgrid([-2:.25:2]);
--> z=x.*exp(-x.^2-y.^2);
--> contour3(x,y,z,30);
--> axis square;
--> view(-15,25)
```

The resulting plot



23.14 COPPER Copper Colormap

23.14.1 Usage

Returns a copper colormap. The syntax for its use is

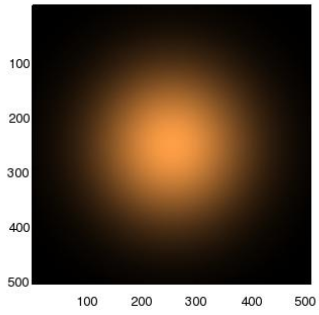
```
y = copper
```

23.14.2 Example

Here is an example of an image displayed with the `copper` colormap

```
--> x = linspace(-1,1,512)'*ones(1,512);
--> y = x';
--> Z = exp(-(x.^2+y.^2)/0.3);
--> image(Z);
--> colormap(copper);
```

which results in the following image



23.15 COPY Copy Figure Window

23.15.1 Usage

Copies the currently active figure window to the clipboard. The syntax for its use is:

```
copy
```

The resulting figure is copied as a bitmap to the clipboard, and can then be pasted into any suitable application.

23.16 COUNTOUR Contour Object Properties

23.16.1 Usage

Below is a summary of the properties for a line series.

- `contourmatrix` - `array` - the matrix containing contour data for the plot. This is a `2 x N` matrix containing x and y coordinates for points on the contours. In addition, each contour line starts with a column containing the number of points and the contour value.
- `displayname` - `string` - The name of this line series as it appears in a legend.
- `floating` - `{'on','off'}` - set to on to have floating (3D) contours
- `levellist` - `vector` - a vector of Z-values for the contours
- `levellistmode` - `{'auto','manual'}` - set to auto for automatic selection of Z-values of the contours.
- `linecolor` - color of the contour lines.
- `linestyle` - `{'-','--',':','-.','none'}` - the line style to draw the contour in.
- `linewidth` - `scalar` - the width of the lines
- `parent` - `handle` - The axis that contains this object
- `tag` - `string` - A string that can be used to tag the object.
- `type` - `string` - Returns the string `'contour'`.
- `userdata` - `array` - Available to store any variable you want in the handle object.
- `visible` - `{'on','off'}` - Controls visibility of the the line.
- `xdata` - `matrix` - Contains the x coordinates of the surface on which the zdata is defined. This can either be a monotonic vector of the same number of columns as `zdata`, or a 2D matrix that is the same size as `zdata`.
- `xdatamode` - `{'auto','manual'}` - When set to `'auto'` FreeMat will autogenerate the x coordinates for the contours. These values will be `1,..,N` where `N` is the number of columns of `zdata`.
- `ydata` - `matrix` - Contains the y coordinates of the surface on which the zdata is defined. This can either be a monotonic vector of the same number of rows as `zdata` or a 2D matrix that is the same size as `zdata`.
- `ydatamode` - `{'auto','manual'}` - When set to `'auto'` FreeMat will autogenerate the y coordinates for the contour data.
- `zdata` - `matrix` - The matrix of z values that are to be contoured.

23.17 DATACURSORMODE Interactive Data Cursor

23.17.1 Usage

Toggles the data cursor which allows interactive data exploration.

```
datacursormode('on')
```

23.18 DRAWNOW Flush the Event Queue

23.18.1 Usage

The `drawnow` function can be used to process the events in the event queue of the FreeMat application. The syntax for its use is

```
drawnow
```

Now that FreeMat is threaded, you do not generally need to call this function, but it is provided for compatibility.

23.19 FIGLOWER Lower a Figure Window

23.19.1 Usage

Lowers a figure window indicated by the figure number. The syntax for its use is

```
figlower(fignum)
```

where `fignum` is the number of the figure to lower. The figure will be lowerd to the bottom of the GUI stack (meaning that it we be behind other windows). Note that this function does not cause `fignum` to become the current figure, you must use the `figure` command for that. Similarly, if `fignum` is the current figure, it will remain the current figure (even though the figure is now behind others).

23.20 FIGRAISE Raise a Figure Window

23.20.1 Usage

Raises a figure window indicated by the figure number. The syntax for its use is

```
figraise(fignum)
```

where `fignum` is the number of the figure to raise. The figure will be raised to the top of the GUI stack (meaning that it we be visible). Note that this function does not cause `fignum` to become the current figure, you must use the `figure` command for that.

23.21 FIGURE Figure Window Select and Create Function

23.21.1 Usage

Changes the active figure window to the specified figure number. The general syntax for its use is

```
figure(number)
```

where `number` is the figure number to use. If the figure window corresponding to `number` does not already exist, a new window with this number is created. If it does exist then it is brought to the forefront and made active. You can use `gcf` to obtain the number of the current figure.

Note that the figure number is also the handle for the figure. While for most graphical objects (e.g., axes, lines, images), the handles are large integers, for figures, the handle is the same as the figure number. This means that the figure number can be passed to `set` and `get` to modify the properties of the current figure, (e.g., the colormap). So, for figure `3`, for example, you can use `get(3,'colormap')` to retrieve the colormap for the current figure.

23.22 FIGUREPROPERTIES Figure Object Properties

23.22.1 Usage

Below is a summary of the properties for the axis.

- `alphamap` - `vector` - Contains the alpha (transparency) map for the figure. If this is set to a scalar, then all values are mapped to the same transparency. It defaults to `1`, which is all values being fully opaque. If you set this to a vector, the values of graphics objects will be mapped to different transparency values, based on the setting of their `alphadatamapping` property.
- `color` - `colorspec` - The background color of the figure (defaults to a gray `[0.6,0.6,0.6]`). During printing, this color is set to white, and then is restored.
- `colormap` - `color vector` - an `N x 3` matrix of RGB values that specifies the colormap for the figure. Defaults to an `HSV` map.
- `children` - `handle vector` - the handles for objects that are children of this figure. These should be axis objects.
- `currentaxes` - `handle` - the handle for the current axes. Also returned by `gca`.
- `doublebuffer` - Not used.
- `parent` - Not used.
- `position` - Not used.
- `type` - `string` - returns the string `'figure'`.
- `userdata` - `array` - arbitrary array you can use to store data associated with the figure.
- `nextplot` - `{'add','replace','replacechildren'}` - If set to `'add'` then additional axes are added to the list of children for the current figure. If set to `'replace'`, then a new axis replaces all of the existing children.
- `figsize` - `two vector` - the size of the figure window in pixels (width x height).
- `renderer` - `{'painters','opengl'}` - When set to `'painters'` drawing is based on the Qt drawing methods (which can handle flat shading of surfaces with transparency). If you set the renderer to `'opengl'` then OpenGL is used for rendering. Support for OpenGL is currently in the alpha stage, and FreeMat does not enable it automatically. You can set the renderer mode to `'opengl'` manually to experiment. Also, OpenGL figures cannot be printed yet.

23.23 GCA Get Current Axis

23.23.1 Usage

Returns the handle for the current axis. The syntax for its use is

```
handle = gca
```

where `handle` is the handle of the active axis. All object creation functions will be children of this axis.

23.24 GCF Get Current Figure

23.24.1 Usage

Returns the figure number for the current figure (which is also its handle, and can be used to set properties of the current figure using `set`). The syntax for its use is

```
figure_number = gcf
```

where `figure\_number` is the number of the active figure (also the handle of the figure).

Note that figures have handles, just like axes, images, plots, etc. However the handles for figures match the figure number (while handles for other graphics objects tend to be large, somewhat arbitrary integers). So, to retrieve the colormap of the current figure, you could use `get(gcf,'colormap')`, or to obtain the colormap for figure 3, use `get(3,'colormap')`.

23.25 GET Get Object Property

23.25.1 Usage

This function allows you to retrieve the value associated with a property. The syntax for its use is

```
value = get(handle,property)
```

where `property` is a string containing the name of the property, and `value` is the value for that property. The type of the variable `value` depends on the property being set. See the help for the properties to see what values you can set.

23.26 GLSHOW Show a GL Assembly in a GL Window

23.26.1 Usage

Shows a GL Assembly in a GL Window. The syntax for its use is

```
glshow(name,scale)
```

which shows the `glassembly` named `name` in a new GL window, with the scale set to `scale`. Roughly speaking `scale` should represent the radial size of the object that you want to see in the window.

23.27 GRAY Gray Colormap

23.27.1 Usage

Returns a gray colormap. The syntax for its use is

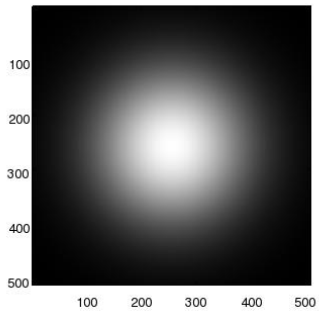
```
y = gray
```

23.27.2 Example

Here is an example of an image displayed with the `gray` colormap

```
--> x = linspace(-1,1,512)'*ones(1,512);
--> y = x';
--> Z = exp(-(x.^2+y.^2)/0.3);
--> image(Z);
--> colormap(gray);
```

which results in the following image



23.28 GRID Plot Grid Toggle Function

23.28.1 Usage

Toggles the drawing of grid lines on the currently active plot. The general syntax for its use is

```
grid(state)
```

where `state` is either

```
grid('on')
```

to activate the grid lines, or

```
grid('off')
```

to deactivate the grid lines. If you specify no argument then `grid` toggles the state of the grid:

```
grid
```

You can also specify a particular axis to the grid command

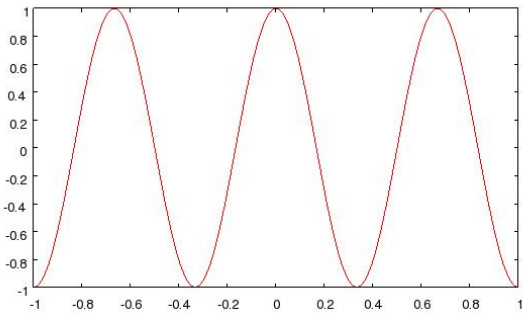
```
grid(handle,...)
```

where `handle` is the handle for a particular axis.

23.28.2 Example

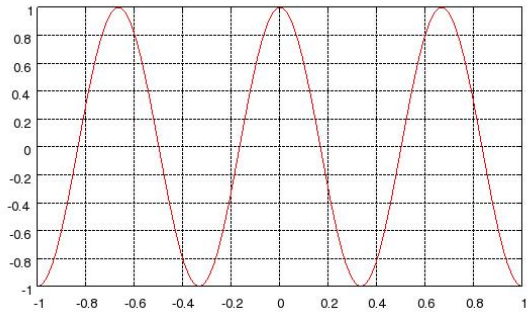
Here is a simple plot without grid lines.

```
--> x = linspace(-1,1);
--> y = cos(3*pi*x);
--> plot(x,y,'r-');
```



Next, we activate the grid lines.

```
--> plot(x,y,'r-');
--> grid on
```



23.29 HCONTOUR Create a contour object

23.29.1 Usage

Creates a contour object and parents it to the current axis. The syntax for its use is

```
handle = hcontour(property,value,property,value,...)
```

where `property` and `value` are set. The handle ID for the resulting object is returned. It is automatically added to the children of the current axis.

23.30 HIMAGE Create a image object

23.30.1 Usage

Creates a image object and parents it to the current axis. The syntax for its use is

```
handle = himage(property,value,property,value,...)
```

where `property` and `value` are set. The handle ID for the resulting object is returned. It is automatically added to the children of the current axis.

23.31 HIST Histogram Function

23.31.1 Usage

```
n=hist (y)
n=hist (y,x)
n=hist (y,x,norm)
```

Produce histogram counts or plots.

With one vector input argument, plot a histogram of the values with 10 bins. The range of the histogram bins is determined by the range of the data.

Given a second scalar argument, use that as the number of bins.

Given a second vector argument, use that as the centers of the bins, with the width of the bins determined from the adjacent values in the vector.

If third argument is provided, the histogram is normalised such that the sum of the bars is equal to `norm`.

Extreme values are lumped in the first and last bins.

23.32 HLINE Create a line object

23.32.1 Usage

Creates a line object and parents it to the current axis. The syntax for its use is

```
handle = hline(property,value,property,value,...)
```

where `property` and `value` are set. The handle ID for the resulting object is returned. It is automatically added to the children of the current axis.

23.33 HOLD Plot Hold Toggle Function

23.33.1 Usage

Toggles the hold state on the currently active plot. The general syntax for its use is

```
hold(state)
```

where `state` is either

```
hold('on')
```

to turn hold on, or

```
hold('off')
```

to turn hold off. If you specify no argument then `hold` toggles the state of the hold:

```
hold
```

You can also specify a particular axis to the hold command

```
hold(handle,...)
```

where `handle` is the handle for a particular axis.

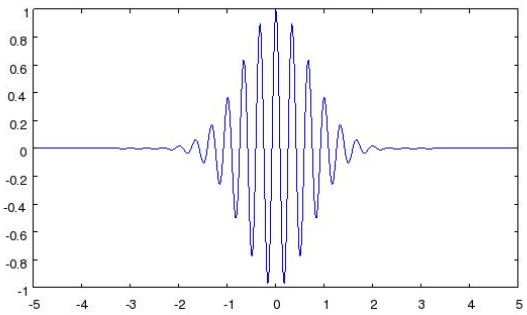
23.33.2 Function Internals

The `hold` function allows one to construct a plot sequence incrementally, instead of issuing all of the plots simultaneously using the `plot` command.

23.33.3 Example

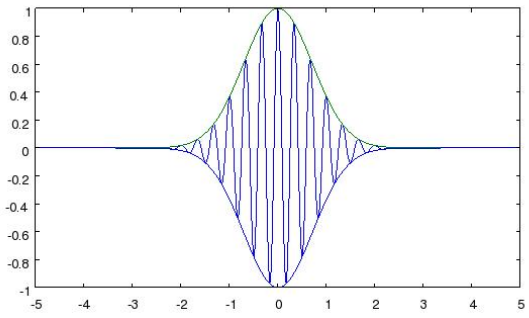
Here is an example of using both the `hold` command and the multiple-argument `plot` command to construct a plot composed of three sets of data. The first is a plot of a modulated Gaussian.

```
--> x = linspace(-5,5,500);
--> t = exp(-x.^2);
--> y = t.*cos(2*pi*x*3);
--> plot(x,y);
```



We now turn the hold state to `'on'`, and add another plot sequence, this time composed of the top and bottom envelopes of the modulated Gaussian. We add the two envelopes simultaneously using a single `plot` command. The fact that `hold` is `'on'` means that these two envelopes are added to (instead of replace) the current contents of the plot.

```
--> plot(x,y);
--> hold on
--> plot(x,t,'g-',x,-t,'b-')
```



23.34 HPATCH Create a patch object

23.34.1 Usage

Creates a patch object and parents it to the current axis. The syntax for its use is

```
handle = hpatch(property,value,property,value,...)
```

where `property` and `value` are set. The handle ID for the resulting object is returned. It is automatically added to the children of the current axis.

23.35 HPOINT Get Point From Window

23.35.1 Usage

This function waits for the user to click on the current figure window, and then returns the coordinates of that click. The generic syntax for its use is

```
[x,y] = hpoint
```

23.36 HRAWPLOT Generate a Raw Plot File

23.36.1 Usage

This function takes a sequence of commands, and generates a raw plot (to a file) that renders the commands. It is a useful tool for creating high quality fully customized PDF plots from within FreeMat scripts that are portable. The syntax for its use

```
hrawplot(filename,commands)
```

where `filename` is the name of the file to plot to, and `commands` is a cell array of strings. Each entry in the cell array contains a string with a command text. The commands describe a simple mini-language for describing plots. The complete dictionary of commands is given

- `LINE x1 y1 x2 y2` – draw a line

- `FONT name size` – select a font of the given name and size
- `TEXT x1 y1 string` – draw the given text string at the given location
- `STYLE style` – select line style ('solid' or 'dotted')
- `PAGE` – force a new page
- `SIZE x1 y1` – Set the page mapping
- `BOX x1 y1 x2 y2` – draw a filled box covering the given coordinates
- `HTEXTBOX x1 y1 x2 y2 string` – Draw a horizontal box with the given string centered in it
- `VTEXTBOX x1 y1 x2 y2 string` – Draw a vertical box with the given string centered in it (rotated 90 degrees)
- `BRUSH string` – Set the brush color ('red','blue', etc)
- `PEN string` – Set the pen color

23.37 HSURFACE Create a surface object

23.37.1 Usage

Creates a surface object and parents it to the current axis. The syntax for its use is

```
handle = hsurface(property,value,property,value,...)
```

where `property` and `value` are set. The handle ID for the resulting object is returned. It is automatically added to the children of the current axis.

23.38 HTEXT Create a text object

23.38.1 Usage

Creates a text object and parents it to the current axis. The syntax for its use is

```
handle = htext(property,value,property,value,...)
```

where `property` and `value` are set. The handle ID for the resulting object is returned. It is automatically added to the children of the current axis.

23.39 HTEXTBITMAP Get Text Rendered as a Bitmap

23.39.1 Usage

This function takes a fontname, a size, and a text string and returns a bitmap containing the text. The generic syntax for its use is

```
bitmap = htextbitmap(fontname,size,text)
```

where the output bitmap contains the text rendered into a matrix.

23.40 IMAGE Image Display Function

23.40.1 Usage

The `image` command has the following general syntax

```
handle = image(x,y,C,properties...)
```

where `x` is a two vector containing the `x` coordinates of the first and last pixels along a column, and `y` is a two vector containing the `y` coordinates of the first and last pixels along a row. The matrix `C` constitutes the image data. It must either be a scalar matrix, in which case the image is colormapped using the `colormap` for the current figure. If the matrix is `M x N x 3`, then `C` is intepreted as RGB data, and the image is not colormapped. The `properties` argument is a set of `property/value` pairs that affect the final image. You can also omit the `x` and `y`,

```
handle = image(C, properties...)
```

in which case they default to `x = [1,size(C,2)]` and `y = [1,size(C,1)]`. Finally, you can use the `image` function with only formal arguments

```
handle = image(properties...)
```

To support legacy FreeMat code, you can also use the following form of `image`

```
image(C, zoomfactor)
```

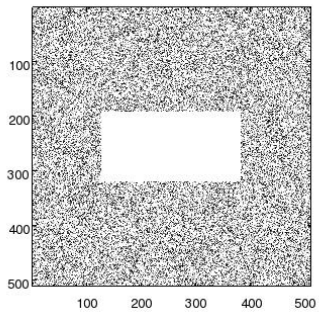
which is equivalent to `image(C)` with the axes removed so that the image takes up the full figure window, and the size of the figure window adjusted to achieve the desired zoom factor using the `zoom` command.

23.40.2 Example

In this example, we create an image that is `512 x 512` pixels square, and set the background to a noise pattern. We set the central `128 x 256` pixels to be white.

```
--> x = rand(512);
--> x((-64:63)+256,(-128:127)+256) = 1.0;
--> figure

ans =
 1

--> image(x)
--> colormap(gray)
```

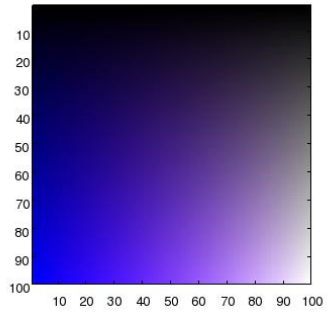
The resulting image looks like:



Here is an example of an RGB image

```
--> t = linspace(0,1);
--> red = t'*t;
--> green = t'*(t.^2);
--> blue = t'*(0*t+1);
--> A(:,:,1) = red;
--> A(:,:,2) = green;
--> A(:,:,3) = blue;
--> image(A);
```

The resulting image looks like:



23.41 IMAGEPROPERTIES Image Object Properties

23.41.1 Usage

Below is a summary of the properties for the axis.

- `alphadata` - `vector` - This is a vector that should contain as many elements as the image data itself `cdata`, or a single scalar. For a single scalar, all values of the image take on the same transparency. Otherwise, the transparency of each pixel is determined by the corresponding value from the `alphadata` vector.

- `alphadatamapping` - `{'scaled','direct','none'}` - For `none` mode (the default), no transparency is applied to the data. For `direct` mode, the vector `alphadata` contains values between @[0,M-1]— where `M` is the length of the alpha map stored in the figure. For `scaled` mode, the `alim` vector for the figure is used to linearly rescale the alpha data prior to lookup in the alpha map.

- `cdata` - `array` - This is either a `M x N` array or an `M x N x 3` array. If the data is `M x N` the image is a scalar image (indexed mode), where the color associated with each image pixel is computed using the colormap and the `cdatamapping` mode. If the data is `M x N x 3` the image is assumed to be in RGB mode, and the colorpanes are taken directly from `cdata` (the colormap is ignored). Note that in this case, the data values must be between @[0,1]— for each color channel and each pixel.

- `cdatamapping` - `{'scaled','direct'}` - For `scaled` (the default), the pixel values are scaled using the `clim` vector for the figure prior to looking up in the colormap. For `direct` mode, the pixel values must be in the range `[0,N-1` where `N` is the number of colors in the colormap if the data is integer type. For floating point data types, values must be in the range `[1,N]`.

- `children` - Not used.

- `parent` - `handle` - The axis containing the image.

- `tag` - `string` - You can set this to any string you want.

- `type` - `string` - Set to the string `'image'`.

- xdata - two vector - contains the x coordinates of the first and last column (respectively). Defaults to [1,C] where C is the number of columns in the image.
- ydata - two vector - contains the y coordinates of the first and last row (respectively). Defaults to [1,R] where R is the number of rows in the image.
- userdata - array - Available to store any variable you want in the handle object.
- visible - {'on','off'} - Controls whether the image is visible or not.

23.42 IMAGESC Image Display Function

23.42.1 Usage

The imagesc command has the following general syntax

```
handle = imagesc(x,y,C,clim)
```

where x is a two vector containing the x coordinates of the first and last pixels along a column, and y is a two vector containing the y coordinates of the first and last pixels along a row. The matrix C constitutes the image data. It must either be a scalar matrix, in which case the image is colormapped using the colormap for the current figure. If the matrix is M x N x 3, then C is intepreted as RGB data, and the image is not colormapped. The clim argument is a pairs [low high] that specifies scaling. You can also omit the x and y,

```
handle = imagesc(C, clim)
```

in which case they default to x = [1,size(C,2)] and y = [1,size(C,1)]. Finally, you can use the image function with only formal arguments

```
handle = imagesc(properties...)
```

23.42.2 Example

In this example, we create an image that is 512 x 512 pixels square, and set the background to a noise pattern. We set the central 128 x 256 pixels to be white.

```
--> x = rand(512);
--> x((-64:63)+256,(-128:127)+256) = 1.0;
--> figure

ans =
 1

--> imagesc(x,[0 .5])
--> colormap(gray)
```

23.43 IS2DVIEW Test Axes For 2D View

23.43.1 Usage

This function returns true if the current axes are in a 2-D view, and false otherwise. The generic syntax for its use is

```
y = is2dview(x)
```

where x is the handle of an axes object.

23.44 ISHOLD Test Hold Status

23.44.1 Usage

Returns the state of the `hold` flag on the currently active plot. The general syntax for its use is

```
ishold
```

and it returns a logical 1 if `hold` is `on`, and a logical 0 otherwise.

23.45 LEGEND Add Legent to Plot

23.45.1 Usage

This command adds a legend to the current plot. Currently, the following forms of the `legend` command are supported. The first form creates a legend with the given labels for the data series:

```
legend('label1','label2',...)
```

where `'label1'` is the text label associated with data plot 1 and so on. You can also use the `legend` command to control the appearance of the legend in the current plot. To remove the legend from the current plot, use

```
legend('off')
```

To hide the legend for the current plot (but do not remove it)

```
legend('hide')
```

And to show the legend that has been hidden, use

```
legend('show')
```

You can also toggle the display of the box surrounding the legend. Use

```
legend('boxoff')
```

or

```
legend('boxon')
```

to turn the legend box off or on, respectively. To toggle the visible state of the current legend, use

```
legend('toggle')
```

Specifying no arguments at all (apart from an optional location argument as specified below) results in the legend being rebuilt. This form is useful for picking up font changes or relocating the legend.

```
legend
```

By default, the `legend` command places the new legend in the upper right corner of the current plot. To change this behavior, use the `'location'` specifier (must be the last two options to the command)

```
legend(...,'location',option)
```

where `option` takes on the following possible values

- `north`,`N` - top center of plot
- `south`,`S` - bottom center of plot
- `east`,`E` - middle right of plot

- `west`,`W` - middle left of plot
- `northeast`,`NE` - top right of plot (default behavior)
- `northwest`,`NW` - top left of plot
- `southeast`,`SE` - bottom right of plot
- `southwest`,`SW` - bottom left of plot

This implementation of `legend` is incomplete relative to the MATLAB API. The functionality will be improved in future versions of FreeMat.

23.46 LINE Line Display Function

23.46.1 Usage

The `line` command has the following general syntax

```
handle = line(x,y,z,properties...)
```

where...

23.47 LINEPROPERTIES Line Series Object Properties

23.47.1 Usage

Below is a summary of the properties for a line series.

- `color` - `colorspec` - The color that is used to draw the line.
- `children` - Not used.
- `displayname` - The name of this line series as it appears in a legend.
- `linestyle` - `{'-','--',':','-.','none'}` - The style of the line.
- `linewidth` - `scalar` - The width of the line.
- `marker` - `{'+','o','*','.','x','square','s','diamond','d','\^','v','>','<'}` - The marker for data points on the line. Some of these are redundant, as `'square'` `'s'` are synonyms, and `'diamond'` and `'d'` are also synonyms.
- `markeredgecolor` - `colorspec` - The color used to draw the marker. For some of the markers (circle, square, etc.) there are two colors used to draw the marker. This property controls the edge color (which for unfilled markers) is the primary color of the marker.
- `markerfacecolor` - `colorspec` - The color used to fill the marker. For some of the markers (circle, square, etc.) there are two colors used to fill the marker.
- `markersize` - `scalar` - Control the size of the marker. Defaults to 6, which is effectively the radius (in pixels) of the markers.
- `parent` - `handle` - The axis that contains this object.
- `tag` - `string` - A string that can be used to tag the object.
- `type` - `string` - Returns the string `'line'`.
- `visible` - `{'on','off'}` - Controls visibility of the the line.

- `xdata` - `vector` - Vector of x coordinates of points on the line. Must be the same size as the `ydata` and `zdata` vectors.
- `ydata` - `vector` - Vector of y coordinates of points on the line. Must be the same size as the `xdata` and `zdata` vectors.
- `zdata` - `vector` - Vector of z coordinates of points on the line. Must be the same size as the `xdata` and `ydata` vectors.
- `xdatamode` - `{'auto','manual'}` - When set to `'auto'` FreeMat will autogenerate the x coordinates for the points on the line. These values will be `1,..,N` where `N` is the number of points in the line.
- `userdata` - `array` - Available to store any variable you want in the handle object.

23.48 LOGLOG Log-Log Plot Function

23.48.1 Usage

This command has the exact same syntax as the `plot` command:

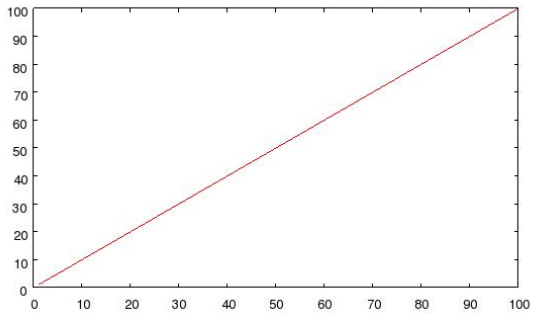
```
loglog(<data 1>,{linespec 1},<data 2>,{linespec 2}...,properties...)
```

in fact, it is a simple wrapper around `plot` that sets the x and y axis to have a logarithmic scale.

23.48.2 Example

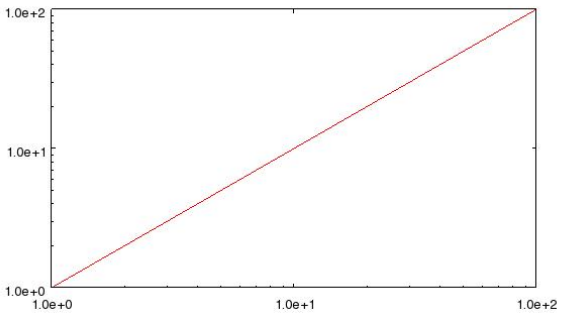
Here is an example of a doubly exponential signal plotted first on a linear plot:

```
--> x = linspace(1,100);
--> y = x;
--> plot(x,y,'r-');
```



and now on a log-log plot

```
--> loglog(x,y,'r-');
```



23.49 NEWPLOT Get Handle For Next Plot

23.49.1 Usage

Returns the handle for the next plot operation. The general syntax for its use is

```
h = newplot
```

This routine checks the `nextplot` properties of the current figure and axes to see if they are set to `replace` or not. If the figures `nextplot` property is set to replace, the current figure is cleared. If the axes `nextplot` property is set to `replace` then the axes are cleared for the next operation.

23.50 PATCH Patch Graphics Function

23.50.1 Usage

This routine is used to create a patch object that can be plotting 2D and 3D surfaces. A patch is a polygon defined by the xyz coordinates of its vertices and optionally by the color at the vertices. There are several forms for the `patch` function:

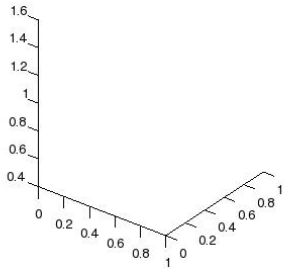
```
h = patch(X,Y,C,properties...)
h = patch(X,Y,Z,C,properties...)
h = patch(properties...)
h = patch(V)
```

Where `X`, `Y` and `Z` are matrices or vectors of `x`, `y` or `z` coordinates and `C` is a matrix or vector of color values (the colormap for the current fig is applied).

23.50.2 Example

Here we generate a surface specifying all four components.

```
--> x = [ 0 1 0 1]';
--> y = [ 0 0 1 1]';
--> c = [ 1 1 1 ];
--> patch(x,y,c)
--> axis equal
--> view(3)
```



23.51 PCOLOR Pseudocolor Plot

23.51.1 Usage

This routine is used to create a pseudocolor plot of the data. A pseudocolor plot is a essentially a surface plot seen from above. There are two forms for the `pcolor` command:

```
pcolor(C)
```

which uses a rectangular grid. Alternately, you can specify `X,Y` matrices or vectors.

```
pcolor(X,Y,C)
```

23.52 PLOT Plot Function

23.52.1 Usage

This is the basic plot command for FreeMat. The general syntax for its use is

```
plot(<data 1>,{linespec 1},<data 2>,{linespec 2}...,properties...)
```

where the `<data>` arguments can have various forms, and the `linespec` arguments are optional. We start with the `<data>` term, which can take on one of multiple forms:

- Vector Matrix Case – In this case the argument data is a pair of variables. A set of `x` coordinates in a numeric vector, and a set of `y` coordinates in the columns of the second, numeric matrix. `x` must have as many elements as `y` has columns (unless `y` is a vector, in which case only the number of elements must match). Each column of `y` is plotted sequentially against the common vector `x`.
- Unpaired Matrix Case – In this case the argument data is a single numeric matrix `y` that constitutes the `y`-values of the plot. An `x` vector is synthesized as `x = 1:length(y)`, and each column of `y` is plotted sequentially against this common `x` axis.
- Complex Matrix Case – Here the argument data is a complex matrix, in which case, the real part of each column is plotted against the imaginary part of each column. All columns receive the same line styles.

Multiple data arguments in a single plot command are treated as a *sequence*, meaning that all of the plots are overlapped on the same set of axes. The `linespec` is a string used to change the characteristics of the line. In general, the `linespec` is composed of three optional parts, the `colorspec`, the `symbolspec` and the `linestylespec` in any order. Each of these specifications is a single character that determines the corresponding characteristic. First, the `colorspec`:

- `'b'` - Color Blue
- `'g'` - Color Green
- `'r'` - Color Red
- `'c'` - Color Cyan
- `'m'` - Color Magenta
- `'y'` - Color Yellow
- `'k'` - Color Black

The `symbolspec` specifies the (optional) symbol to be drawn at each data point:

- `'.'` - Dot symbol
- `'o'` - Circle symbol
- `'x'` - Times symbol
- `'+'` - Plus symbol
- `'*'` - Asterisk symbol

- 's' - Square symbol
- 'd' - Diamond symbol
- 'v' - Downward-pointing triangle symbol
- '\^' - Upward-pointing triangle symbol
- '<' - Left-pointing triangle symbol
- '>' - Right-pointing triangle symbol

The linestylespec specifies the (optional) line style to use for each data series:

- '-' - Solid line style
- ':' - Dotted line style
- '-.' - Dot-Dash-Dot-Dash line style
- '--' - Dashed line style

For sequences of plots, the linespec is recycled with color order determined by the properties of the current axes. You can also use the properties argument to specify handle properties that will be inherited by all of the plots generated during this event. Finally, you can also specify the handle for the axes that are the target of the plot operation.

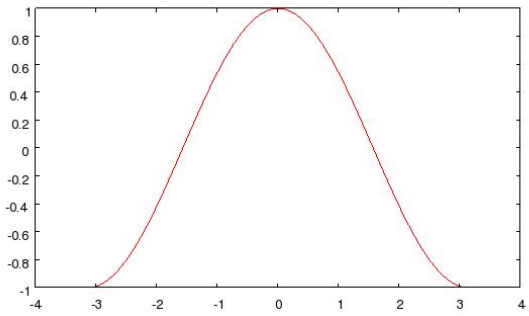
```
handle = plot(handle,...)
```

23.52.2 Example

The most common use of the plot command probably involves the vector-matrix paired case. Here, we generate a simple cosine, and plot it using a red line, with no symbols (i.e., a linespec of 'r-').

```
--> x = linspace(-pi,pi);
--> y = cos(x);
--> plot(x,y,'r-');
```

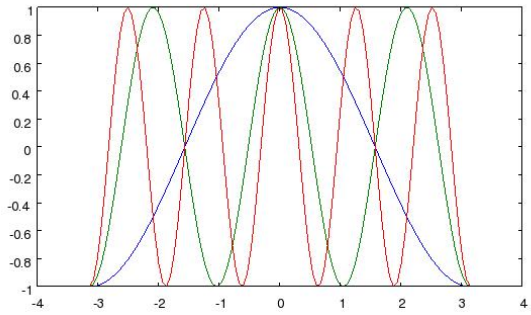
which results in the following plot.



Next, we plot multiple sinusoids (at different frequencies). First, we construct a matrix, in which each column corresponds to a different sinusoid, and then plot them all at once.

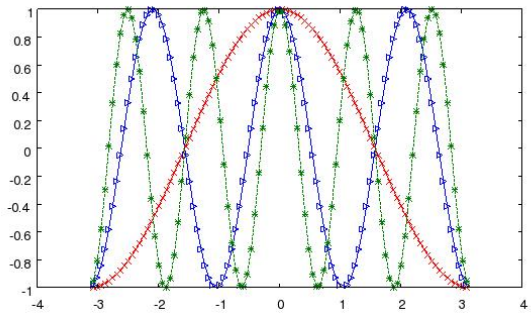
```
--> x = linspace(-pi,pi);
--> y = [cos(x(:)),cos(3*x(:)),cos(5*x(:))];
--> plot(x,y);
```

In this case, we do not specify a `linespec`, so that we cycle through the colors automatically (in the order listed in the previous section).



This time, we produce the same plot, but as we want to assign individual `linespecs` to each line, we use a sequence of arguments in a single plot command, which has the effect of plotting all of the data sets on a common axis, but which allows us to control the `linespec` of each plot. In the following example, the first line (harmonic) has red, solid lines with times symbols marking the data points, the second line (third harmonic) has blue, solid lines with right-pointing triangle symbols, and the third line (fifth harmonic) has green, dotted lines with asterisk symbols.

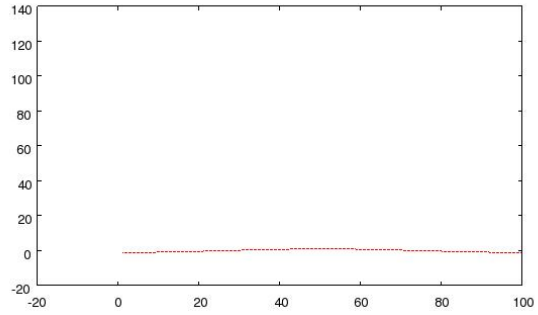
```
--> plot(x,y(:,1),'rx-',x,y(:,2),'b>-',x,y(:,3),'g*:');
```



The second most frequently used case is the unpaired matrix case. Here, we need to provide only one data component, which will be automatically plotted against a vector of natural number of the appropriate length. Here, we use a plot sequence to change the style of each line to be dotted, dot-dashed, and dashed.

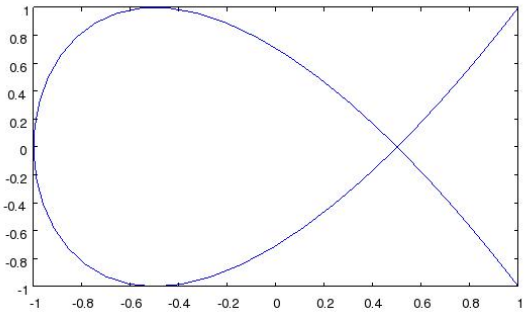
```
--> plot(y(:,1),'r:',y(:,2),'b;',y(:,3),'g|');
```

Note in the resulting plot that the `x`-axis no longer runs from `[-pi,pi]`, but instead runs from `[1,100]`.



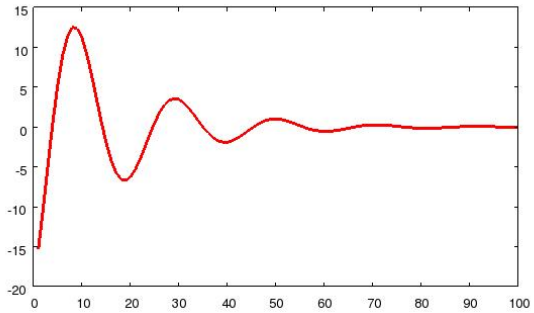
The final case is for complex matrices. For complex arguments, the real part is plotted against the imaginary part. Hence, we can generate a 2-dimensional plot from a vector as follows.

```
--> y = cos(2*x) + i * cos(3*x);
--> plot(y);
```



Here is an example of using the handle properties to influence the behavior of the generated lines.

```
--> t = linspace(-3,3);
--> plot(cos(5*t).*exp(-t),'r-','linewidth',3);
```



23.53 PLOT3 Plot 3D Function

23.53.1 Usage

This is the 3D plot command. The general syntax for its use is

```
plot3(X,Y,Z,{linespec 1},X,Y,Z,{linespec 2},...,properties...)
```

where `X` `Y` and `Z` are the coordinates of the points on the 3D line. Note that in general, all three should be vectors. If some or all of the quantities are matrices, then FreeMat will attempt to expand the vector arguments to the same size, and then generate multiple plots, one for each column of the matrices. The linespec is optional, see `plot` for details. You can specify `properties` for the generated line plots. You can also specify a handle as an axes to target

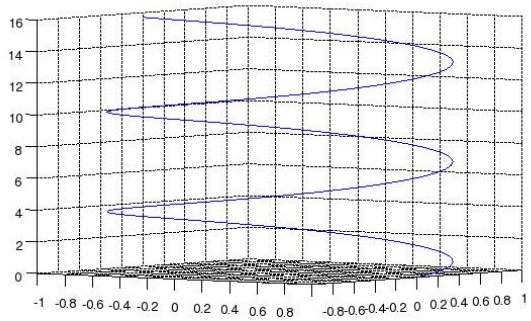
```
plot3(handle,...)
```

23.53.2 Example

Here is a simple example of a 3D helix.

```
--> t = linspace(0,5*pi,200);
--> x = cos(t); y = sin(t); z = t;
--> plot3(x,y,z);
--> view(3);
```

Shown here



23.54 POINT Get Axis Position From Mouse Click

23.54.1 Usage

Returns information about the currently displayed image based on a use supplied mouse-click. The general syntax for its use is

```
t = point
```

The returned vector `y` has two elements:

$$t = [x, y]$$

where `x,y` are the coordinates in the current axes of the click. This function has changed since FreeMat 1.10. If the click is not inside the active area of any set of axes, a pair of NaNs are returned.

23.55 PRINT Print a Figure To A File

23.55.1 Usage

This function "prints" the currently active fig to a file. The generic syntax for its use is

```
print(filename)
```

or, alternately,

```
print filename
```

where `filename` is the (string) filename of the destined file. The current fig is then saved to the output file using a format that is determined by the extension of the filename. The exact output formats may vary on different platforms, but generally speaking, the following extensions should be supported cross-platform:

- `jpg`, `jpeg` – JPEG file
- `pdf` – Portable Document Format file
- `png` – Portable Net Graphics file
- `svg` – Scalable Vector Graphics file

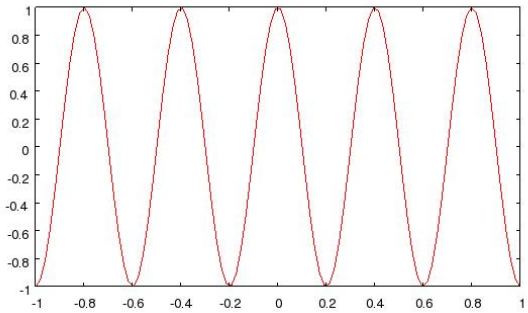
Postscript (PS, EPS) is supported on non-Mac-OSX Unix only. Note that only the fig is printed, not the window displaying the fig. If you want something like that (essentially a window-capture) use a seperate utility or your operating system's built in screen capture ability.

23.55.2 Example

Here is a simple example of how the figures in this manual are generated.

```
--> x = linspace(-1,1);
--> y = cos(5*pi*x);
--> plot(x,y,'r-');
--> print('printfig1.jpg')
--> print('printfig1.png')
```

which creates two plots `printfig1.png`, which is a Portable Net Graphics file, and `printfig1.jpg` which is a JPEG file.



23.56 PVALID Validate Property Name

23.56.1 Usage

This function checks to see if the given string is a valid property name for an object of the given type. The syntax for its use is

```
b = pvalid(type,propertyname)
```

where `string` is a string that contains the name of a valid graphics object type, and `propertyname` is a string that contains the name of the property to test for.

23.56.2 Example

Here we test for some properties on an `axes` object.

```
--> pvalid('axes','type')

ans =
 1

--> pvalid('axes','children')

ans =
 1

--> pvalid('axes','foobar')

ans =
 0
```

23.57 SEMILOGX Semilog X Axis Plot Function

23.57.1 Usage

This command has the exact same syntax as the `plot` command:

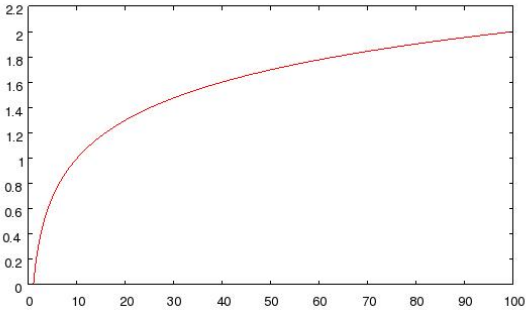
```
semilogx(<data 1>,{linespec 1},<data 2>,{linespec 2}...,properties...)
```

in fact, it is a simple wrapper around `plot` that sets the x axis to have a logarithmic scale.

23.57.2 Example

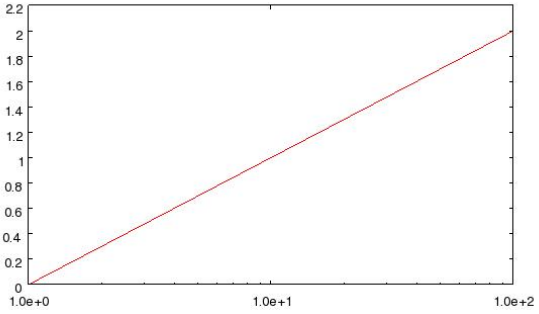
Here is an example of an exponential signal plotted first on a linear plot:

```
--> y = linspace(0,2);
--> x = (10).^y;
--> plot(x,y,'r-');
```



and now with a logarithmic x axis

```
--> semilogx(x,y,'r-');
```



23.58 SEMILOGY Semilog Y Axis Plot Function

23.58.1 Usage

This command has the exact same syntax as the `plot` command:

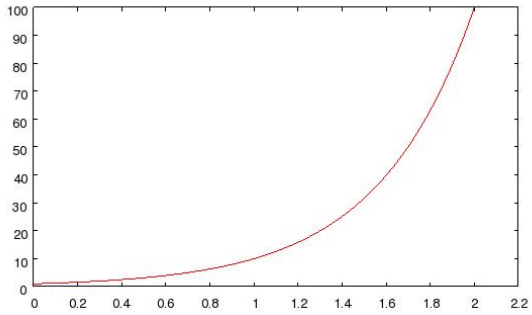
```
semilogy(<data 1>,{linespec 1},<data 2>,{linespec 2}...,properties...)
```

in fact, it is a simple wrapper around `plot` that sets the y axis to have a logarithmic scale.

23.58.2 Example

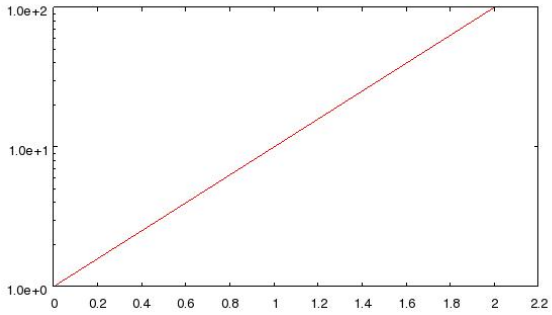
Here is an example of an exponential signal plotted first on a linear plot:

```
--> x = linspace(0,2);
--> y = 10.0.^x;
--> plot(x,y,'r-');
```



and now with a logarithmic y axis

```
--> semilogy(x,y,'r-');
```



23.59 SET Set Object Property

23.59.1 Usage

This function allows you to change the value associated with a property. The syntax for its use is

```
set(handle,property,value,property,value,...)
```

where `property` is a string containing the name of the property, and `value` is the value for that property. The type of the variable `value` depends on the property being set. See the help for the properties to see what values you can set.

23.60 SIZEFIG Set Size of Figure

23.60.1 Usage

The `sizefig` function changes the size of the currently selected fig window. The general syntax for its use is

```
sizefig(width,height)
```

where `width` and `height` are the dimensions of the fig window.

23.61 SUBPLOT Subplot Function

23.61.1 Usage

This function divides the current figure into a 2-dimensional grid, each of which can contain a plot of some kind. The function has a number of syntaxes. The first version

```
subplot(row,col,num)
```

which either activates subplot number `num`, or sets up a subplot grid of size `row x col`, and then activates `num`. You can also set up subplots that cover multiple grid elements

```
subplot(row,col,[vec])
```

where `vec` is a set of indexes covered by the new subplot. Finally, as a shortcut, you can specify a string with three components

```
subplot('mnp')
```

or using the alternate notation

```
subplot mnp
```

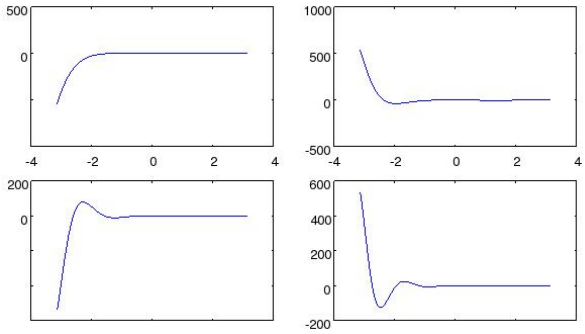
where `m` is the number of rows, `n` is the number of columns and `p` is the index. You can also specify the location of the subplot explicitly using the syntax

```
subplot('position',[left bottom width height])
```

23.61.2 Example

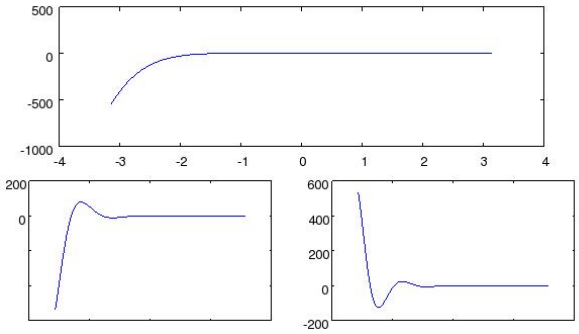
Here is the use of `subplot` to set up a `2 x 2` grid of plots

```
--> t = linspace(-pi,pi);
--> subplot(2,2,1)
--> plot(t,cos(t).*exp(-2*t));
--> subplot(2,2,2);
--> plot(t,cos(t*2).*exp(-2*t));
--> subplot(2,2,3);
--> plot(t,cos(t*3).*exp(-2*t));
--> subplot(2,2,4);
--> plot(t,cos(t*4).*exp(-2*t));
```



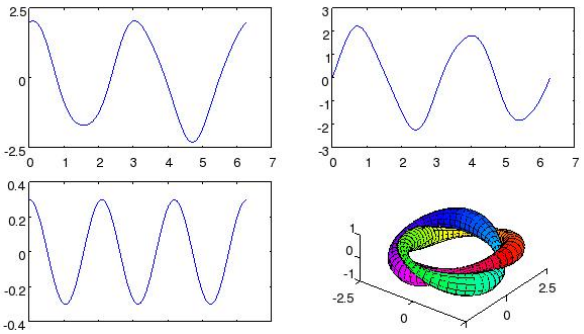
Here we use the second form of `subplot` to generate one subplot that is twice as large.

```
--> t = linspace(-pi,pi);
--> subplot(2,2,[1,2])
--> plot(t,cos(t).*exp(-2*t));
--> subplot(2,2,3);
--> plot(t,cos(t*3).*exp(-2*t));
--> subplot(2,2,4);
--> plot(t,cos(t*4).*exp(-2*t));
```



Note that the subplots can contain any handle graphics objects, not only simple plots.

```
--> t=0:(2*pi/100):(2*pi);
--> x=cos(t*2).*(2+sin(t*3)*.3);
--> y=sin(t*2).*(2+sin(t*3)*.3);
--> z=cos(t*3)*.3;
--> subplot(2,2,1)
--> plot(t,x);
--> subplot(2,2,2);
--> plot(t,y);
--> subplot(2,2,3);
--> plot(t,z);
--> subplot(2,2,4);
--> tubeplot(x,y,z,0.14*sin(t*5)+.29,t,10)
--> axis equal
--> view(3)
```



23.62 SURF Surface Plot Function

23.62.1 Usage

This routine is used to create a surface plot of data. A surface plot is a 3D surface defined by the xyz coordinates of its vertices and optionally by the color at the vertices. The most general syntax for the `surf` function is

```
h = surf(X,Y,Z,C,properties...)
```

Where `X` is a matrix or vector of `x` coordinates, `Y` is a matrix or vector of `y` coordinates, `Z` is a 2D matrix of coordinates, and `C` is a 2D matrix of color values (the colormap for the current fig is applied). In general, `X` and `Y` should be the same size as `Z`, but FreeMat will expand vectors to match the matrix if possible. If you want the color of the surface to be defined by the height of the surface, you can omit `C`

```
h = surf(X,Y,Z,properties...)
```

in which case `C=Z`. You can also eliminate the `X` and `Y` matrices in the specification

```
h = surf(Z,properties)
```

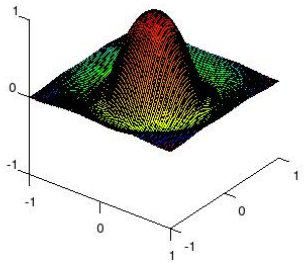
in which case they are set to `1:size(Z,2)` and `1:size(Y,2)` respectively. You can also specify a handle as the target of the `surf` command via

```
h = surf(handle,...)
```

23.62.2 Example

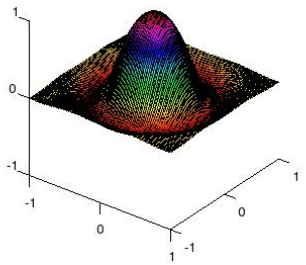
Here we generate a surface specifying all four components.

```
--> x = repmat(linspace(-1,1),[100,1]);
--> y = x';
--> r = x.^2+y.^2;
--> z = exp(-r*3).*cos(5*r);
--> c = r;
--> surf(x,y,z,c)
--> axis equal
--> view(3)
```



If we allow FreeMat to specify the color component, we see that the colorfield is the same as the height

```
--> surf(x,y,z)
--> axis equal
--> view(3)
```



23.63 SURFACEPROPERTIES Surface Object Properties

23.63.1 Usage

Below is a summary of the properties for the axis.

- `alphadata` - `vector` - This is a vector that should contain as many elements as the surface data itself `cdata`, or a single scalar. For a single scalar, all values of the surface take on the same transparency. Otherwise, the transparency of each pixel is determined by the corresponding value from the `alphadata` vector.
- `alphadatamapping` - `{'scaled','direct','none'}` - For `none` mode (the default), no transparency is applied to the data. For `direct` mode, the vector `alphadata` contains values between @[0,M-1]— where `M` is the length of the alpha map stored in the figure. For `scaled` mode, the `alim` vector for the figure is used to linearly rescale the alpha data prior to lookup in the alpha map.
- `ambientstrength` - Not used.
- `backfacelighting` - Not used.
- `cdata` - `array` - This is either a `M x N` array or an `M x N x 3` array. If the data is `M x N` the surface is a scalar surface (indexed mode), where the color associated with each surface pixel is computed using the colormap and the `cdatamapping` mode. If the data is `M x N x 3` the surface is assumed to be in RGB mode, and the colorpanes are taken directly from `cdata` (the colormap is ignored). Note that in this case, the data values must be between @[0,1]— for each color channel and each point on the surface.
- `cdatamapping` - `{'scaled','direct'}` - For `scaled` (the default), the pixel values are scaled using the `clim` vector for the figure prior to looking up in the colormap. For `direct` mode, the pixel values must be in the range `[0,N-1` where `N` is the number of colors in the colormap.
- `children` - Not used.
- `diffusestrength` - Not used.
- `edgealpha` - `{'flat','interp','scalar'}` - Controls how the transparency is mapped for the edges of the surface.
- `edgecolor` - `{'flat','interp','none',colorspec}` - Specifies how the edges are colored. For `'flat'` the edges are flat colored, meaning that the line segments that make up the edges are not shaded. The color for the line is determined by the first edge point it is connected to.
- `edgelighting` - Not used.
- `facealpha` - `{'flat','interp','texturemap',scalar}` - Controls how the transparency of the faces of the surface are controlled. For flat shading, the faces are constant transparency. For interp mode, the faces are smoothly transparently mapped. If set to a scalar, all faces have the same transparency.
- `facecolor` - `{'none','flat','interp',colorspec}` - Controls how the faces are colored. For `'none'` the faces are uncolored, and the surface appears as a mesh without hidden lines removed. For `'flat'` the surface faces have a constant color. For `'interp'` smooth shading is applied to the surface. And if a colorspec is provided, then the faces all have the same color.
- `facelighting` - Not used.
- `linestyle` - `{'-','--',':','-.','none'}` - The style of the line used to draw the edges.
- `linewidth` - `scalar` - The width of the line used to draw the edges.
- `marker` - `{'+','o','*','.','x','square','s','diamond','d','\^','v','>','<'}` - The marker for data points on the line. Some of these are redundant, as `'square' 's'` are synonyms, and `'diamond'` and `'d'` are also synonyms.
- `markeredgecolor` - `colorspec` - The color used to draw the marker. For some of the markers (circle, square, etc.) there are two colors used to draw the marker. This property controls the edge color (which for unfilled markers) is the primary color of the marker.

- markerfacecolor - colorspec - The color used to fill the marker. For some of the markers (circle, square, etc.) there are two colors used to fill the marker.
- markersize - scalar - Control the size of the marker. Defaults to 6, which is effectively the radius (in pixels) of the markers.
- meshstyle - {'both','rows','cols} - This property controls how the mesh is drawn for the surface. For rows and cols modes, only one set of edges is drawn.
- normalmode - Not used.
- parent - handle - The axis containing the surface.
- specularcolorreflectance - Not used.
- specularexponent - Not used.
- specularstrength - Not used.
- tag - string - You can set this to any string you want.
- type - string - Set to the string 'surface'.
- userdata - array - Available to store any variable you want in the handle object.
- vertexnormals - Not used.
- xdata - array - Must be a numeric array of size M x N which contains the x location of each point in the defined surface. Must be the same size as ydata and zdata. Alternately, you can specify an array of size 1 x N in which case FreeMat replicates the vector to fill out an M x N matrix.
- xdatamode - {'auto','manual'} - When set to auto then FreeMat will automatically generate the x coordinates.
- ydata - array - Must be a numeric array of size M x N which contains the y location of each point in the defined surface. Must be the same size as xdata and zdata. Alternately, you can specify an array of size M x 1 in which case FreeMat replicates the vector to fill out an M x N matrix.
- ydatamode - {'auto','manual'} - When set to auto then FreeMat will automatically generate the y coordinates.
- zdata - array - Must be a numeric array of size M x N which contains the y location of each point in the defined surface. Must be the same size as xdata and ydata.
- visible - {'on','off'} - Controls whether the surface is visible or not.

23.64 TEXT Add Text Label to Plot

23.64.1 Usage

Adds a text label to the currently active plot. The general syntax for it is use is either

```
text(x,y,'label')
```

where x and y are both vectors of the same length, in which case the text 'label' is added to the current plot at each of the coordinates x(i),y(i) (using the current axis to map these to screen coordinates). The second form supplies a cell-array of strings as the second argument, and allows you to place many labels simultaneously

```
text(x,y,{'label1','label2',....})
```

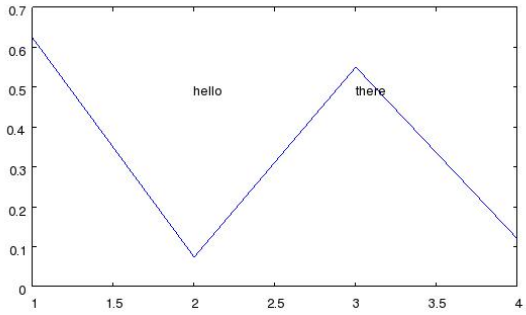
where the number of elements in the cell array must match the size of vectors x and y. You can also specify properties for the labels via

```
handles = text(x,y,{labels},properties...)
```

23.64.2 Example

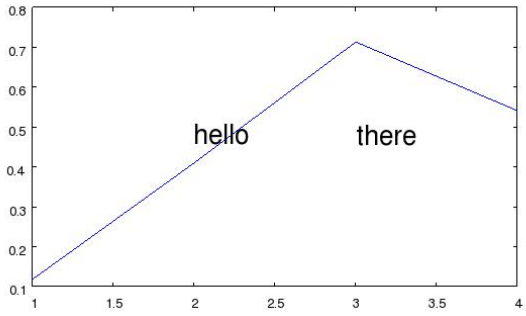
Here is an example of a few labels being added to a random plot:

```
--> plot(rand(1,4))
--> text([2,3],[0.5,0.5],{'hello','there'})
```



Here is the same example, but with larger labels:

```
--> plot(rand(1,4))
--> text([2,3],[0.5,0.5],{'hello','there'},'fontsize',20)
```



23.65 TEXTPROPERTIES Text Object Properties

23.65.1 Usage

Below is a summary of the properties for a text object.

- `boundingbox` - `four vector` - The size of the bounding box containing the text (in pixels). May contain negative values if the text is slanted.
- `children` - Not used.
- `string` - `string` - The text contained in the label.
- `extent` - Not used.
- `horizontalalignment` - `{'left','center','right'}` - Controls the alignment of the text relative to the specified position point.
- `position` - `three vector` - The position of the label in axis coordinates.
- `rotation` - `scalar` - The rotation angle (in degrees) of the label.
- `units` - Not used.

- `verticalalignment` - `{'top','bottom','middle'}` - Controls the alignment fo the text relative to the specified position point in the vertical position.
- `backgroundcolor` - `colorspec` - The color used to fill in the background rectangle for the label. Normally this is `none`.
- `edgecolor` - `colorspec` - The color used to draw the bounding rectangle for the label. Normally this is `none`.
- `linewidth` - `scalar` - The width of the line used to draw the border.
- `linestyle` - `{'-','--',':','-.','none'}` - The style of the line used to draw the border.
- `margin` - `scalar` - The amount of spacing to place around the text as padding when drawing the rectangle.
- `fontangle` - `{'normal','italic','oblique'}` - The angle of the fonts used for the labels.
- `fontsize` - `scalar` - The size of fonts used for the text.
- `fontunits` - Not used.
- `fontweight` - `{'normal','bold','light','demi'}` - The weight of the font used for the label
- `visible` - `{'on','off'}` - Controls visibility of the the line.
- `color` - `colorspec` - The color of the text of the label.
- `children` - Not used.
- `parent` - The handle of the axis that owns this label.
- `tag` - `string` - A string that can be used to tag the object.
- `type` - `string` - Returns the string `'text'`.
- `userdata` - `array` - Available to store any variable you want in the handle object.

23.66 TITLE Plot Title Function

23.66.1 Usage

This command adds a title to the plot. The general syntax for its use is

```
title('label')
```

or in the alternate form

```
title 'label'
```

or simply

```
title label
```

Here `label` is a string variable. You can also specify properties for the label, and a handle to serve as a target for the operation

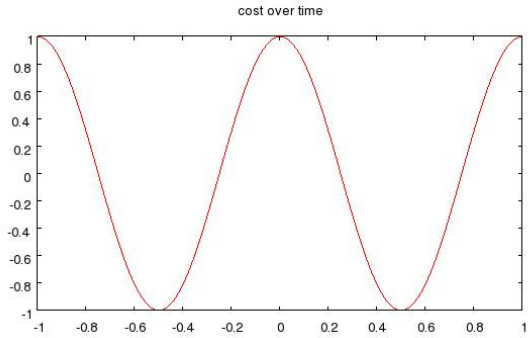
```
title(handle,'label',properties...)
```

23.66.2 Example

Here is an example of a simple plot with a title.

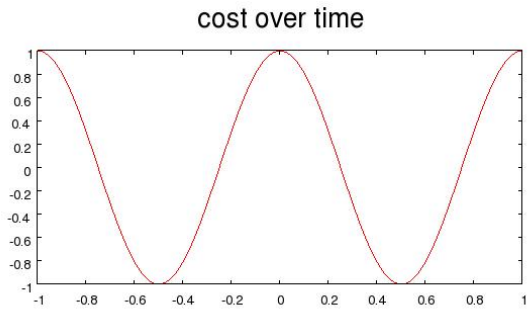
```
--> x = linspace(-1,1);
--> y = cos(2*pi*x);
--> plot(x,y,'r-');
--> title('cost over time');
```

which results in the following plot.



We now increase the size of the font using the properties of the `label`

```
--> title('cost over time','fontsize',20);
```



23.67 TUBEPLOT Creates a Tubeplot

23.67.1 Usage

This `tubeplot` function is from the tubeplot package written by Anders Sandberg. The simplest syntax for the `tubeplot` routine is

```
tubeplot(x,y,z)
```

plots the basic tube with radius 1, where `x,y,z` are vectors that describe the tube. If the radius of the tube is to be varied, use the second form

```
tubeplot(x,y,z,r)
```

which plots the basic tube with variable radius r (either a vector or a scalar value). The third form allows you to specify the coloring using a vector of values:

```
tubeplot(x,y,z,r,v)
```

where the coloring is now dependent on the values in the vector `v`. If you want to create a tube plot with a greater degree of tangential subdivisions (i.e., the tube is more circular, use the form

```
tubeplot(x,y,z,r,v,s)
```

where `s` is the number of tangential subdivisions (default is 6) You can also use `tubeplot` to calculate matrices to feed to `mesh` and `surf`.

```
[X,Y,Z]=tubeplot(x,y,z)
```

returns `N x 3` matrices suitable for mesh or surf.

Note that the tube may pinch at points where the normal and binormal misbehaves. It is suitable for general space curves, not ones that contain straight sections. Normally the tube is calculated using the Frenet frame, making the tube minimally twisted except at inflexion points.

To deal with this problem there is an alternative frame:

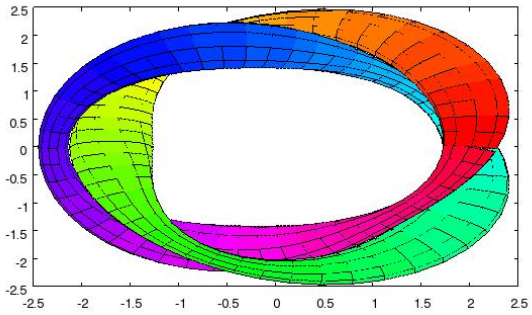
```
tubeplot(x,y,z,r,v,s,vec)
```

calculates the tube by setting the normal to the cross product of the tangent and the vector vec. If it is chosen so that it is always far from the tangent vector the frame will not twist unduly.

23.67.2 Example

Here is an example of a `tubeplot`.

```
--> t=0:(2*pi/100):(2*pi);
--> x=cos(t*2).*(2+sin(t*3)*.3);
--> y=sin(t*2).*(2+sin(t*3)*.3);
--> z=cos(t*3)*.3;
--> tubeplot(x,y,z,0.14*sin(t*5)+.29,t,10);
```



Written by Anders Sandberg, asa@nada.kth.se, 2005 Website says the package is free for anybody to use. www.nada.kth.se/ asa/Ray/Tubeplot/tubeplot.html

23.68 UICONTROL Create a UI Control object

23.68.1 Usage

Creates a UI control object and parents it to the current figure. The syntax for its use is

```
handle = uicontrol(property,value,property,value,...)
```

where `property` and `value` are set. The handle ID for the resulting object is returned. It is automatically added to the children of the current figure.

23.69 UICONTROLPROPERTIES UI Control Properties

23.69.1 Usage

Below is a summary of the properties for user interface controls.

- `backgroundcolor` - `colorspec` - The background color for the widget.
- `busyaction` - Not used.
- `buttondownfcn` - Not used.
- `callback` - `string` - the callback to execute when the GUI control does its action. Clicking a button or moving a scroller will cause the callback to be executed. Also, pressing enter in a text box causes the callback to be executed.
- `cdata` - an `M x N x 3` array that represents an RGB image to use as the truecolor image displayed on push bottons or toggle buttons. The values must be between 0 and 1.
- `children` - Not used.
- `createfcn` - Not used.
- `deletefcn` - Not used;
- `enable` - `{'on','inactive','off'}` - For `on` (the default) the uicontrol behaves normally. For inactive, it is not operational, but looks the same as `on`. For `off`, the control is grayed out.
- `extent` - a read only property that contains the extent of the text for the control.
- `fontangle` - `{'normal','italic','oblique'}` - The angle of the fonts used for text labels (e.g., tick labels).
- `fontsize` - `scalar` - The size of fonts used for text labels (tick labels).
- `fontunits` - Not used.
- `fontname` - `string` - The name of the font to use for the widget.
- `fontweight` - `{'normal','bold','light','demi'}` - The weight of the font used
- `foregroundcolor` - `colorspec` - the foreground color for text.
- `handlevisibility` - Not used.
- `hittest` - Not used.
- `horizontalalignment` - `{'left','center','right}` - determines the justification of text.
- `interruptible` - Not used.
- `keypressfcn` - `functionspec` - a string or function handle that is called when a key is pressed and a uicontrol object has focus.
- `listboxtop` - a scalar (used only by the listbox style of uicontrols) that specifies which string appears at the top of the list box.
- `max` - a scalar that specifies the largest value allowed for the `value` property. The interpretation varies depending on the type of the control
  - `check boxes` - specifies what `value` is set to when the check box is selected.
  - `edit box` - if `max-min>1` then the text box allows for multiple lines of input. Otherwise, it is a single line only.

  - `list box` - if `max-min>1` then multiple item selections are allowed. Otherwise, only single item selections are allowed.
  - `radio buttons` - specifies what `value` is set to when the radio button is selected.
  - `slider` - the maximum value the slider can take.
  - `toggle button` - specifies what `value` is set to when the toggle button is selected.
- `min` - a scalar that specifies the smallest value for the `value` property. The interpretation of it depends on the type of the control
  - `check boxes` - specifies what `value` is set to when the check box is not selected.
  - `edit box` - if `max-min>1` then the text box allows for multiple lines of input. Otherwise, it is a single line only.
  - `list box` - if `max-min>1` then multiple item selections are allowed. Otherwise, only single item selections are allowed.
  - `radio buttons` - specifies what `value` is set to when the radio button is not selected.
  - `slider` - the minimum value the slider can take.
  - `toggle button` - specifies what `value` is set to when the toggle button is not selected.
- `parent` - the handle of the parent object.
- `position` - size and location of the uicontrol as a four vector `[left, bottom, width, height]`. If `width>height` then sliders are horizontal, otherwise the slider is oriented vertically.
- `selected` - `{'on','off'}` - not used.
- `selectionhighlight` - `{'on','off'}` - not used.
- `sliderstep` - a two vector `[min\_step max\_step]` that controls the amount the slider `value` changes when you click the mouse on the control. If you click the arrow for the slider, the value changes by `min\_step`, while if you click the trough, the value changes by `max\_step`. Each value must be in the range `[0,1]`, and is a percentage of the range `max-min`.
- `string` - `string` - the text for the control.
- `style` - @—'pushbutton','toggle','radiobutton','checkbox', 'edit','text','slider','frame','listbox','popupmenu'—.
- `tag` - `string` - user specified label.
- `tooltipstring` - `string` the tooltip for the control.
- `type` - `string` - the text is set to `'uicontrol'`.
- `uicontextmenu` - `handle` the handle of the `uicontextmenu` that shows up when you right-click over the control.
- `units` - not used.
- `userdata` - `array` - any data you want to associate with the control.
- `value` - The meaning of this property depends on the type of the control:
  - check box - set to `max` when checked, and `min` when off.
  - list box - set to a vector of indices corresponding to selected items, with `1` corresponding to the first item in the list.
  - pop up menu - set to the index of the item selected (starting with 1)
  - radio buttons - set to `max` when selected, and set to `min` when not selected.

  - sliders - set to the value of the slider
  - toggle buttons - set to `max` when selected, and set to `min` when not selected.
  - text controls, push buttons - do not use this property.

- `visible` - `{'on','off'}` - controls whether the control is visible or not

23.70 VIEW Set Graphical View

23.70.1 Usage

The `view` function sets the view into the current plot. The simplest form is

```
view(n)
```

where `n=2` sets a standard view (azimuth 0 and elevation 90), and `n=3` sets a standard 3D view (azimuth 37.5 and elevation 30). With two arguments,

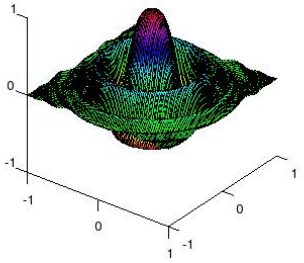
```
view(az,el)
```

you set the viewpoint to azimuth `az` and elevation `el`.

23.70.2 Example

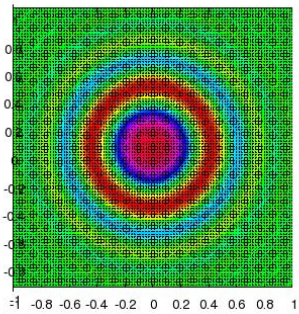
Here is a 3D surface plot shown with a number of viewpoints. First, the default view for a 3D plot.

```
--> x = repmat(linspace(-1,1),[100,1]);
--> y = x';
--> r = x.^2+y.^2;
--> z = exp(-r*3).*cos(5*pi*r);
--> surf(x,y,z);
--> axis equal
--> view(3)
```



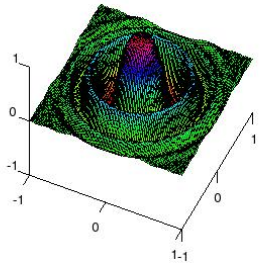
Next, we look at it as a 2D plot

```
--> surf(x,y,z);
--> axis equal
--> view(2)
```



Finally, we generate a different view of the same surface.

```
--> surf(x,y,z);
--> axis equal
--> view(25,50);
```



23.71 WINLEV Image Window-Level Function

23.71.1 Usage

Adjusts the data range used to map the current image to the current colormap. The general syntax for its use is

```
winlev(window,level)
```

where `window` is the new window, and `level` is the new level, or

```
winlev
```

in which case it returns a vector containing the current window and level for the active image.

23.71.2 Function Internals

FreeMat deals with scalar images on the range of `[0,1]`, and must therefor map an arbitrary image `x` to this range before it can be displayed. By default, the `image` command chooses

$$\text{window} = \max x - \min x,$$

and

$$\text{level} = \frac{\text{window}}{2}$$

This ensures that the entire range of image values in `x` are mapped to the screen. With the `winlev` function, you can change the range of values mapped. In general, before display, a pixel `x` is mapped to `[0,1]` via:

$$\max\left(0, \min\left(1, \frac{x - \text{level}}{\text{window}}\right)\right)$$

23.71.3 Examples

The window level function is fairly easy to demonstrate. Consider the following image, which is a Gaussian pulse image that is very narrow:

```
--> t = linspace(-1,1,256);
--> xmat = ones(256,1)*t; ymat = xmat';
--> A = exp(-(xmat.^2 + ymat.^2)*100);
--> image(A);
```

The data range of `A` is `[0,1]`, as we can verify numerically:

```
--> min(A(:))

ans =
 1.3839e-87

--> max(A(:))

ans =
    0.9969
```

To see the tail behavior, we use the `winlev` command to force FreeMat to map a smaller range of `A` to the colormap.

```
--> image(A);
--> winlev(1e-4,0.5e-4)
```

The result is a look at more of the tail behavior of `A`. We can also use the winlev function to find out what the window and level are once set, as in the following example.

```
--> image(A);
--> winlev(1e-4,0.5e-4)
--> winlev

ans =
 1.0000e-04
```

23.72 XLABEL Plot X-axis Label Function

23.72.1 Usage

This command adds a label to the x-axis of the plot. The general syntax for its use is

```
  xlabel('label')
```

or in the alternate form

```
  xlabel 'label'
```

or simply

```
  xlabel label
```

Here `label` is a string variable. You can also specify properties for that label using the syntax

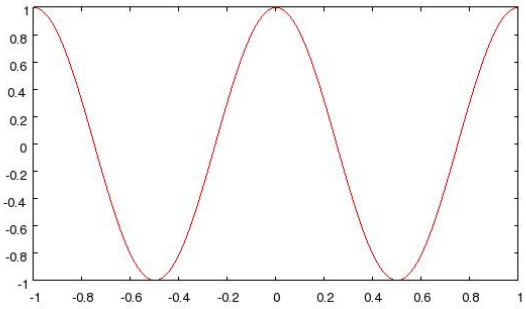
```
  xlabel('label',properties...)
```

23.72.2 Example

Here is an example of a simple plot with a label on the x-axis.

```
--> x = linspace(-1,1);
--> y = cos(2*pi*x);
--> plot(x,y,'r-');
--> xlabel('time');
```

which results in the following plot.



23.73 XLIM Adjust X Axis limits of plot

23.73.1 Usage

There are several ways to use `xlim` to adjust the X axis limits of a plot. The various syntaxes are

```
xlim
xlim([lo,hi])
xlim('auto')
xlim('manual')
xlim('mode')
xlim(handle,...)
```

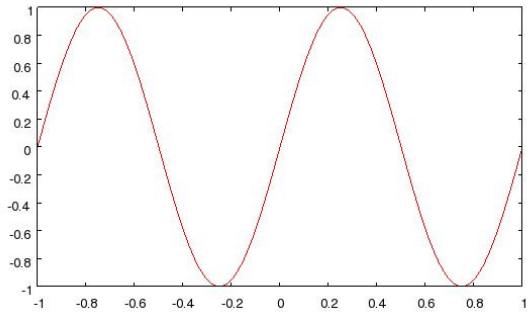
The first form (without arguments), returns a 2-vector containing the current limits. The second form sets the limits on the plot to `[lo,hi]`. The third and fourth form set the mode for the limit to `auto` and `manual` respectively. In `auto` mode, FreeMat chooses the range for the axis automatically. The `xlim('mode')` form returns the current mode for the axis (either `'auto'` or `'manual'`). Finally, you can specify the handle of an axis to manipulate instead of using the current one.

As an additional feature, you can now specify `inf` for a limit, and FreeMat will take that limit from the automatic set. So, for example `xlim([10,inf])` will set the minimum for the x axis, but use the automatic value for the maximum.

23.73.2 Example

```
--> x = linspace(-1,1);
--> y = sin(2*pi*x);
--> plot(x,y,'r-');
--> xlim  % what are the current limits?

ans =
 -1  1
```

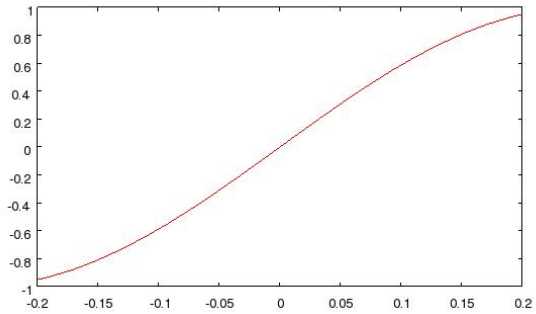
which results in



Next, we zoom in on the plot using the `xlim` function

```
--> plot(x,y,'r-')
--> xlim([-0.2,0.2])
```

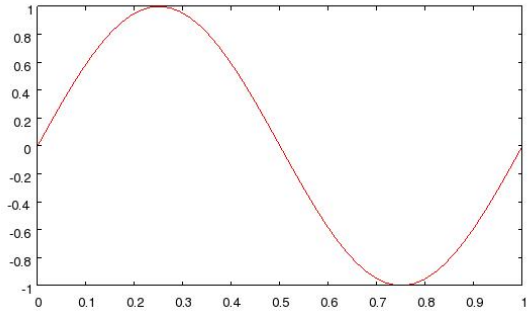
which results in



To demonstrate the infinite limits feature. Consider the following

```
--> plot(x,y,'r-');
--> xlim([0,inf])
```

which results in



23.74 YLABEL Plot Y-axis Label Function

23.74.1 Usage

This command adds a label to the y-axis of the plot. The general syntax for its use is

```
ylabel('label')
```

or in the alternate form

```
ylabel 'label'
```

or simply

```
ylabel label
```

You can also specify properties for that label using the syntax

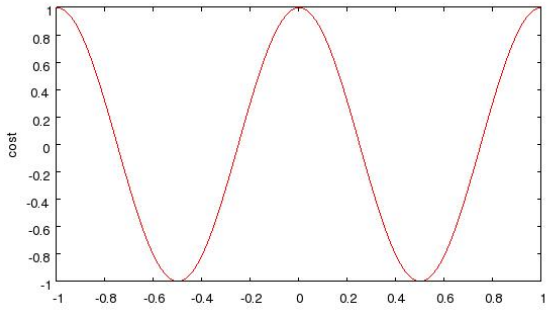
```
ylabel('label',properties...)
```

23.74.2 Example

Here is an example of a simple plot with a label on the y-axis.

```
--> x = linspace(-1,1);
--> y = cos(2*pi*x);
--> plot(x,y,'r-');
--> ylabel('cost');
```

which results in the following plot.



23.75 YLIM Adjust Y Axis limits of plot

23.75.1 Usage

There are several ways to use `ylim` to adjust the Y axis limits of a plot. The various syntaxes are

```
ylim
ylim([lo,hi])
ylim('auto')
ylim('manual')
ylim('mode')
ylim(handle,...)
```

The first form (without arguments), returns a 2-vector containing the current limits. The second form sets the limits on the plot to `[lo,hi]`. The third and fourth form set the mode for the limit to `auto` and `manual` respectively. In `auto` mode, FreeMat chooses the range for the axis automatically. The `ylim('mode')` form returns the current mode for the axis (either `'auto'` or `'manual'`). Finally, you can specify the handle of an axis to manipulate instead of using the current one.

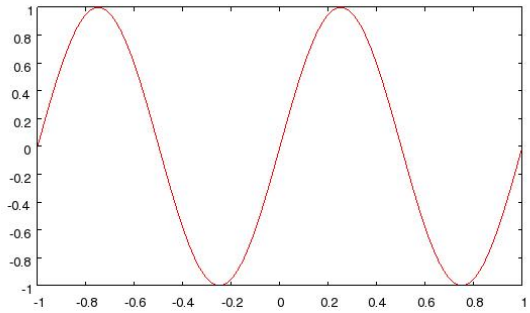
As an additional feature, you can now specify `inf` for a limit, and FreeMat will take that limit from the automatic set. So, for example `ylim([10,inf])` will set the minimum for the y axis, but use the automatic value for the maximum.

23.75.2 Example

```
--> x = linspace(-1,1);
--> y = sin(2*pi*x);
```

```
--> plot(x,y,'r-');
--> ylim  % what are the current limits?

ans =
   -0.9999    0.9999
```

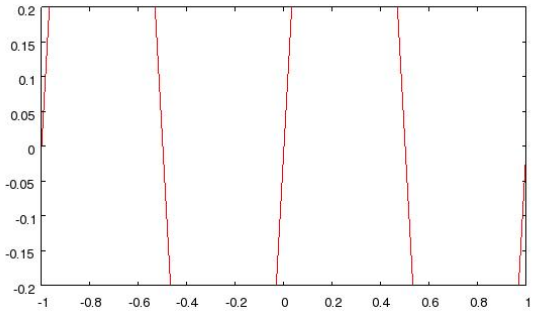
which results in



Next, we zoom in on the plot using the `ylim` function

```
--> plot(x,y,'r-')
--> ylim([-0.2,0.2])
```

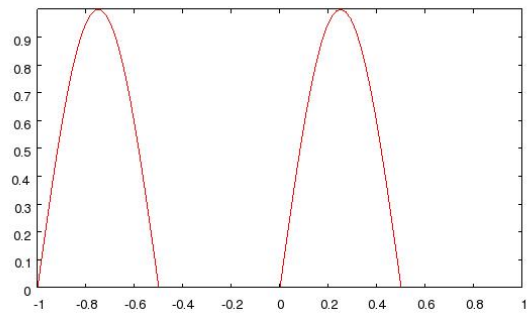
which results in



To demonstrate the infinite limits feature. Consider the following

```
--> plot(x,y,'r-');
--> ylim([0,inf])
```

which results in



23.76 ZLABEL Plot Z-axis Label Function

23.76.1 Usage

This command adds a label to the z-axis of the plot. The general syntax for its use is

```
zlabel('label')
```

or in the alternate form

```
zlabel 'label'
```

or simply

```
zlabel label
```

Here `label` is a string variable. You can also specify properties for that label using the syntax

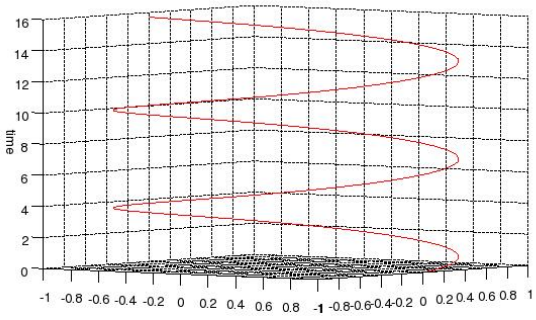
```
zlabel('label',properties...)
```

23.76.2 Example

Here is an example of a simple plot with a label on the `z`-axis.

```
--> t = linspace(0,5*pi);
--> x = cos(t);
--> y = sin(t);
--> z = t;
--> plot3(x,y,z,'r-');
--> view(3);
--> zlabel('time');
```

which results in the following plot.



23.77 ZLIM Adjust Z Axis limits of plot

23.77.1 Usage

There are several ways to use `zlim` to adjust the Z axis limits of a plot. The various syntaxes are

```
zlim
zlim([lo,hi])
zlim('auto')
zlim('manual')
zlim('mode')
zlim(handle,...)
```

The first form (without arguments), returns a 2-vector containing the current limits. The second form sets the limits on the plot to `[lo,hi]`. The third and fourth form set the mode for the limit to `auto` and `manual` respectively. In `auto` mode, FreeMat chooses the range for the axis automatically. The `zlim('mode')` form returns the current mode for the axis (either `'auto'` or `'manual'`). Finally, you can specify the handle of an axis to manipulate instead of using the current one.

23.78 ZOOM Image Zoom Function

23.78.1 Usage

This function changes the zoom factor associated with the currently active image. It is a legacy support function only, and thus is not quite equivalent to the `zoom` function from previous versions of FreeMat. However, it should achieve roughly the same effect. The generic syntax for its use is

```
zoom(x)
```

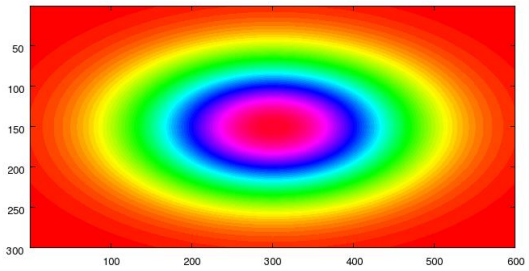
where `x` is the zoom factor to be used. The exact behavior of the zoom factor is as follows:

- `x>0` The image is zoomed by a factor `x` in both directions.
- `x=0` The image on display is zoomed to fit the size of the image window, but the aspect ratio of the image is not changed. (see the Examples section for more details). This is the default zoom level for images displayed with the `image` command.
- `x<0` The image on display is zoomed to fit the size of the image window, with the zoom factor in the row and column directions chosen to fill the entire window. The aspect ratio of the image is not preserved. The exact value of `x` is irrelevant.

23.78.2 Example

To demonstrate the use of the `zoom` function, we create a rectangular image of a Gaussian pulse. We start with a display of the image using the `image` command, and a zoom of 1.

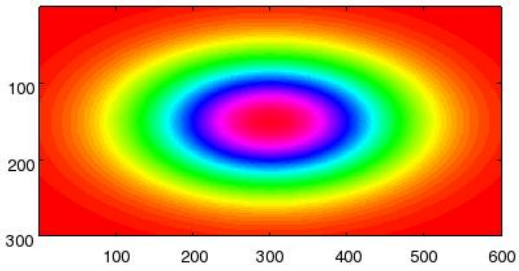
```
--> x = linspace(-1,1,300)'*ones(1,600);
--> y = ones(300,1)*linspace(-1,1,600);
--> Z = exp(-(x.^2+y.^2)/0.3);
--> image(Z);
--> zoom(1.0);
```



At this point, resizing the window accomplishes nothing, as with a zoom factor greater than zero, the size of the image is fixed.

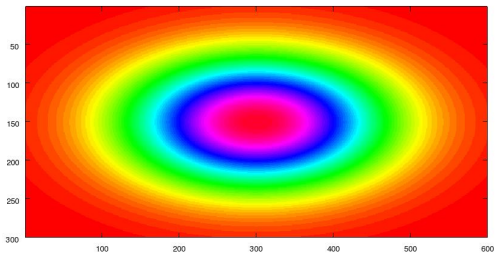
If we change the zoom to another factor larger than 1, we enlarge the image by the specified factor (or shrink it, for zoom factors `0 < x < 1`. Here is the same image zoomed out to 60

```
--> image(Z);
--> zoom(0.6);
```



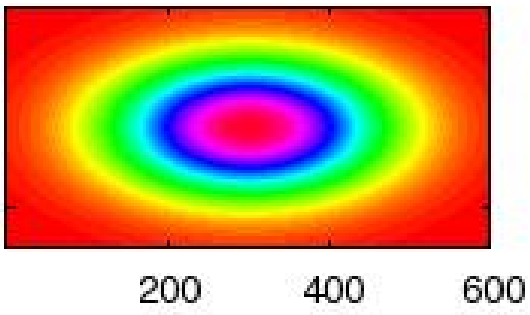
Similarly, we can enlarge it to 130

```
--> image(Z)
--> zoom(1.3);
```



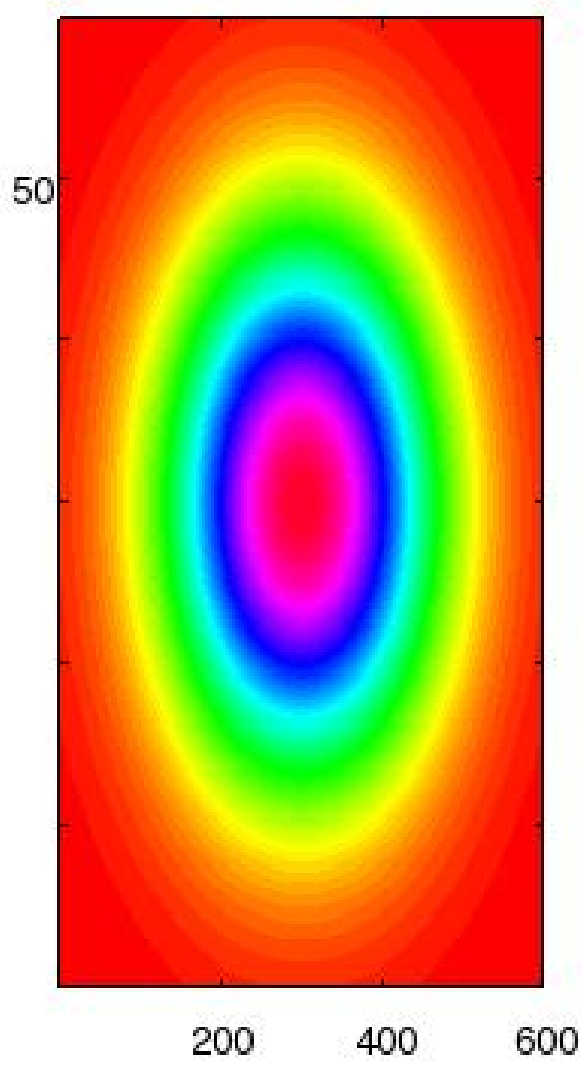
The "free" zoom of `x = 0` results in the image being zoomed to fit the window without changing the aspect ratio. The image is zoomed as much as possible in one direction.

```
--> image(Z);
--> zoom(0);
--> sizefig(200,400);
```



The case of a negative zoom `x < 0` results in the image being scaled arbitrarily. This allows the image aspect ratio to be changed, as in the following example.

```
--> image(Z);
--> zoom(-1);
--> sizefig(200,400);
```



23.79 ZPLANE Zero-pole plot

23.79.1 Usage

This function makes a zero-pole plot of a discrete-time system defined by its zeros and poles. The various syntaxes are

```
zplane(z,p)
```

where `z` and `p` are the zeros and the poles of the system stored as column vectors, or

```
zplane(b,a)
```

where `a` and `b` are the polynomial coefficients of the numerator and denominator stored as line vectors (`roots` is used to find the zeros and poles). The symbol `'o'` represents a zero and the symbol `'x'` represents a pole. The plot includes the unit circle for reference. Contributed by Paulo Xavier Candeias under GPL

Chapter 24

OpenGL Models

24.1 GLASSEMBLY Create a GL Assembly

24.1.1 Usage

Define a GL Assembly. A GL Assembly consists of one or more GL Nodes or GL Assemblies that are placed relative to the coordinate system of the assembly. For example, if we have `glnode` definitions for `'bread'` and `'cheese'`, then a `glassembly` of sandwich would consist of placements of two `'bread'` nodes with a `'cheese'` node in between. Furthermore, a `'lunch'` assembly could consist of a `'sandwich'` a `'chips'` and `'soda'`. Hopefully, you get the idea. The syntax for the `glassembly` command is

```
glassembly(name,part1,transform1,part2,transform2,...)
```

where `part1` is the name of the first part, and could be either a `glnode` or itself be another `glassembly`. Here `transform1` is the `4 x 4 matrix` that transforms the part into the local reference coordinate system.

WARNING!! Currently FreeMat does not detect or gracefully handle self-referential assemblies (i.e, if you try to make a `sandwich` contain a `sandwich`, which you can do by devious methods that I refuse to explain). Do not do this! You have been warned.

24.2 GLCLUMP Create a GL Clump

24.2.1 Usage

Defines an aggregate clump of objects that can be treated as a node. A GL Clump is defined by a vector consisting of the following elements:

```
[r1 g1 b1 n1 p1 p2 p3 ... r2 g2 b2 n2 p1 p2 p3 ... ]
```

i.e., an RGB color spec, followed by a point count `ni`, followed by a length `ni` vector of coordinates that are `x,y,z` triplets. The usage of this function is

```
glclump(name,vector)
```

where `name` is the name of the clump and `vector` is the aforementioned vector of points.

24.3 GLDEFMATERIAL Defines a GL Material

24.3.1 Usage

Define a material. The syntax for its use is

```
gldefmaterial(name,ambient,diffuse,specular,shininess)
```

where `name` is the name of the material, and `ambient` is a `4 x 1` vector containing the ambient component of the material property, and `diffuse` is a `4 x 1` vector and `specular` is a `4 x 1` vector containing the specular component of the material properties and `shininess` is the exponent that governs the shinines of the material.

24.4 GLLINES Create a GL Lineset

24.4.1 Usage

Defines a set of lines that can be treated as a node. A GL Lines is defined by a vector consisting of the following elements:

```
[m1 x1 y1 z1 ... xn yn zn m2 x1 y1 z1 .... ]
```

i.e., a point count followed by that number of triplets. The usage of this function is

```
gllines(name,vector,color)
```

where `name` is the name of the lineset and `vector` is the aforementioned vector of points.

24.5 GLNODE Create a GL Node

24.5.1 Usage

Define a GL Node. A GL Node is an object that can be displayed in a GL Window. It is defined by a triangular mesh of vertices. It must also have a material that defines its appearance (i.e. color, shininess, etc.). The syntax for the `glnode` command is

```
glnode(name,material,pointset)
```

where `material` is the name of a material that has already been defined with `gldefmaterial`, `pointset` is a `3 x N` matrix of points that define the geometry of the object. Note that the points are assumed to be connected in triangular facts, with the points defined counter clock-wise as seen from the outside of the facet. `FreeMat` will compute the normals. The `name` argument must be unique. If you want multiple instances of a given `glnode` in your GLWindow, that is fine, as instances of a `glnode` are created through a `glassembly`.

Chapter 25

Object Oriented Programming

25.1 AND Overloaded Logical And Operator

25.1.1 Usage

This is a method that is invoked to combine two variables using a logical and operator, and is invoked when you call

```
c = and(a,b)
```

or for

```
c = a & b
```

25.2 CAT Concatenation of Arrays

25.2.1 Usage

This function concatenates arrays in a given dimension. The syntax for its use is

```
cat (DIM, A, B)
cat (DIM, A, B, C ...)
```

to return the concatenation along the dimension `DIM` of all arguments. `cat(1, A, B, C)` is the same as `[A; B; C]` or `vertcat(A, B, C)`. `cat(2, A, B, C)` is the same as `[A, B, C]` or `horzcat(A, B, C)`.

25.3 CLASS Class Support Function

25.3.1 Usage

There are several uses for the `class` function. The first version takes a single argument, and returns the class of that variable. The syntax for this form is

```
classname = class(variable)
```

and it returns a string containing the name of the class for `variable`. The second form of the class function is used to construct an object of a specific type based on a structure which contains data elements for the class. The syntax for this version is

```
classvar = class(template, classname, parent1, parent2,...)
```

This should be called inside the constructor for the class. The resulting class will be of the type `classname`, and will be derived from `parent1`, `parent2`, etc. The `template` argument should be a structure array that contains the members of the class. See the `constructors` help for some details on how to use the `class` function. Note that if the `template` argument is an empty structure matrix, then the resulting variable has no fields beyond those inherited from the parent classes.

25.4 COLON Overloaded Colon Operator

25.4.1 Usage

This is a method that is invoked in one of two forms, either the two argument version

```
c = colon(a,b)
```

which is also called using the notation

```
c = a:b
```

and the three argument version

```
d = colon(a,b,c)
```

which is also called using the notation

```
d = a:b:c
```

25.5 CONSTRUCTORS Class Constructors

25.5.1 Usage

When designing a constructor for a FreeMat class, you should design the constructor to take a certain form. The following is the code for the sample `mat` object

```
function p = mat(a)
  if (nargin == 0)
    p.c = [];
    p = class(p,'mat');
  elseif isa(a,'mat')
    p = a;
  else
    p.c = a;
    p = class(p,'mat');
  end
```

Generally speaking when it is provided with zero arguments, the constructor returns a default version of the class using a template structure with the right fields populated with default values. If the constructor is given a single argument that matches the class we are trying to construct, the constructor passes through the argument. This form of the constructor is used for type conversion. In particular,

```
p = mat(a)
```

guarantees that `p` is an array of class `mat`. The last form of the constructor builds a class object given the input. The meaning of this form depends on what makes sense for your class. For example, for a polynomial class, you may want to pass in the coefficients of the polynomial.

25.6 CTRANSPOSE Overloaded Conjugate Transpose Operator

25.6.1 Usage

This is a method that is invoked when a variable has the conjugate transpose operator method applied, and is invoked when you call

```
c = ctranspose(a)
```

or

```
/  c = a'
```

25.7 EQ Overloaded Equals Comparison Operator

25.7.1 Usage

This is a method that is invoked to combine two variables using an equals comparison operator, and is invoked when you call

```
c = eq(a,b)
```

or for

```
c = a == b
```

25.8 GE Overloaded Greater-Than-Equals Comparison Operator

25.8.1 Usage

This is a method that is invoked to combine two variables using a greater than or equals comparison operator, and is invoked when you call

```
c = ge(a,b)
```

or for

```
c = a >= b
```

25.9 GT Overloaded Greater Than Comparison Operator

25.9.1 Usage

This is a method that is invoked to combine two variables using a greater than comparison operator, and is invoked when you call

```
c = gt(a,b)
```

or for

```
c = a > b
```

25.10 HORZCAT Overloaded Horizontal Concatenation

25.10.1 Usage

This is a method for a class that is invoked to concatenate two or more variables of the same class type together. Besides being called when you invoke

```
c = horzcat(a,b,c)
```

when `a` is a class, it is also called for

```
c = [a,b,c]
```

when one of these variables is a class. The exact meaning of horizontal concatenation depends on the class you have designed.

25.11 LDIVIDE Overloaded Left Divide Operator

25.11.1 Usage

This is a method that is invoked when two variables are divided and is invoked when you call

```
c = ldivide(a,b)
```

or for

```
c = a .\ b
```

25.12 LE Overloaded Less-Than-Equals Comparison Operator

25.12.1 Usage

This is a method that is invoked to compare two variables using a less than or equals comparison operator, and is invoked when you call

```
c = le(a,b)
```

or for

```
c = a <= b
```

25.13 LT Overloaded Less Than Comparison Operator

25.13.1 Usage

This is a method that is invoked to compare two variables using a less than comparison operator, and is invoked when you call

```
c = lt(a,b)
```

or for

```
c = a < b
```

25.14 MINUS Overloaded Addition Operator

25.14.1 Usage

This is a method that is invoked when two variables are subtracted and is invoked when you call

```
c = minus(a,b)
```

or for

```
c = a - b
```

25.15 MLDIVIDE Overloaded Matrix Left Divide Operator

25.15.1 Usage

This is a method that is invoked when two variables are divided using the matrix (left) divide operator, and is invoked when you call

```
c = mldivide(a,b)
```

or for

```
c = a \ b
```

25.16 MPOWER Overloaded Matrix Power Operator

25.16.1 Usage

This is a method that is invoked when one variable is raised to another variable using the matrix power operator, and is invoked when you call

```
c = mpower(a,b)
```

or

```
c = a^b
```

25.17 MRDIVIDE Overloaded Matrix Right Divide Operator

25.17.1 Usage

This is a method that is invoked when two variables are divided using the matrix divide operator, and is invoked when you call

```
c = mrdivide(a,b)
```

or for

```
c = a / b
```

25.18 MTIMES Overloaded Matrix Multiplication Operator

25.18.1 Usage

This is a method that is invoked when two variables are multiplied using the matrix operator and is invoked when you call

```
c = mtimes(a,b)
```

or for

```
c = a * b
```

25.19 NE Overloaded Not-Equals Comparison Operator

25.19.1 Usage

This is a method that is invoked to combine two variables using a not-equals comparison operator, and is invoked when you call

```
c = ne(a,b)
```

or for

```
c = a != b
```

25.20 NOT Overloaded Logical Not Operator

25.20.1 Usage

This is a method that is invoked when a variable is logically inverted, and is invoked when you call

```
c = not(a)
```

or for

```
c = ~a
```

25.21 OR Overloaded Logical Or Operator

25.21.1 Usage

This is a method that is invoked to combine two variables using a logical or operator, and is invoked when you call

```
c = or(a,b)
```

or for

```
c = a | b
```

25.22 PLUS Overloaded Addition Operator

25.22.1 Usage

This is a method that is invoked when two variables are added and is invoked when you call

```
c = plus(a,b)
```

or for

```
c = a + b
```

25.23 POWER Overloaded Power Operator

25.23.1 Usage

This is a method that is invoked when one variable is raised to another variable using the dot-power operator, and is invoked when you call

```
c = power(a,b)
```

or

```
c = a.^b
```

25.24 RDIVIDE Overloaded Right Divide Operator

25.24.1 Usage

This is a method that is invoked when two variables are divided and is invoked when you call

```
c = rdivide(a,b)
```

or for

```
c = a ./ b
```

25.25 SUBSASGN Overloaded Class Assignment

25.25.1 Usage

This method is called for expressions of the form

```
a(b) = c, a{b} = c, a.b = c
```

and overloading the `subsasgn` method can allow you to define the meaning of these expressions for objects of class `a`. These expressions are mapped to a call of the form

```
a = subsasgn(a,s,b)
```

where `s` is a structure array with two fields. The first field is

- `type` is a string containing either `'()'` or `'{}'` or `'.'` depending on the form of the call.
- `subs` is a cell array or string containing the the subscript information.

When multiple indexing experssions are combined together such as `a(5).foo{:} = b`, the `s` array contains the following entries

```
s(1).type = '()'  s(1).subs = {5}
s(2).type = '.'   s(2).subs = 'foo'
s(3).type = '{}'  s(3).subs = ':'
```

25.26 SUBSINDEX Overloaded Class Indexing

25.26.1 Usage

This method is called for classes in the expressions of the form

```
c = subsindex(a)
```

where `a` is an object, and `c` is an index vector. It is also called for

```
c = b(a)
```

in which case `subsindex(a)` must return a vector containing integers between `0` and `N-1` where `N` is the number of elements in the vector `b`.

25.27 SUBSREF Overloaded Class Indexing

25.27.1 Usage

This method is called for expressions of the form

```
c = a(b), c = a{b}, c = a.b
```

and overloading the `subsref` method allows you to define the meaning of these expressions for objects of class `a`. These expressions are mapped to a call of the form

```
b = subsref(a,s)
```

where `s` is a structure array with two fields. The first field is

- `type` is a string containing either `'()'` or `'{}'` or `'.'` depending on the form of the call.
- `subs` is a cell array or string containing the the subscript information.

When multiple indexing experssions are combined together such as `b = a(5).foo{:}`, the `s` array contains the following entries

```
s(1).type = '()'  s(1).subs = {5}
s(2).type = '.'   s(2).subs = 'foo'
s(3).type = '{}'  s(3).subs = ':'
```

25.28 TIMES Overloaded Multiplication Operator

25.28.1 Usage

This is a method that is invoked when two variables are multiplied and is invoked when you call

```
c = times(a,b)
```

or for

```
c = a .* b
```

25.29 TRANSPOSE Overloaded Transpose Operator

25.29.1 Usage

This is a method that is invoked when a variable has the transpose operator method applied, and is invoked when you call

```
c = transpose(a)
```

or

```
/  c = a.'
```

25.30 UMINUS Overloaded Unary Minus Operator

25.30.1 Usage

This is a method that is invoked when a variable is negated, and is invoked when you call

```
c = uminus(a)
```

or for

```
c = -a
```

25.31 UPLUS Overloaded Unary Plus Operator

25.31.1 Usage

This is a method that is invoked when a variable is preceeded by a "+", and is invoked when you call

```
c = uplus(a)
```

or for

```
c = +a
```

25.32 VERTCAT Overloaded Vertical Concatenation

25.32.1 Usage

This is a method for a class that is invoked to concatenate two or more variables of the same class type together. Besides being called when you invoke

```
c = vertcat(a,b,c)
```

when `a` is a class, it is also called for

```
c = [a;b;c]
```

when one of the variables is a class. The exact meaning of vertical concatenation depends on the class you have designed.

Chapter 26

Bitwise Operations

26.1 BITAND Bitwise Boolean And Operation

26.1.1 Usage

Performs a bitwise binary and operation on the two arguments and returns the result. The syntax for its use is

```
y = bitand(a,b)
```

where `a` and `b` are multi-dimensional unsigned integer arrays. The and operation is performed using 32 bit unsigned intermediates. Note that if a or b is a scalar, then each element of the other array is and'ed with that scalar. Otherwise the two arrays must match in size.

26.1.2 Example

```
--> bitand(uint16([1,16,255]),uint16([3,17,128]))

ans =
   1  16 128

--> bitand(uint16([1,16,255]),uint16(3))

ans =
 1 0 3
```

26.2 BITCMP Bitwise Boolean Complement Operation

26.2.1 Usage

Usage

Performs a bitwise binary complement operation on the argument and returns the result. The syntax for its use is

```
y = bitcmp(a)
```

where a is an unsigned integer arrays. This version of the command uses as many bits as required by the type of a. For example, if a is an uint8 type, then the complement is formed using 8 bits. The second form of bitcmp allows you to specify the number of bits to use,

```
y = bitcmp(a,n)
```

in which case the complement is taken with respect to n bits, where n must be less than the length of the integer type.

26.2.2 Example

```
--> bitcmp(uint16(2^14-2))

ans =
 49153

--> bitcmp(uint16(2^14-2),14)

ans =
 1
```

26.3 BITOR Bitwise Boolean Or Operation

26.3.1 Usage

Performs a bitwise binary or operation on the two arguments and returns the result. The syntax for its use is

```
y = bitor(a,b)
```

where `a` and `b` are multi-dimensional unsigned integer arrays. The and operation is performed using 32 bit unsigned intermediates. Note that if a or b is a scalar, then each element of the other array is or'ed with that scalar. Otherwise the two arrays must match in size.

26.3.2 Example

```
--> bitand(uint16([1,16,255]),uint16([3,17,128]))

ans =
   1  16 128

--> bitand(uint16([1,16,255]),uint16(3))

ans =
 1 0 3
```

26.4 BITXOR Bitwise Boolean Exclusive-Or (XOR) Operation

26.4.1 Usage

Performs a bitwise binary xor operation on the two arguments and returns the result. The syntax for its use is

```
y = bitxor(a,b)
```

where `a` and `b` are multi-dimensional unsigned integer arrays. The and operation is performed using 32 bit unsigned intermediates. Note that if a or b is a scalar, then each element of the other array is xor'ed with that scalar. Otherwise the two arrays must match in size.

26.4.2 Example

```
--> bitand(uint16([1,16,255]),uint16([3,17,128]))

ans =
   1  16 128
```

```
--> bitand(uint16([1,16,255]),uint16(3))

ans =
 1 0 3
```

Chapter 27

FreeMat Threads

27.1 THREADCALL Call Function In A Thread

27.1.1 Usage

The `threadcall` function is a convenience function for executing a function call in a thread. The syntax for its use is

```
[val1,...,valn] = threadcall(threadid,timeout,funcname,arg1,arg2,...)
```

where `threadid` is the ID of the thread (as returned by the `threadnew` function), `funcname` is the name of the function to call, and `argi` are the arguments to the function, and `timeout` is the amount of time (in milliseconds) that the function is allowed to take.

27.1.2 Example

Here is an example of executing a simple function in a different thread.

```
--> id = threadnew

id =
 3

--> d = threadcall(id,1000,'cos',1.02343)

d =
    0.5204

--> threadfree(id)
```

27.2 THREADFREE Free thread resources

27.2.1 Usage

The `threadfree` is a function to free the resources claimed by a thread that has finished. The syntax for its use is

```
threadfree(handle)
```

where `handle` is the handle returned by the call to `threadnew`. The `threadfree` function requires that the thread be completed. Otherwise it will wait for the thread to complete, potentially for an arbitrarily long period of time. To fix this, you can either call `threadfree` only on threads that are known to have completed, or you can call it using the syntax

```
threadfree(handle,timeout)
```

where `timeout` is a time to wait in milliseconds. If the thread fails to complete before the timeout expires, an error occurs.

27.3 THREADID Get Current Thread Handle

27.3.1 Usage

The `threadid` function in FreeMat tells you which thread is executing the context you are in. Normally, this is thread 1, the main thread. However, if you start a new thread using `threadnew`, you will be operating in a new thread, and functions that call `threadid` from the new thread will return their handles.

27.3.2 Example

From the main thread, we have

```
--> threadid

ans =
 2
```

But from a launched auxilliary thread, we have

```
--> t_id = threadnew

t_id =
 3

--> id = threadcall(t_id,1000,'threadid')

id =
 3

--> threadfree(t_id);
```

27.4 THREADKILL Halt execution of a thread

27.4.1 Usage

The `threadkill` function stops (or attempts to stop) execution of the given thread. It works only for functions defined in M-files (i.e., not for built in or imported functions), and it works by setting a flag that causes the thread to stop execution at the next available statement. The syntax for this function is

```
threadkill(handle)
```

where `handle` is the value returned by a `threadnew` call. Note that the `threadkill` function returns immediately. It is still your responsibility to call `threadfree` to free the thread you have halted.

You cannot kill the main thread (thread id `1`).

27.4.2 Example

Here is an example of stopping a runaway thread using `threadkill`. Note that the thread function in this case is an M-file function. We start by setting up a free running counter, where we can access the counter from the global variables.

```
    freecount.m
function freecount
  global count
  if (~exist('count')) count = 0; end  % Initialize the counter
  while (1)
    count = count + 1;                 % Update the counter
  end
```

We now launch this function in a thread, and use `threadkill` to stop it:

```
--> a = threadnew;
--> global count                    % register the global variable count
--> count = 0;
--> threadstart(a,'freecount',0)    % start the thread
--> count                           % it is counting

ans =
 70

--> sleep(1)                        % Wait a bit
--> count                           % it is still counting

ans =
 545650

--> threadkill(a)                   % kill the counter
--> threadwait(a,1000)              % wait for it to finish

ans =
 1

--> count                           % The count will no longer increase

ans =
 545729

--> sleep(1)
--> count

ans =
 545729

--> threadfree(a)
```

27.5 THREADNEW Create a New Thread

27.5.1 Usage

The `threadnew` function creates a new FreeMat thread, and returns a handle to the resulting thread. The `threadnew` function takes no arguments. They general syntax for the `threadnew` function is

```
handle = threadnew
```

Once you have a handle to a thread, you can start the thread on a computation using the `threadstart` function. The threads returned by `threadnew` are in a dormant state (i.e., not running). Once you are finished with the thread you must call `threadfree` to free the resources associated with that thread.

Some additional important information. Thread functions operate in their own context or workspace, which means that data cannot be shared between threads. The exception is `global` variables, which provide a thread-safe way for multiple threads to share data. Accesses to global variables are serialized so that they can be used to share data. Threads and FreeMat are a new feature, so there is room for improvement in the API and behavior. The best way to improve threads is to experiment with them, and send feedback.

27.6 THREADSTART Start a New Thread Computation

27.6.1 Usage

The `threadstart` function starts a new computation on a FreeMat thread, and you must provide a function (no scripts are allowed) to run inside the thread, pass any parameters that the thread function requires, as well as the number of output arguments expected. The general syntax for the `threadstart` function is

```
threadstart(threadid,function,nargout,arg1,arg2,...)
```

where `threadid` is a thread handle (returned by `threadnew`), where `function` is a valid function name (it can be a built-in imported or M-function), `nargout` is the number of output arguments expected from the function, and `arg1` is the first argument that is passed to the function. Because the function runs in its own thread, the return values of the function are not available imediately. Instead, execution of that function will continue in parallel with the current thread. To retrieve the output of the thread function, you must wait for the thread to complete using the `threadwait` function, and then call `threadvalue` to retrieve the result. You can also stop the running thread prematurely by using the `threadkill` function. It is important to call `threadfree` on the handle you get from `threadnew` when you are finished with the thread to ensure that the resoures are properly freed.

It is also perfectly reasonable to use a single thread multiple times, calling `threadstart` and `threadreturn` multiple times on a single thread. The context is preserved between threads. When calling `threadstart` on a pre-existing thread, FreeMat will attempt to wait on the thread. If the wait fails, then an error will occur.

Some additional important information. Thread functions operate in their own context or workspace, which means that data cannot be shared between threads. The exception is `global` variables, which provide a thread-safe way for multiple threads to share data. Accesses to global variables are serialized so that they can be used to share data. Threads and FreeMat are a new feature, so there is room for improvement in the API and behavior. The best way to improve threads is to experiment with them, and send feedback.

27.6.2 Example

Here we do something very simple. We want to obtain a listing of all files on the system, but do not want the results to stop our computation. So we run the `system` call in a thread.

```
--> a = threadnew;                          % Create the thread
--> threadstart(a,'system',1,'ls -lrt /'); % Start the thread
--> b = rand(100)\rand(100,1);              % Solve some equations simultaneously
--> c = threadvalue(a);                     % Retrieve the file list
--> size(c)                                 % It is large!

ans =
  1 25

--> threadfree(a);
```

The possibilities for threads are significant. For example, we can solve equations in parallel, or take Fast Fourier Transforms on multiple threads. On multi-processor machines or multicore CPUs, these threaded calculations will execute in parallel. Neat.

The reason for the `nargout` argument is best illustrated with an example. Suppose we want to compute the Singular Value Decomposition `svd` of a matrix `A` in a thread. The documentation for the `svd` function

tells us that the behavior depends on the number of output arguments we request. For example, if we want a full decomposition, including the left and right singular vectors, and a diagonal singular matrix, we need to use the three-output syntax, instead of the single output syntax (which returns only the singular values in a column vector):

```
--> A = float(rand(4))

A =
    0.4011    0.4747    0.9193    0.8655
    0.8633    0.0123    0.0599    0.5917
    0.4939    0.5458    0.9481    0.4566
    0.9335    0.8614    0.7993    0.6394

--> [u,s,v] = svd(A)    % Compute the full decomposition
u =
   -0.5290    0.2711    0.7178    0.3626
   -0.3004   -0.8911    0.2379   -0.2431
   -0.4868    0.3556   -0.0927   -0.7925
   -0.6269   -0.0778   -0.6478    0.4259

s =
    2.5579         0         0         0
         0    0.7905         0         0
         0         0    0.4392         0
         0         0         0    0.1705

v =
   -0.5071   -0.7054   -0.3579   -0.3422
   -0.4146    0.3096   -0.6032    0.6070
   -0.5735    0.5955    0.1558   -0.5406
   -0.4921   -0.2279    0.6955    0.4713

--> sigmas = svd(A)     % Only want the singular values

sigmas =
    2.5579
    0.7905
    0.4392
    0.1705
```

Normally, FreeMat uses the left hand side of an assignment to calculate the number of outputs for the function. When running a function in a thread, we separate the assignment of the output from the invokation of the function. Hence, we have to provide the number of arguments at the time we invoke the function. For example, to compute a full decomposition in a thread, we specify that we want 3 output arguments:

```
--> a = threadnew;               % Create the thread
--> threadstart(a,'svd',3,A);    % Start a full decomposition
--> [u1,s1,v1] = threadvalue(a); % Retrieve the function values
--> threadfree(a);
```

If we want to compute just the singular values, we start the thread function with only one output argument:

```
--> a = threadnew;
--> threadstart(a,'svd',1,A);
--> sigmas = threadvalue(a);
--> threadfree(a)
```

27.7 THREADVALUE Retrieve the return values from a thread

27.7.1 Usage

The `threadvalue` function retrieves the values returned by the function specified in the `threadnew` call. The syntax for its use is

```
[arg1,arg2,...,argN] = threadvalue(handle)
```

where `handle` is the value returned by a `threadnew` call. Note that there are issues with `nargout`. See the examples section of `threadnew` for details on how to work around this limitation. Because the function you have spawned with `threadnew` may still be executing, `threadvalue` must first `threadwait` for the function to complete before retrieving the output values. This wait may take an arbitrarily long time if the thread function is caught in an infinite loop. Hence, you can also specify a timeout parameter to `threadvalue` as

```
[arg1,arg2,...,argN] = threadvalue(handle,timeout)
```

where the `timeout` is specified in milliseconds. If the wait times out, an error is raised (that can be caught with a `try` and `catch` block.

In either case, if the thread function itself caused an error and ceased execution abruptly, then calling `threadvalue` will cause that function to raise an error, allowing you to retrieve the error that was caused and correct it. See the examples section for more information.

27.7.2 Example

Here we do something very simple. We want to obtain a listing of all files on the system, but do not want the results to stop our computation. So we run the `system` call in a thread.

```
--> a = threadnew;                          % Create the thread
--> threadstart(a,'system',1,'ls -lrt /'); % Start the thread
--> b = rand(100)\rand(100,1);              % Solve some equations simultaneously
--> c = threadvalue(a);                     % Retrieve the file list
--> size(c)                                 % It is large!

ans =
  1 25

--> threadfree(a);
```

In this example, we force the threaded function to cause an exception (by calling the `error` function as the thread function). When we call `threadvalue`, we get an error, instead of the return value of the function

```
--> a = threadnew

a =
 3

--> threadstart(a,'error',0,'Hello world!'); % Will immediately stop due to error
--> c = threadvalue(a)                      % The error comes to us
Error: Thread: Hello world!
--> threadfree(a)
```

Note that the error has the text `Thread:` prepended to the message to help you identify that this was an error in a different thread.

27.8 THREADWAIT Wait on a thread to complete execution

27.8.1 Usage

The `threadwait` function waits for the given thread to complete execution, and stops execution of the current thread (the one calling `threadwait`) until the given thread completes. The syntax for its use is

```
success = threadwait(handle)
```

where `handle` is the value returned by `threadnew` and `success` is a `logical` vaariable that will be `1` if the wait was successful or `0` if the wait times out. By default, the wait is indefinite. It is better to use the following form of the function

```
success = threadwait(handle,timeout)
```

where `timeout` is the amount of time (in milliseconds) for the `threadwait` function to wait before a timeout occurs. If the `threadwait` function succeeds, then the return value is a logical `1`, and if it fails, the return value is a logical `0`. Note that you can call `threadwait` multiple times on a thread, and if the thread is completed, each one will succeed.

27.8.2 Example

Here we lauch the `sleep` function in a thread with a time delay of 10 seconds. This means that the thread function will not complete until 10 seconds have elapsed. When we call `threadwait` on this thread with a short timeout, it fails, but not when the timeout is long enough to capture the end of the function call.

```
--> a = threadnew;
--> threadstart(a,'sleep',0,10);  % start a thread that will sleep for 10
--> threadwait(a,2000)            % 2 second wait is not long enough

ans =
 0

--> threadwait(a,10000)           % 10 second wait is long enough

ans =
 1

--> threadfree(a)
```

Chapter 28

Function Related Functions

28.1 INLINE Construct Inline Function

28.1.1 Usage

Constructs an inline function object. The syntax for its use is either

```
y = inline(expr)
```

which uses the `symvar` function to identify the variables in the expression, or the explicit form

```
y = inline(expr,var1,var2,...,varn)
```

where the variables are explicitly given. Note that inline functions are only partially supported in FreeMat. If you need features of the inline function that are not currently implemented, please file a feature request at the FreeMat website.

28.1.2 Example

Here we construct an inline expression using the autodetection of `symvar`

```
--> a = inline('x^2')

a =
  inline function object
  f(x) = x^2
--> a(3)

ans =
 9

--> a(i)

ans =
  -1.0000 +  0.0000i
```

In this case, we have multiple arguments (again, autodetected)

```
--> a = inline('x+y-cos(x+y)')

a =
  inline function object
  f(x,y) = x+y-cos(x+y)
```

```
--> a(pi,-pi)

ans =
 -1
```

In this form, we specify which arguments we want to use (thereby also specifying the order of the arguments

```
--> a = inline('x+t-sin(x)','x','t')

a =
  inline function object
  f(x,t) = x+t-sin(x)
--> a(0.5,1)

ans =
    1.0206
```

Inline objects can also be used with `feval`

```
--> a = inline('cos(t)')

a =
  inline function object
  f(t) = cos(t)
--> feval(a,pi/2)

ans =
 6.1232e-17
```

28.2 SYMVAR Find Symbolic Variables in an Expression

28.2.1 Usage

Finds the symbolic variables in an expression. The syntax for its use is

```
  syms = symvar(expr)
```

where `expr` is a string containing an expression, such as `'x\^2 + cos(t+alpha)'`. The result is a cell array of strings containing the non-function identifiers in the expression. Because they are usually not used as identifiers in expressions, the strings `'pi','inf','nan','eps','i','j'` are ignored.

28.2.2 Example

Here are some simple examples:

```
--> symvar('x^2+sqrt(x)')  % sqrt is eliminated as a function

ans =
 [x]

--> symvar('pi+3')         % No identifiers here

ans =
  Empty array 0x1
--> symvar('x + t*alpha')  % x, t and alpha
```

```
ans =
 [alpha]
 [t]
 [x]
```

Chapter 29

FreeMat External Interface

29.1 CENUM Lookup Enumerated C Type

29.1.1 Usage

The `cenum` function allows you to use the textual strings of C enumerated types (that have been defined using `ctypedefine`) in your FreeMat scripts isntead of the hardcoded numerical values. The general syntax for its use is

```
enum_int = cenum(enum_type,enum_string)
```

which looks up the integer value of the enumerated type based on the string. You can also supply an integer argument, in which case `cenum` will find the matching string

```
enum_string = cenum(enum_type,enum_int)
```

29.2 CTYPECAST Cast FreeMat Structure to C Structure

29.2.1 Usage

The `ctypecast` function is a convenience function for ensuring that a FreeMat structure fits the definition of a C struct (as defined via `ctypedefine`. It does so by encoding the structure to a byte array using `ctypefreeze` and then recovering it using the `ctypethaw` function. The usage is simply

```
s = ctypecast(s,typename)
```

where `s` is the structure and `typename` is the name of the C structure that describes the desired layout and types for elements of `s`. This function is equivalent to calling `ctypefreeze` and `ctypethaw` in succession on a FreeMat structure.

29.3 CTYPEDEFINE Define C Type

29.3.1 Usage

The `ctypedefine` function allows you to define C types for use with FreeMat. Three variants of C types can be used. You can use structures, enumerations, and aliases (typedefs). All three are defined through a single function `ctypedefine`. The general syntax for its use is

```
ctypedefine(typeclass,typename,...)
```

where `typeclass` is the variant of the type (legal values are `'struct'`, `'alias'`, `'enum'`). The second argument is the name of the C type. The remaining arguments depend on what the class of the typedef is.

To define a C structure, use the `'struct'` type class. The usage in this case is:

```
ctypedefine('struct',typename,field1,type1,field2,type2,...)
```

The argument `typename` must be a valid identifier string. Each of of the `field` arguments is also a valid identifier string that describe in order, the elements of the C structure. The `type` arguments are `typespecs`. They can be of three types:

- Built in types, e.g. `'uint8'` or `'double'` to name a couple of examples.
- C types that have previously been defined with a call to `ctypedefine`, e.g. `'mytype'` where `'mytype'` has already been defined through a call to `ctypedefine`.
- Arrays of either built in types or previously defined C types with the length of the array coded as an integer in square brackets, for example: `'uint8[10]'` or `'double[1000]'`.

To define a C enumeration, use the `'enum'` type class. The usage in this case is: ctypedefine('enum',typename,name1,value1 @] The argument `typename` must be a valid identifier string. Each of the `name` arguments must also be valid identifier strings that describe the possible values that the enumeration can take an, and their corresponding integer values. Note that the names should be unique. The behavior of the various `cenum` functions is undefined if the names are not unique.

To define a C alias (or typedef), use the following form of `ctypedefine`:

```
ctypedefine('alias',typename,aliased_typename)
```

where `aliased\_typename` is the type that is being aliased to.

29.4 CTYPEFREEZE Convert FreeMat Structure to C Type

29.4.1 Usage

The `ctypefreeze` function is used to convert a FreeMat structure into a C struct as defined by a C structure typedef. To use the `cstructfreeze` function, you must first define the type of the C structure using the `ctypedefine` function. The `ctypefreeze` function then serializes a FreeMat structure to a set of bytes, and returns it as an array. The usage for `ctypefreeze` is

```
byte_array = ctypefreeze(mystruct, 'typename')
```

where `mystruct` is the array we want to 'freeze' to a memory array, and `typename` is the name of the C type that we want the resulting byte array to conform to.

29.5 CTYPENEW Create New Instance of C Structure

29.5.1 Usage

The `ctypenew` function is a convenience function for creating a FreeMat structure that corresponds to a C structure. The entire structure is initialized with zeros. This has some negative implications, because if the structure definition uses `cenums`, they may come out as `'unknown'` values if there are no enumerations corresponding to zero. The use of the function is

```
a = ctypenew('typename')
```

which creates a single structure of C structure type `'typename'`. To create an array of structures, we can provide a second argument

```
a = ctypenew('typename',count)
```

where `count` is the number of elements in the structure array.

29.6 CTYPEPRINT Print C Type

29.6.1 Usage

The `ctypeprint` function prints a C type on the console. The usage is

```
ctypeprint(typename)
```

where `typename` is a string containing the name of the C type to print. Depending on the class of the C type (e.g., structure, alias or enumeration) the `ctypeprint` function will dump information about the type definition.

29.7 CTYPEREAD Read a C Structure From File

29.7.1 Usage

The `ctyperead` function is a convenience function for reading a C structure from a file. This is generally a very bad idea, as direct writing of C structures to files is notoriously unportable. Consider yourself warned. The syntax for this function is

```
a = ctyperead(fid,'typename')
```

where `'typename'` is a string containing the name of the C structure as defined using `ctypedefine`, and `fid` is the file handle returned by the `fopen` command. Note that this form will read a single structure from the file. If you want to read multiple structures into an array, use the following form

```
a = ctyperead(fid,'typename',count)
```

Note that the way this function works is by using `ctypesize` to compute the size of the structure, reading that many bytes from the file, and then calling `ctypethaw` on the resulting buffer. A consequence of this behavior is that the byte-endian corrective behavior of FreeMat does not work.

29.8 CTYPESIZE Compute Size of C Struct

29.8.1 Usage

The `ctypesize` function is used to compute the size of a C structure that is defined using the `ctypedefine` function. The usage of `ctypesize` is

```
size = ctypesize('typename')
```

where `typename` is the name of the C structure you want to compute the size of. The returned count is measured in bytes. Note that as indicated in the help for `ctypedefine` that FreeMat does not automatically pad the entries of the structure to match the particulars of your C compiler. Instead, the assumption is that you have adequate padding entries in your structure to align the FreeMat members with the C ones. See `ctypedefine` for more details. You can also specify an optional count parameter if you want to determine how large multiple structures are

```
size = ctypesize('typename',count)
```

29.9 CTYPETHAW Convert C Struct to FreeMat Structure

29.9.1 Usage

The `ctypethaw` function is used to convert a C structure that is encoded in a byte array into a FreeMat structure using a C structure typedef. To use the `ctypethaw` function, you must first define the type of the C structure using the `ctypedefine` function. The usage of `ctypethaw` is

```
mystruct = ctypethaw(byte_array, 'typename')
```

where `byte\_array` is a `uint8` array containing the bytes that encode the C structure, and `typename` is a string that contains the type description as registered with `ctypedefine`. If you want to retrieve multiple structures from a single byte array, you can specify a count as

```
mystruct = ctypethaw(byte_array, 'typename', count)
```

where `count` is an integer containing the number of structures to retrieve. Sometimes it is also useful to retrieve only part of the structure from a byte array, and then (based on the contents of the structure) retrieve more data. In this case, you can retrieve the residual byte array using the optional second output argument of `ctypethaw`:

```
[mystruct,byte_array_remaining] = ctypethaw(byte_array, 'typename',...)
```

29.10 CTYPEWRITE Write a C Typedef To File

29.10.1 Usage

The `ctypewrite` function is a convenience function for writing a C typedef to a file. This is generally a very bad idea, as writing of C typedefs to files is notoriously unportable. Consider yourself warned. The syntax for this function is

```
ctypewrite(fid,a,'typename')
```

where `a` is the FreeMat typedef to write, `'typename'` is a string containing the name of the C typedef to use when writing the typedef to the file (previously defined using `ctypedefine`), and `fid` is the file handle returned by `fopen`.

29.11 IMPORT Foreign Function Import

29.11.1 Usage

The import function allows you to call functions that are compiled into shared libraries, as if they were FreeMat functions. The usage is

```
import(libraryname,symbol,function,return,arguments)
```

The argument `libraryname` is the name of the library (as a string) to import the function from. The second argument `symbol` (also a string), is the name of the symbol to import from the library. The third argument `function` is the the name of the function after its been imported into Freemat. The fourth argument is a string that specifies the return type of the function. It can take on one of the following types:

- 'uint8' for an unsigned, 8-bit integer.
- 'int8' for a signed, 8-bit integer.
- 'uint16' an unsigned, 16-bit integer.
- 'int16' a signed, 16-bit integer.
- 'uint32' for an unsigned, 32-bit integer.
- 'int32' for a signed, 32-bit integer.
- 'single' for a 32-bit floating point.
- 'double' for a 64-bit floating point.
- 'void' for no return type.

The fourth argument is more complicated. It encodes the arguments of the imported function using a special syntax. In general, the argument list is a string consisting of entries of the form:

```
type[optional bounds check] {optional &}name
```

Here is a list of various scenarios (expressed in 'C'), and the corresponding entries, along with snippets of code.

Scalar variable passed by value: Suppose a function is defined in the library as

```
int fooFunction(float t),
```

i.e., it takes a scalar value (or a string) that is passed by value. Then the corresponding argument string would be

```
'float t'
```

For a C-string, which corresponds to a function prototype of

```
int fooFunction(const char *t),
```

the corresponding argument string would be

```
'string t'
```

Other types are as listed above. Note that FreeMat will automatically promote the type of scalar variables to the type expected by the `C` function. For example, if we call a function expecting a `float` with a `double` or `int16` argument, then FreeMat will automatically apply type promotion rules prior to calling the function.

Scalar variable passed by reference: Suppose a function is defined in the library as

```
int fooFunction(float *t),
```

i.e., it takes a scalar value (or a string) that is passed as a pointer. Then the corresponding argument string would be

```
'float &t'
```

If the function `fooFunction` modifies `t`, then the argument passed in FreeMat will also be modified.

Array variable passed by value: In `C`, it is impossible to distinguish an array being passed from a simple pointer being passed. More often than not, another argument indicates the length of the array. FreeMat has the ability to perform bounds-checking on array values. For example, suppose we have a function of the form

```
int sum_onehundred_ints(int *t),
```

where `sum\_onehundred\_ints` assumes that `t` is a length `100` vector. Then the corresponding FreeMat argument is

```
'float32[100] t'.
```

Note that because the argument is marked as not being passed by reference, that if `sub\_onehundred\_ints` modifies the array `t`, this will not affect the FreeMat argument. Note that the bounds-check expression can be any legal scalar expression that evaluates to an integer, and can be a function of the arguments. For example to pass a square $N \times N$ matrix to the following function:

```
float determinantmatrix(int N, float *A),
```

we can use the following argument to `import`:

```
'int32 N, float[N*N] t'.
```

Array variable passed by reference: If the function in `C` modifies an array, and we wish this to be reflected in the FreeMat side, we must pass that argument by reference. Hence, consider the following hypothetical function that squares the elements of an array (functionally equivalent to $x.^2$):

```
void squarearray(int N, float *A)
```

we can use the following argument to `import`:

```
'int32 N, float[N] &A'.
```

Note that to avoid problems with memory allocation, external functions are not allowed to return pointers. As a result, as a general operating mechanism, the FreeMat code must allocate the proper arrays, and then pass them by reference to the external function.

Sending text to the FreeMat console: Starting with FreeMat 4.0, it is possible for external code that is called using the `import` statement to send text to the FreeMat console. To do so, you must define in each library that wants to send text a function with the name `freemat\_io\_handler` that captures its argument (a function pointer), and stores it for use by functions in the library. That function pointer takes a standard C string argument. Here is a snippet of code to demonstrate how this works:

```
/* just to make it readable */
typedef void (*strfunc)(const char*);

/* The name we want to use for the function */
strfunc FreeMatText;

/* this name is case sensitive and must not be mangled (i.e., use extern "C") */
void freemat_io_handler(strfunc printFunc) {
   FreeMatText = printFunc;
}

double SomeImportedFunction(double t) {
   FreeMatText("I am going to double that argument!\n");
   return (t*2);
}
```

In this case, once `SomeImportedFunction` is called from within FreeMat, the text `''I am going to double that argument'` will appear in the FreeMat console.

Your `freemat\_io\_handler` function is automatically called when your library is loaded by FreeMat, which happens with the first `import` statement that references it.

Repeating `import` calls to import the same function name will be ignored, except the first call. In order to refresh the function without restarting FreeMat, you have to first clear all imported libraries via:

```
clear 'libs'
```

29.11.2 Example

Here is a complete example. We have a `C` function that adds two float vectors of the same length, and stores the result in a third array that is modified by the function. First, the `C` code:

```
    addArrays.c
void addArrays(int N, float *a, float *b, float *c) {
  int i;

  for (i=0;i<N;i++)
   c[i] = a[i] + b[i];
}
```

We then compile this into a dynamic library, say, `add.so`. The import command would then be:

```
import('add.so','addArrays','addArrays','void', ...
       'int32 N, float[N] a, float[N] b, float[N] &c');
```

We could then exercise the function exactly as if it had been written in FreeMat. The following only works on systems using the GNU C Compiler:

```
--> if (strcmp(computer,'MAC')) system('gcc -bundle -flat_namespace -undefined suppress -o add.so addAr
--> if (~strcmp(computer,'MAC')) system('gcc -shared -fPIC -o add.so addArrays.c'); end;
--> import('add.so','addArrays','addArrays','void','int32 N, float[N] a, float[N] b, float[N] &c');
--> a = [3,2,3,1];
--> b = [5,6,0,2];
--> c = [0,0,0,0];
--> addArrays(length(a),a,b,c)

ans =
  []
--> c

ans =
 8 8 3 3
```

29.12 LOADLIB Load Library Function

29.12.1 Usage

The `loadlib` function allows a function in an external library to be added to FreeMat dynamically. This interface is generally to be used as last resort, as the form of the function being called is assumed to match the internal implementation. In short, this is not the interface mechanism of choice. For all but very complicated functions, the `import` function is the preferred approach. Thus, only a very brief summary of it is presented here. The syntax for `loadlib` is

```
loadlib(libfile, symbolname, functionname, nargin, nargout)
```

where `libfile` is the complete path to the library to use, `symbolname` is the name of the symbol in the library, `functionname` is the name of the function after it is imported into FreeMat (this is optional, it defaults to the `symbolname`), `nargin` is the number of input arguments (defaults to 0), and `nargout` is the number of output arguments (defaults to 0). If the number of (input or output) arguments is variable then set the corresponding argument to `-1`.

Chapter 30

Visualization Toolkit Common Classes

30.1 vtkAbstractArray

30.1.1 Usage

vtkAbstractArray is an abstract superclass for data array objects. This class defines an API that all subclasses must support. The data type must be assignable and copy-constructible, but no other assumptions about its type are made. Most of the subclasses of this array deal with numeric data either as scalars or tuples of scalars. A program can use the IsNumeric() method to check whether an instance of vtkAbstractArray contains numbers. It is also possible to test for this by attempting to SafeDownCast an array to an instance of vtkDataArray, although this assumes that all numeric arrays will always be descended from vtkDataArray.

¡p¿

Every array has a character-string name. The naming of the array occurs automatically when it is instantiated, but you are free to change this name using the SetName() method. (The array name is used for data manipulation.)

To create an instance of class vtkAbstractArray, simply invoke its constructor as follows

```
obj = vtkAbstractArray
```

30.1.2 Methods

The class vtkAbstractArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractArray = obj.NewInstance ()`
- `vtkAbstractArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.Allocate (vtkIdType sz, vtkIdType ext)` - Allocate memory for this array. Delete old storage only if necessary. Note that ext is no longer used.
- `obj.Initialize ()` - Release storage and reset array to initial state.
- `int = obj.GetDataType ()` - Return the underlying data type. An integer indicating data type is returned as specified in vtkSetGet.h.

- `int = obj.GetDataTypeSize ()` - Return the size of the underlying data type. For a bit, 0 is returned. For string 0 is returned. Arrays with variable length components return 0.

- `int = obj.GetElementComponentSize ()` - Return the size, in bytes, of the lowest-level element of an array. For vtkDataArray and subclasses this is the size of the data type. For vtkStringArray, this is sizeof(vtkStdString::value_type), which winds up being sizeof(char).

- `obj.SetNumberOfComponents (int )` - Set/Get the dimention (n) of the components. Must be ¿= 1. Make sure that this is set before allocation.

- `int = obj.GetNumberOfComponentsMinValue ()` - Set/Get the dimention (n) of the components. Must be ¿= 1. Make sure that this is set before allocation.

- `int = obj.GetNumberOfComponentsMaxValue ()` - Set/Get the dimention (n) of the components. Must be ¿= 1. Make sure that this is set before allocation.

- `int = obj.GetNumberOfComponents ()` - Set the number of tuples (a component group) in the array. Note that this may allocate space depending on the number of components. Also note that if allocation is performed no copy is performed so existing data will be lost (if data conservation is sought, one may use the Resize method instead).

- `obj.SetNumberOfTuples (vtkIdType number)` - Set the number of tuples (a component group) in the array. Note that this may allocate space depending on the number of components. Also note that if allocation is performed no copy is performed so existing data will be lost (if data conservation is sought, one may use the Resize method instead).

- `vtkIdType = obj.GetNumberOfTuples ()` - Set the tuple at the ith location using the jth tuple in the source array. This method assumes that the two arrays have the same type and structure. Note that range checking and memory allocation is not performed; use in conjunction with SetNumberOfTuples() to allocate space.

- `obj.SetTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Set the tuple at the ith location using the jth tuple in the source array. This method assumes that the two arrays have the same type and structure. Note that range checking and memory allocation is not performed; use in conjunction with SetNumberOfTuples() to allocate space.

- `obj.InsertTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at ith location in this array. Note that memory allocation is performed as necessary to hold the data.

- `vtkIdType = obj.InsertNextTuple (vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at the end in this array. Note that memory allocation is performed as necessary to hold the data. Returns the location at which the data was inserted.

- `obj.GetTuples (vtkIdList ptIds, vtkAbstractArray output)` - Given a list of point ids, return an array of tuples. You must insure that the output array has been previously allocated with enough space to hold the data.

- `obj.GetTuples (vtkIdType p1, vtkIdType p2, vtkAbstractArray output)` - Get the tuples for the range of points ids specified (i.e., p1-¿p2 inclusive). You must insure that the output array has been previously allocated with enough space to hold the data.

- `obj.DeepCopy (vtkAbstractArray da)` - Deep copy of data. Implementation left to subclasses, which should support as many type conversions as possible given the data type.

  Subclasses should call vtkAbstractArray::DeepCopy() so that the information object (if one exists) is copied from da.

- `obj.InterpolateTuple (vtkIdType i, vtkIdList ptIndices, vtkAbstractArray source, double weights)` - Set the ith tuple in this array as the interpolated tuple value, given the ptIndices in the source array and associated interpolation weights. This method assumes that the two arrays are of the same type and strcuture.
- `obj.InterpolateTuple (vtkIdType i, vtkIdType id1, vtkAbstractArray source1, vtkIdType id2, vtkAbstr`
- `obj.Squeeze ()` - Resize object to just fit data requirement. Reclaims extra memory.
- `int = obj.Resize (vtkIdType numTuples)` - Resize the array while conserving the data. Returns 1 if resizing succeeded and 0 otherwise.
- `obj.Reset ()` - Return the size of the data.
- `vtkIdType = obj.GetSize ()` - What is the maximum id currently in the array.
- `vtkIdType = obj.GetMaxId ()` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the supplied array.
- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this data array. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.
- `obj.SetName (string )` - Set/get array's name
- `string = obj.GetName ()` - Set/get array's name
- `string = obj.GetDataTypeAsString (void )` - Creates an array for dataType where dataType is one of VTK_BIT, VTK_CHAR, VTK_UNSIGNED_CHAR, VTK_SHORT, VTK_UNSIGNED_SHORT, VTK_INT, VTK_UNSIGNED_INT, VTK_LONG, VTK_UNSIGNED_LONG, VTK_DOUBLE, VTK_DOUBLE, VTK_ID_TYPE, VTK_STRING. Note that the data array returned has be deleted by the user.
- `int = obj.IsNumeric ()` - This method is here to make backward compatibility easier. It must return true if and only if an array contains numeric data.
- `vtkArrayIterator = obj.NewIterator ()` - Subclasses must override this method and provide the right kind of templated vtkArrayIteratorTemplate.
- `vtkIdType = obj.GetDataSize ()` - Tell the array explicitly that the data has changed. This is only necessary to call when you modify the array contents without using the array's API (i.e. you retrieve a pointer to the data and modify the array contents). You need to call this so that the fast lookup will know to rebuild itself. Otherwise, the lookup functions will give incorrect results.
- `obj.DataChanged ()` - Tell the array explicitly that the data has changed. This is only necessary to call when you modify the array contents without using the array's API (i.e. you retrieve a pointer to the data and modify the array contents). You need to call this so that the fast lookup will know to rebuild itself. Otherwise, the lookup functions will give incorrect results.
- `obj.ClearLookup ()` - Delete the associated fast lookup data structure on this array, if it exists. The lookup will be rebuilt on the next call to a lookup function.
- `vtkInformation = obj.GetInformation ()` - Get an information object that can be used to annotate the array. This will always return an instance of vtkInformation, if one is not currently associated with the array it will be created.
- `bool = obj.HasInformation ()`

30.2 vtkAbstractTransform

30.2.1 Usage

vtkAbstractTransform is the superclass for all VTK geometric transformations. The VTK transform hierarchy is split into two major branches: warp transformations and homogeneous (including linear) transformations. The latter can be represented in terms of a 4x4 transformation matrix, the former cannot. ¡p¿Transformations can be pipelined through two mechanisms: ¡p¿1) GetInverse() returns the pipelined inverse of a transformation i.e. if you modify the original transform, any transform previously returned by the GetInverse() method will automatically update itself according to the change. ¡p¿2) You can do pipelined concatenation of transformations through the vtkGeneralTransform class, the vtkPerspectiveTransform class, or the vtkTransform class.

To create an instance of class vtkAbstractTransform, simply invoke its constructor as follows

```
obj = vtkAbstractTransform
```

30.2.2 Methods

The class vtkAbstractTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractTransform = obj.NewInstance ()`
- `vtkAbstractTransform = obj.SafeDownCast (vtkObject o)`
- `obj.TransformPoint (float in[3], float out[3])` - Apply the transformation to a coordinate. You can use the same array to store both the input and output point.
- `obj.TransformPoint (double in[3], double out[3])` - Apply the transformation to a double-precision coordinate. You can use the same array to store both the input and output point.
- `double = obj.TransformPoint (double x, double y, double z)` - Apply the transformation to a double-precision coordinate. Use this if you are programming in Python, tcl or Java.
- `double = obj.TransformPoint (double point[3])` - Apply the transformation to a double-precision coordinate. Use this if you are programming in Python, tcl or Java.
- `float = obj.TransformFloatPoint (float x, float y, float z)` - Apply the transformation to an (x,y,z) coordinate. Use this if you are programming in Python, tcl or Java.
- `float = obj.TransformFloatPoint (float point[3])` - Apply the transformation to an (x,y,z) coordinate. Use this if you are programming in Python, tcl or Java.
- `double = obj.TransformDoublePoint (double x, double y, double z)` - Apply the transformation to a double-precision (x,y,z) coordinate. Use this if you are programming in Python, tcl or Java.
- `double = obj.TransformDoublePoint (double point[3])` - Apply the transformation to a double-precision (x,y,z) coordinate. Use this if you are programming in Python, tcl or Java.
- `obj.TransformNormalAtPoint (float point[3], float in[3], float out[3])` - Apply the transformation to a normal at the specified vertex. If the transformation is a vtkLinearTransform, you can use TransformNormal() instead.

- `obj.TransformNormalAtPoint (double point[3], double in[3], double out[3])` - Apply the transformation to a normal at the specified vertex. If the transformation is a vtkLinearTransform, you can use TransformNormal() instead.
- `double = obj.TransformNormalAtPoint (double point[3], double normal[3])`
- `double = obj.TransformDoubleNormalAtPoint (double point[3], double normal[3])` - Apply the transformation to a double-precision normal at the specified vertex. If the transformation is a vtkLinearTransform, you can use TransformDoubleNormal() instead.
- `float = obj.TransformFloatNormalAtPoint (float point[3], float normal[3])` - Apply the transformation to a single-precision normal at the specified vertex. If the transformation is a vtkLinearTransform, you can use TransformFloatNormal() instead.
- `obj.TransformVectorAtPoint (float point[3], float in[3], float out[3])` - Apply the transformation to a vector at the specified vertex. If the transformation is a vtkLinearTransform, you can use TransformVector() instead.
- `obj.TransformVectorAtPoint (double point[3], double in[3], double out[3])` - Apply the transformation to a vector at the specified vertex. If the transformation is a vtkLinearTransform, you can use TransformVector() instead.
- `double = obj.TransformVectorAtPoint (double point[3], double vector[3])`
- `double = obj.TransformDoubleVectorAtPoint (double point[3], double vector[3])` - Apply the transformation to a double-precision vector at the specified vertex. If the transformation is a vtkLinearTransform, you can use TransformDoubleVector() instead.
- `float = obj.TransformFloatVectorAtPoint (float point[3], float vector[3])` - Apply the transformation to a single-precision vector at the specified vertex. If the transformation is a vtkLinearTransform, you can use TransformFloatVector() instead.
- `obj.TransformPoints (vtkPoints inPts, vtkPoints outPts)` - Apply the transformation to a series of points, and append the results to outPts.
- `obj.TransformPointsNormalsVectors (vtkPoints inPts, vtkPoints outPts, vtkDataArray inNms, vtkDataAr` - Apply the transformation to a combination of points, normals and vectors.
- `vtkAbstractTransform = obj.GetInverse ()` - Get the inverse of this transform. If you modify this transform, the returned inverse transform will automatically update. If you want the inverse of a vtkTransform, you might want to use GetLinearInverse() instead which will type cast the result from vtkAbstractTransform to vtkLinearTransform.
- `obj.SetInverse (vtkAbstractTransform transform)` - Set a transformation that this transform will be the inverse of. This transform will automatically update to agree with the inverse transform that you set.
- `obj.Inverse ()` - Invert the transformation.
- `obj.DeepCopy (vtkAbstractTransform )` - Copy this transform from another of the same type.
- `obj.Update ()` - Update the transform to account for any changes which have been made. You do not have to call this method yourself, it is called automatically whenever the transform needs an update.
- `obj.InternalTransformPoint (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.

- `int = obj.CircuitCheck (vtkAbstractTransform transform)` - Check for self-reference. Will return true if concatenating with the specified transform, setting it to be our inverse, or setting it to be our input will create a circular reference. CircuitCheck is automatically called by SetInput(), SetInverse(), and Concatenate(vtkXTransform *). Avoid using this function, it is experimental.

- `long = obj.GetMTime ()` - Override GetMTime necessary because of inverse transforms.

- `obj.UnRegister (vtkObjectBase O)` - Needs a special UnRegister() implementation to avoid circular references.

- `obj.Identity ()` - @deprecated This method is deprecated in the base class. It is still valid to use it on many of the specialized classes.

30.3 vtkAmoebaMinimizer

30.3.1 Usage

vtkAmoebaMinimizer will modify a set of parameters in order to find the minimum of a specified function. The method used is commonly known as the amoeba method, it constructs an n-dimensional simplex in parameter space (i.e. a tetrahedron if the number or parameters is 3) and moves the vertices around parameter space until a local minimum is found. The amoeba method is robust, reasonably efficient, but is not guaranteed to find the global minimum if several local minima exist.

To create an instance of class vtkAmoebaMinimizer, simply invoke its constructor as follows

```
obj = vtkAmoebaMinimizer
```

30.3.2 Methods

The class vtkAmoebaMinimizer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAmoebaMinimizer class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkAmoebaMinimizer = obj.NewInstance ()`

- `vtkAmoebaMinimizer = obj.SafeDownCast (vtkObject o)`

- `obj.SetParameterValue (string name, double value)` - Set the initial value for the specified parameter. Calling this function for any parameter will reset the Iterations and the FunctionEvaluations counts to zero. You must also use SetParameterScale() to specify the step size by which the parameter will be modified during the minimization. It is preferable to specify parameters by name, rather than by number.

- `obj.SetParameterValue (int i, double value)` - Set the initial value for the specified parameter. Calling this function for any parameter will reset the Iterations and the FunctionEvaluations counts to zero. You must also use SetParameterScale() to specify the step size by which the parameter will be modified during the minimization. It is preferable to specify parameters by name, rather than by number.

- `obj.SetParameterScale (string name, double scale)` - Set the scale to use when modifying a parameter, i.e. the initial amount by which the parameter will be modified during the search for the minimum. It is preferable to identify scalars by name rather than by number.

- `double = obj.GetParameterScale (string name)` - Set the scale to use when modifying a parameter, i.e. the initial amount by which the parameter will be modified during the search for the minimum. It is preferable to identify scalars by name rather than by number.

- `obj.SetParameterScale (int i, double scale)` - Set the scale to use when modifying a parameter, i.e. the initial amount by which the parameter will be modified during the search for the minimum. It is preferable to identify scalars by name rather than by number.

- `double = obj.GetParameterScale (int i)` - Set the scale to use when modifying a parameter, i.e. the initial amount by which the parameter will be modified during the search for the minimum. It is preferable to identify scalars by name rather than by number.

- `double = obj.GetParameterValue (string name)` - Get the value of a parameter at the current stage of the minimization. Call this method within the function that you are minimizing in order to get the current parameter values. It is preferable to specify parameters by name rather than by index.

- `double = obj.GetParameterValue (int i)` - Get the value of a parameter at the current stage of the minimization. Call this method within the function that you are minimizing in order to get the current parameter values. It is preferable to specify parameters by name rather than by index.

- `string = obj.GetParameterName (int i)` - For completeness, an unchecked method to get the name for particular parameter (the result will be NULL if no name was set).

- `int = obj.GetNumberOfParameters ()` - Get the number of parameters that have been set.

- `obj.Initialize ()` - Initialize the minimizer. This will reset the number of parameters to zero so that the minimizer can be reused.

- `obj.Minimize ()` - Iterate until the minimum is found to within the specified tolerance, or until the MaxIterations has been reached.

- `int = obj.Iterate ()` - Perform one iteration of minimization. Returns zero if the tolerance stopping criterion has been met.

- `obj.SetFunctionValue (double )` - Get the function value resulting from the minimization.

- `double = obj.GetFunctionValue ()` - Get the function value resulting from the minimization.

- `obj.SetTolerance (double )` - Specify the fractional tolerance to aim for during the minimization.

- `double = obj.GetTolerance ()` - Specify the fractional tolerance to aim for during the minimization.

- `obj.SetMaxIterations (int )` - Specify the maximum number of iterations to try before giving up.

- `int = obj.GetMaxIterations ()` - Specify the maximum number of iterations to try before giving up.

- `int = obj.GetIterations ()` - Return the number of interations that have been performed. This is not necessarily the same as the number of function evaluations.

- `int = obj.GetFunctionEvaluations ()` - Return the number of times that the function has been evaluated.

- `obj.EvaluateFunction ()` - Evaluate the function. This is usually called internally by the minimization code, but it is provided here as a public method.

30.4 vtkAnimationCue

30.4.1 Usage

vtkAnimationCue and vtkAnimationScene provide the framework to support animations in VTK. vtkAnimationCue represents an entity that changes/ animates with time, while vtkAnimationScene represents scene or setup for the animation, which consists on individual cues or other scenes.

A cue has three states: UNINITIALIZED, ACTIVE and INACTIVE. UNINITIALIZED represents an point in time before the start time of the cue. The cue is in ACTIVE state at a point in time between start time and end time for the cue. While, beyond the end time, it is in INACTIVE state. When the cue enters the ACTIVE state, StartAnimationCueEvent is fired. This event may be handled to initialize the entity to be animated. When the cue leaves the ACTIVE state, EndAnimationCueEvent is fired, which can be handled to cleanup after having run the animation. For every request to render during the ACTIVE state, AnimationCueTickEvent is fired, which must be handled to perform the actual animation.

To create an instance of class vtkAnimationCue, simply invoke its constructor as follows

```
obj = vtkAnimationCue
```

30.4.2 Methods

The class vtkAnimationCue has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAnimationCue class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAnimationCue = obj.NewInstance ()`
- `vtkAnimationCue = obj.SafeDownCast (vtkObject o)`
- `obj.SetTimeMode (int mode)` - Get/Set the time mode. In Normalized mode, the start and end times of the cue are normalized [0,1] with respect to the start and end times of the container scene. In Relative mode the start and end time of the cue are specified in offset seconds relative to the start time of the container scene.
- `int = obj.GetTimeMode ()` - Get/Set the time mode. In Normalized mode, the start and end times of the cue are normalized [0,1] with respect to the start and end times of the container scene. In Relative mode the start and end time of the cue are specified in offset seconds relative to the start time of the container scene.
- `obj.SetTimeModeToRelative ()` - Get/Set the time mode. In Normalized mode, the start and end times of the cue are normalized [0,1] with respect to the start and end times of the container scene. In Relative mode the start and end time of the cue are specified in offset seconds relative to the start time of the container scene.
- `obj.SetTimeModeToNormalized ()` - Get/Set the Start time for this cue. When the current time is ¿= StartTime, the Cue is in ACTIVE state. if Current time i ¡ StartTime, the Cue is in UNINITIALIZED state. Whenever the cue enters the ACTIVE state from an INACTIVE state, it triggers the StartEvent. The Start time is in seconds relative to the start of the container Scene (when in Relative time mode) or is normalized over the span of the container Scene (when in Normalized time mode).
- `obj.SetStartTime (double )` - Get/Set the Start time for this cue. When the current time is ¿= StartTime, the Cue is in ACTIVE state. if Current time i ¡ StartTime, the Cue is in UNINITIALIZED state. Whenever the cue enters the ACTIVE state from an INACTIVE state, it triggers the StartEvent. The Start time is in seconds relative to the start of the container Scene (when in Relative time mode) or is normalized over the span of the container Scene (when in Normalized time mode).

- `double = obj.GetStartTime ()` - Get/Set the Start time for this cue. When the current time is ¿= StartTime, the Cue is in ACTIVE state. if Current time i ¡ StartTime, the Cue is in UNINITIALIZED state. Whenever the cue enters the ACTIVE state from an INACTIVE state, it triggers the StartEvent. The Start time is in seconds relative to the start of the container Scene (when in Relative time mode) or is normalized over the span of the container Scene (when in Normalized time mode).

- `obj.SetEndTime (double )` - Get/Set the End time for this cue. When the current time is ¿ EndTime, the Cue is in INACTIVE state. Whenever the cue leaves an ACTIVE state to enter INACTIVE state, the EndEvent is triggered. The End time is in seconds relative to the start of the container Scene (when in Relative time mode) or is normalized over the span of the container Scene (when in Normalized time mode).

- `double = obj.GetEndTime ()` - Get/Set the End time for this cue. When the current time is ¿ EndTime, the Cue is in INACTIVE state. Whenever the cue leaves an ACTIVE state to enter INACTIVE state, the EndEvent is triggered. The End time is in seconds relative to the start of the container Scene (when in Relative time mode) or is normalized over the span of the container Scene (when in Normalized time mode).

- `obj.Tick (double currenttime, double deltatime, double clocktime)` - Indicates a tick or point in time in the animation. Triggers a Tick event if currenttime ¿= StartTime and currenttime ¡= EndTime. Whenever the state of the cue changes, either StartEvent or EndEvent is triggerred depending upon whether the cue entered Active state or quit active state respectively. The current time is relative to the start of the container Scene (when in Relative time mode) or is normalized over the span of the container Scene (when in Normalized time mode). deltatime is the time since last call to Tick. deltatime also can be in seconds relative to the start of the container Scene or normalized depending upon the cue's Time mode. clocktime is the time from the scene i.e. it does not depend on the time mode for the cue. For the first call to Tick after a call to Initialize(), the deltatime is 0;

- `obj.Initialize ()` - Called when the playing of the scene begins. This will set the Cue to UNINITIALIZED state.

- `obj.Finalize ()` - Called when the scene reaches the end. If the cue state is ACTIVE when this method is called, this will trigger a EndAnimationCueEvent.

- `double = obj.GetAnimationTime ()` - This is valid only in a AnimationCueTickEvent handler. Before firing the event the animation cue sets the AnimationTime to the time of the tick.

- `double = obj.GetDeltaTime ()` - This is valid only in a AnimationCueTickEvent handler. Before firing the event the animation cue sets the DeltaTime to the difference in time between the current tick and the last tick.

- `double = obj.GetClockTime ()` - This is valid only in a AnimationCueTickEvent handler. Before firing the event the animation cue sets the ClockTime to the time of the tick. ClockTime is directly the time from the animation scene neither normalized nor offsetted to the start of the scene.

30.5 vtkAnimationScene

30.5.1 Usage

vtkAnimationCue and vtkAnimationScene provide the framework to support animations in VTK. vtkAnimationCue represents an entity that changes/ animates with time, while vtkAnimationScene represents scene or setup for the animation, which consists of individual cues or other scenes.

A scene can be played in real time mode, or as a seqence of frames 1/frame rate apart in time.

To create an instance of class vtkAnimationScene, simply invoke its constructor as follows

```
obj = vtkAnimationScene
```

30.5.2 Methods

The class vtkAnimationScene has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAnimationScene class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAnimationScene = obj.NewInstance ()`
- `vtkAnimationScene = obj.SafeDownCast (vtkObject o)`
- `obj.SetPlayMode (int )` - Get/Set the PlayMode for running/playing the animation scene. In the Sequence mode, all the frames are generated one after the other. The time reported to each Tick of the constituent cues (during Play) is incremented by 1/frame rate, irrespective of the current time. In the RealTime mode, time indicates the instance in time.
- `obj.SetModeToSequence ()` - Get/Set the PlayMode for running/playing the animation scene. In the Sequence mode, all the frames are generated one after the other. The time reported to each Tick of the constituent cues (during Play) is incremented by 1/frame rate, irrespective of the current time. In the RealTime mode, time indicates the instance in time.
- `obj.SetModeToRealTime ()` - Get/Set the PlayMode for running/playing the animation scene. In the Sequence mode, all the frames are generated one after the other. The time reported to each Tick of the constituent cues (during Play) is incremented by 1/frame rate, irrespective of the current time. In the RealTime mode, time indicates the instance in time.
- `int = obj.GetPlayMode ()` - Get/Set the PlayMode for running/playing the animation scene. In the Sequence mode, all the frames are generated one after the other. The time reported to each Tick of the constituent cues (during Play) is incremented by 1/frame rate, irrespective of the current time. In the RealTime mode, time indicates the instance in time.
- `obj.SetFrameRate (double )` - Get/Set the frame rate (in frames per second). This parameter affects only in the Sequence mode. The time interval indicated to each cue on every tick is progressed by 1/frame-rate seconds.
- `double = obj.GetFrameRate ()` - Get/Set the frame rate (in frames per second). This parameter affects only in the Sequence mode. The time interval indicated to each cue on every tick is progressed by 1/frame-rate seconds.
- `obj.AddCue (vtkAnimationCue cue)` - Add/Remove an AnimationCue to/from the Scene. It's an error to add a cue twice to the Scene.
- `obj.RemoveCue (vtkAnimationCue cue)` - Add/Remove an AnimationCue to/from the Scene. It's an error to add a cue twice to the Scene.
- `obj.RemoveAllCues ()` - Add/Remove an AnimationCue to/from the Scene. It's an error to add a cue twice to the Scene.
- `int = obj.GetNumberOfCues ()` - Add/Remove an AnimationCue to/from the Scene. It's an error to add a cue twice to the Scene.
- `obj.Play ()` - Starts playing the animation scene. Fires a vtkCommand::StartEvent before play beings and vtkCommand::EndEvent after play ends.
- `obj.Stop ()` - Stops the animation scene that is running.
- `obj.SetLoop (int )` - Enable/Disable animation loop.

- `int = obj.GetLoop ()` - Enable/Disable animation loop.
- `obj.SetAnimationTime (double time)` - Makes the state of the scene same as the given time.
- `double = obj.GetAnimationTime ()` - Makes the state of the scene same as the given time.
- `obj.SetTimeMode (int mode)` - Overridden to allow change to Normalized mode only if none of the constituent cues is in Relative time mode.
- `int = obj.IsInPlay ()`

30.6 vtkArray

30.6.1 Usage

vtkArray is the root of a hierarchy of arrays that can be used to store data with any number of dimensions. It provides an abstract interface for retrieving and setting array attributes that are independent of the type of values stored in the array - such as the number of dimensions, extents along each dimension, and number of values stored in the array.

To get and set array values, the vtkTypedArray template class derives from vtkArray and provides type-specific methods for retrieval and update.

Two concrete derivatives of vtkTypedArray are provided at the moment: vtkDenseArray and vtkSparseArray, which provide dense and sparse storage for arbitrary-dimension data, respectively. Toolkit users can create their own concrete derivatives that implement alternative storage strategies, such as compressed-sparse-row, etc. You could also create an array that provided read-only access to 'virtual' data, such as an array that returned a Fibonacci sequence, etc.

To create an instance of class vtkArray, simply invoke its constructor as follows

```
obj = vtkArray
```

30.6.2 Methods

The class vtkArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArray = obj.NewInstance ()`
- `vtkArray = obj.SafeDownCast (vtkObject o)`
- `bool = obj.IsDense ()` - Returns true iff the underlying array storage is "dense", i.e. that GetSize() and GetNonNullSize() will always return the same value. If not, the array is "sparse".
- `obj.Resize (vtkIdType i)` - Resizes the array to the given extents (number of dimensions and size of each dimension). Note that concrete implementations of vtkArray may place constraints on the the extents that they will store, so you cannot assume that GetExtents() will always return the same value passed to Resize().

  The contents of the array are undefined after calling Resize() - you should initialize its contents accordingly. In particular, dimension-labels will be undefined, dense array values will be undefined, and sparse arrays will be empty.

- `obj.Resize (vtkIdType i, vtkIdType j)` - Resizes the array to the given extents (number of dimensions and size of each dimension). Note that concrete implementations of vtkArray may place constraints on the the extents that they will store, so you cannot assume that GetExtents() will always return the same value passed to Resize().

  The contents of the array are undefined after calling Resize() - you should initialize its contents accordingly. In particular, dimension-labels will be undefined, dense array values will be undefined, and sparse arrays will be empty.

- `obj.Resize (vtkIdType i, vtkIdType j, vtkIdType k)` - Resizes the array to the given extents (number of dimensions and size of each dimension). Note that concrete implementations of vtkArray may place constraints on the the extents that they will store, so you cannot assume that GetExtents() will always return the same value passed to Resize().

  The contents of the array are undefined after calling Resize() - you should initialize its contents accordingly. In particular, dimension-labels will be undefined, dense array values will be undefined, and sparse arrays will be empty.

- `vtkIdType = obj.GetDimensions ()` - Returns the number of dimensions stored in the array. Note that this is the same as calling GetExtents().GetDimensions().

- `vtkIdType = obj.GetSize ()` - Returns the number of values stored in the array. Note that this is the same as calling GetExtents().GetSize(), and represents the maximum number of values that could ever be stored using the current extents. This is equal to the number of values stored in a dense array, but may be larger than the number of values stored in a sparse array.

- `vtkIdType = obj.GetNonNullSize ()` - Returns the number of non-null values stored in the array. Note that this value will equal GetSize() for dense arrays, and will be less-than-or-equal to GetSize() for sparse arrays.

- `obj.SetName (vtkStdString \&name)` - Sets the array name.

- `vtkStdString = obj.GetName ()` - Returns the array name.

- `obj.SetDimensionLabel (vtkIdType i, vtkStdString \&label)` - Sets the label for the i-th array dimension.

- `vtkStdString = obj.GetDimensionLabel (vtkIdType i)` - Returns the label for the i-th array dimension.

- `vtkArray = obj.DeepCopy ()` - Returns a new array that is a deep copy of this array.

30.7 vtkArrayIterator

30.7.1 Usage

vtkArrayIterator is used to iterate over elements in any vtkAbstractArray subclass. The vtkArrayIteratorTemplateMacro is used to centralize the set of types supported by Execute methods. It also avoids duplication of long switch statement case lists. Note that in this macro VTK_TT is defined to be the type of the iterator for the given type of array. One must include the vtkArrayIteratorIncludes.h header file to provide for extending of this macro by addition of new iterators.

Example usage:

```
vtkArrayIter* iter = array->NewIterator();
switch(array->GetDataType())
  {
  vtkArrayIteratorTemplateMacro(myFunc(static\_cast<VTK\_TT*>(iter), arg2));
  }
iter->Delete();
```

To create an instance of class vtkArrayIterator, simply invoke its constructor as follows

```
obj = vtkArrayIterator
```

30.7.2 Methods

The class vtkArrayIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArrayIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArrayIterator = obj.NewInstance ()`
- `vtkArrayIterator = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (vtkAbstractArray array)` - Set the array this iterator will iterate over. After Initialize() has been called, the iterator is valid so long as the Array has not been modified (except using the iterator itself). If the array is modified, the iterator must be re-intialized.
- `int = obj.GetDataType ()`

30.8 vtkAssemblyNode

30.8.1 Usage

vtkAssemblyNode represents a node in an assembly. It is used by vtkAssemblyPath to create hierarchical assemblies of props. The props can be either 2D or 3D.

An assembly node refers to a vtkProp, and possibly a vtkMatrix4x4. Nodes are used by vtkAssemblyPath to build fully evaluated path (matrices are concatenated through the path) that is used by picking and other operations involving assemblies.

To create an instance of class vtkAssemblyNode, simply invoke its constructor as follows

```
obj = vtkAssemblyNode
```

30.8.2 Methods

The class vtkAssemblyNode has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAssemblyNode class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAssemblyNode = obj.NewInstance ()`
- `vtkAssemblyNode = obj.SafeDownCast (vtkObject o)`
- `obj.SetViewProp (vtkProp prop)` - Set/Get the prop that this assembly node refers to.
- `vtkProp = obj.GetViewProp ()` - Set/Get the prop that this assembly node refers to.

- `obj.SetMatrix (vtkMatrix4x4 matrix)` - Specify a transformation matrix associated with the prop. Note: if the prop is not a type of vtkProp3D, then the transformation matrix is ignored (and expected to be NULL). Also, internal to this object the matrix is copied because the matrix is used for computation by vtkAssemblyPath.
- `vtkMatrix4x4 = obj.GetMatrix ()` - Specify a transformation matrix associated with the prop. Note: if the prop is not a type of vtkProp3D, then the transformation matrix is ignored (and expected to be NULL). Also, internal to this object the matrix is copied because the matrix is used for computation by vtkAssemblyPath.
- `long = obj.GetMTime ()` - Override the standard GetMTime() to check for the modified times of the prop and matrix.
- `obj.SetProp (vtkProp prop)` - @deprecated Replaced by vtkAssemblyNode::SetViewProp() as of VTK 5.0.
- `vtkProp = obj.GetProp ()` - @deprecated Replaced by vtkAssemblyNode::GetViewProp() as of VTK 5.0.

30.9 vtkAssemblyPath

30.9.1 Usage

vtkAssemblyPath represents an ordered list of assembly nodes that represent a fully evaluated assembly path. This class is used primarily for picking. Note that the use of this class is to add one or more assembly nodes to form the path. (An assembly node consists of an instance of vtkProp and vtkMatrix4x4, the matrix may be NULL.) As each node is added, the matrices are concatenated to create a final, evaluated matrix.

To create an instance of class vtkAssemblyPath, simply invoke its constructor as follows

```
obj = vtkAssemblyPath
```

30.9.2 Methods

The class vtkAssemblyPath has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAssemblyPath class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAssemblyPath = obj.NewInstance ()`
- `vtkAssemblyPath = obj.SafeDownCast (vtkObject o)`
- `obj.AddNode (vtkProp p, vtkMatrix4x4 m)` - Convenience method adds a prop and matrix together, creating an assembly node transparently. The matrix pointer m may be NULL. Note: that matrix is the one, if any, associated with the prop.
- `vtkAssemblyNode = obj.GetNextNode ()` - Get the next assembly node in the list. The node returned contains a pointer to a prop and a 4x4 matrix. The matrix is evaluated based on the preceding assembly hierarchy (i.e., the matrix is not necessarily as the same as the one that was added with AddNode() because of the concatenation of matrices in the assembly hierarchy).
- `vtkAssemblyNode = obj.GetFirstNode ()` - Get the first assembly node in the list. See the comments for GetNextNode() regarding the contents of the returned node. (Note: This node corresponds to the vtkProp associated with the vtkRenderer.

- `vtkAssemblyNode = obj.GetLastNode ()` - Get the last assembly node in the list. See the comments for GetNextNode() regarding the contents of the returned node.
- `obj.DeleteLastNode ()` - Delete the last assembly node in the list. This is like a stack pop.
- `obj.ShallowCopy (vtkAssemblyPath path)` - Perform a shallow copy (reference counted) on the incoming path.
- `long = obj.GetMTime ()` - Override the standard GetMTime() to check for the modified times of the nodes in this path.

30.10 vtkAssemblyPaths

30.10.1 Usage

vtkAssemblyPaths represents an assembly hierarchy as a list of vtkAssemblyPath. Each path represents the complete path from the top level assembly (if any) down to the leaf prop.

To create an instance of class vtkAssemblyPaths, simply invoke its constructor as follows

```
obj = vtkAssemblyPaths
```

30.10.2 Methods

The class vtkAssemblyPaths has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAssemblyPaths class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAssemblyPaths = obj.NewInstance ()`
- `vtkAssemblyPaths = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkAssemblyPath p)` - Add a path to the list.
- `obj.RemoveItem (vtkAssemblyPath p)` - Remove a path from the list.
- `int = obj.IsItemPresent (vtkAssemblyPath p)` - Determine whether a particular path is present. Returns its position in the list.
- `vtkAssemblyPath = obj.GetNextItem ()` - Get the next path in the list.
- `long = obj.GetMTime ()` - Override the standard GetMTime() to check for the modified times of the paths.

30.11 vtkBitArray

30.11.1 Usage

vtkBitArray is an array of bits (0/1 data value). The array is packed so that each byte stores eight bits. vtkBitArray provides methods for insertion and retrieval of bits, and will automatically resize itself to hold new data.

To create an instance of class vtkBitArray, simply invoke its constructor as follows

```
obj = vtkBitArray
```

30.11.2 Methods

The class vtkBitArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBitArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBitArray = obj.NewInstance ()`
- `vtkBitArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.Allocate (vtkIdType sz, vtkIdType ext)` - Allocate memory for this array. Delete old storage only if necessary. Note that ext is no longer used.
- `obj.Initialize ()` - Release storage and reset array to initial state.
- `int = obj.GetDataType ()`
- `int = obj.GetDataTypeSize ()` - Set the number of n-tuples in the array.
- `obj.SetNumberOfTuples (vtkIdType number)` - Set the number of n-tuples in the array.
- `obj.SetTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Set the tuple at the ith location using the jth tuple in the source array. This method assumes that the two arrays have the same type and structure. Note that range checking and memory allocation is not performed; use in conjunction with SetNumberOfTuples() to allocate space.
- `obj.InsertTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at ith location in this array. Note that memory allocation is performed as necessary to hold the data.
- `vtkIdType = obj.InsertNextTuple (vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at the end in this array. Note that memory allocation is performed as necessary to hold the data. Returns the location at which the data was inserted.
- `obj.GetTuple (vtkIdType i, double tuple)` - Copy the tuple value into a user-provided array.
- `obj.SetTuple (vtkIdType i, float tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTuple (vtkIdType i, double tuple)` - Set the tuple value at the ith location in the array.
- `obj.InsertTuple (vtkIdType i, float tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTuple (vtkIdType i, double tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `vtkIdType = obj.InsertNextTuple (float tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTuple (double tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `obj.RemoveTuple (vtkIdType id)` - These methods remove tuples from the data array. They shift data and resize array, so the data array is still valid after this operation. Note, this operation is fairly slow.
- `obj.RemoveFirstTuple ()` - These methods remove tuples from the data array. They shift data and resize array, so the data array is still valid after this operation. Note, this operation is fairly slow.

- `obj.RemoveLastTuple ()` - These methods remove tuples from the data array. They shift data and resize array, so the data array is still valid after this operation. Note, this operation is fairly slow.

- `obj.SetComponent (vtkIdType i, int j, double c)` - Set the data component at the ith tuple and jth component location. Note that i is less then NumberOfTuples and j is less then NumberOfComponents. Make sure enough memory has been allocated (use SetNumberOfTuples() and SetNumberOfComponents()).

- `obj.Squeeze ()` - Free any unneeded memory.

- `int = obj.Resize (vtkIdType numTuples)` - Resize the array while conserving the data.

- `int = obj.GetValue (vtkIdType id)` - Get the data at a particular index.

- `obj.SetNumberOfValues (vtkIdType number)` - Fast method based setting of values without memory checks. First use SetNumberOfValues then use SetValue to actually set them. Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.

- `obj.SetValue (vtkIdType id, int value)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.

- `obj.InsertValue (vtkIdType id, int i)` - Insets values and checks to make sure there is enough memory

- `vtkIdType = obj.InsertNextValue (int i)`

- `obj.InsertComponent (vtkIdType i, int j, double c)` - Insert the data component at ith tuple and jth component location. Note that memory allocation is performed as necessary to hold the data.

- `obj.DeepCopy (vtkDataArray da)` - Deep copy of another bit array.

- `obj.DeepCopy (vtkAbstractArray aa)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array. If save 0, the array must have been allocated with new[] not malloc.

- `obj.SetArray (string array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array. If save 0, the array must have been allocated with new[] not malloc.

- `vtkArrayIterator = obj.NewIterator ()` - Returns a new vtkBitArrayIterator instance.

- `vtkIdType = obj.LookupValue (int value)`

- `obj.LookupValue (int value, vtkIdList ids)`

- `obj.DataChanged ()` - Tell the array explicitly that the data has changed. This is only necessary to call when you modify the array contents without using the array's API (i.e. you retrieve a pointer to the data and modify the array contents). You need to call this so that the fast lookup will know to rebuild itself. Otherwise, the lookup functions will give incorrect results.

- `obj.ClearLookup ()` - Delete the associated fast lookup data structure on this array, if it exists. The lookup will be rebuilt on the next call to a lookup function.

30.12 vtkBox

30.12.1 Usage

vtkBox computes the implicit function and/or gradient for a axis-aligned bounding box. (The superclasses transform can be used to modify this orientation.) Each side of the box is orthogonal to all other sides meeting along shared edges and all faces are orthogonal to the x-y-z coordinate axes. (If you wish to orient this box differently, recall that the superclass vtkImplicitFunction supports a transformation matrix.) vtkCube is a concrete implementation of vtkImplicitFunction.

To create an instance of class vtkBox, simply invoke its constructor as follows

```
obj = vtkBox
```

30.12.2 Methods

The class vtkBox has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBox class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBox = obj.NewInstance ()`
- `vtkBox = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double n[3])`
- `obj.SetXMin (double p[3])` - Set / get the bounding box using various methods.
- `obj.SetXMin (double x, double y, double z)` - Set / get the bounding box using various methods.
- `obj.GetXMin (double p[3])` - Set / get the bounding box using various methods.
- `obj.SetXMax (double p[3])`
- `obj.SetXMax (double x, double y, double z)`
- `obj.GetXMax (double p[3])`
- `obj.SetBounds (double xMin, double xMax, double yMin, double yMax, double zMin, double zMax)`
- `obj.SetBounds (double bounds[6])`
- `obj.GetBounds (double bounds[6])`
- `obj.AddBounds (double bounds[6])` - A special method that allows union set operation on bounding boxes. Start with a SetBounds(). Subsequent AddBounds() methods are union set operations on the original bounds. Retrieve the final bounds with a GetBounds() method.

30.13 vtkBoxMuellerRandomSequence

30.13.1 Usage

vtkGaussianRandomSequence is a sequence of pseudo random numbers distributed according to the Gaussian/normal distribution (mean=0 and standard deviation=1).

It based is calculation from a uniformly distributed pseudo random sequence. The initial sequence is a vtkMinimalStandardRandomSequence.

To create an instance of class vtkBoxMuellerRandomSequence, simply invoke its constructor as follows

```
obj = vtkBoxMuellerRandomSequence
```

30.13.2 Methods

The class vtkBoxMuellerRandomSequence has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBoxMuellerRandomSequence class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBoxMuellerRandomSequence = obj.NewInstance ()`
- `vtkBoxMuellerRandomSequence = obj.SafeDownCast (vtkObject o)`
- `double = obj.GetValue ()` - Current value.
- `obj.Next ()` - Move to the next number in the random sequence.
- `vtkRandomSequence = obj.GetUniformSequence ()` - Return the uniformly distributed sequence of random numbers.
- `obj.SetUniformSequence (vtkRandomSequence uniformSequence)` - Set the uniformly distributed sequence of random numbers. Default is a .

30.14 vtkByteSwap

30.14.1 Usage

vtkByteSwap is used by other classes to perform machine dependent byte swapping. Byte swapping is often used when reading or writing binary files.

To create an instance of class vtkByteSwap, simply invoke its constructor as follows

```
obj = vtkByteSwap
```

30.14.2 Methods

The class vtkByteSwap has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkByteSwap class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkByteSwap = obj.NewInstance ()`
- `vtkByteSwap = obj.SafeDownCast (vtkObject o)`

30.15 vtkCharArray

30.15.1 Usage

vtkCharArray is an array of values of type char. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkCharArray, simply invoke its constructor as follows

```
obj = vtkCharArray
```

30.15.2 Methods

The class vtkCharArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCharArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCharArray = obj.NewInstance ()`
- `vtkCharArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, string tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, string tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, string tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (string tuple)` - Get the data at a particular index.
- `char = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, char value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, char f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (char f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `string = obj.WritePointer (vtkIdType id, vtkIdType number)` - Get the address of a particular data index. Performs no checks to verify that the memory has been allocated etc.
- `string = obj.GetPointer (vtkIdType id)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.

- `obj.SetArray (string array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (string array, vtkIdType size, int save, int deleteMethod)`

30.16 vtkCollection

30.16.1 Usage

vtkCollection is a general object for creating and manipulating lists of objects. The lists are unsorted and allow duplicate entries. vtkCollection also serves as a base class for lists of specific types of objects.

To create an instance of class vtkCollection, simply invoke its constructor as follows

```
obj = vtkCollection
```

30.16.2 Methods

The class vtkCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCollection = obj.NewInstance ()`
- `vtkCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkObject )` - Add an object to the list. Does not prevent duplicate entries.
- `obj.InsertItem (int i, vtkObject )` - Insert item into the list after the i'th item. Does not prevent duplicate entries. If i ¡ 0 the item is placed at the top of the list.
- `obj.ReplaceItem (int i, vtkObject )` - Replace the i'th item in the collection with a
- `obj.RemoveItem (int i)` - Remove the i'th item in the list. Be careful if using this function during traversal of the list using GetNextItemAsObject (or GetNextItem in derived class). The list WILL be shortened if a valid index is given! If this-¿Current is equal to the element being removed, have it point to then next element in the list.
- `obj.RemoveItem (vtkObject )` - Remove an object from the list. Removes the first object found, not all occurrences. If no object found, list is unaffected. See warning in description of RemoveItem(int).
- `obj.RemoveAllItems ()` - Remove all objects from the list.
- `int = obj.IsItemPresent (vtkObject a)` - Search for an object and return location in list. If the return value is 0, the object was not found. If the object was found, the location is the return value-1.
- `int = obj.GetNumberOfItems ()` - Return the number of objects in the list.
- `obj.InitTraversal ()` - Initialize the traversal of the collection. This means the data pointer is set at the beginning of the list.
- `vtkObject = obj.GetNextItemAsObject ()` - Get the next item in the collection. NULL is returned if the collection is exhausted.

- `vtkObject = obj.GetItemAsObject (int i)` - Get the i'th item in the collection. NULL is returned if i is out of range
- `vtkCollectionIterator = obj.NewIterator ()` - Get an iterator to traverse the objects in this collection.
- `obj.Register (vtkObjectBase o)` - Participate in garbage collection.
- `obj.UnRegister (vtkObjectBase o)` - Participate in garbage collection.

30.17 vtkCollectionIterator

30.17.1 Usage

vtkCollectionIterator provides an alternative way to traverse through the objects in a vtkCollection. Unlike the collection's built in interface, this allows multiple iterators to simultaneously traverse the collection. If items are removed from the collection, only the iterators currently pointing to those items are invalidated. Other iterators will still continue to function normally.

To create an instance of class vtkCollectionIterator, simply invoke its constructor as follows

```
obj = vtkCollectionIterator
```

30.17.2 Methods

The class vtkCollectionIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCollectionIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCollectionIterator = obj.NewInstance ()`
- `vtkCollectionIterator = obj.SafeDownCast (vtkObject o)`
- `obj.SetCollection (vtkCollection )` - Set/Get the collection over which to iterate.
- `vtkCollection = obj.GetCollection ()` - Set/Get the collection over which to iterate.
- `obj.InitTraversal ()` - Position the iterator at the first item in the collection.
- `obj.GoToFirstItem ()` - Position the iterator at the first item in the collection.
- `obj.GoToNextItem ()` - Move the iterator to the next item in the collection.
- `int = obj.IsDoneWithTraversal ()` - Test whether the iterator is currently positioned at a valid item. Returns 1 for yes, 0 for no.
- `vtkObject = obj.GetCurrentObject ()` - Get the item at the current iterator position. Valid only when IsDoneWithTraversal() returns 1.
- `vtkObject = obj.GetObject ()` - @deprecated Replaced by vtkCollectionIterator::GetCurrentObject() as of VTK 5.0.

30.18 vtkConditionVariable

30.18.1 Usage

vtkConditionVariable allows the locking of variables which are accessed through different threads. This header file also defines vtkSimpleConditionVariable which is not a subclass of vtkObject.

The win32 implementation is based on notes provided by Douglas C. Schmidt and Irfan Pyarali, Department of Computer Science, Washington University, St. Louis, Missouri. http://www.cs.wustl.edu/ schmidt/win32-cv-1.html

To create an instance of class vtkConditionVariable, simply invoke its constructor as follows

```
obj = vtkConditionVariable
```

30.18.2 Methods

The class vtkConditionVariable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkConditionVariable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkConditionVariable = obj.NewInstance ()`
- `vtkConditionVariable = obj.SafeDownCast (vtkObject o)`
- `obj.Signal ()` - Wake one thread waiting for the condition to change.
- `obj.Broadcast ()` - Wake all threads waiting for the condition to change.
- `int = obj.Wait (vtkMutexLock mutex)` - Wait for the condition to change. Upon entry, the mutex must be locked and the lock held by the calling thread. Upon exit, the mutex will be locked and held by the calling thread. Between entry and exit, the mutex will be unlocked and may be held by other threads.

  @param mutex The mutex that should be locked on entry and will be locked on exit (but not in between) @retval Normally, this function returns 0. Should a thread be interrupted by a signal, a non-zero value may be returned.

30.19 vtkContourValues

30.19.1 Usage

vtkContourValues is a general class to manage the creation, generation, and retrieval of contour values. This class serves as a helper class for contouring classes, or those classes operating on lists of contour values.

To create an instance of class vtkContourValues, simply invoke its constructor as follows

```
obj = vtkContourValues
```

30.19.2 Methods

The class vtkContourValues has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkContourValues class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkContourValues = obj.NewInstance ()`
- `vtkContourValues = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)` - Set the ith contour value.
- `double = obj.GetValue (int i)` - Get the ith contour value. The return value will be clamped if the index i is out of range.
- `obj.GetValues (double contourValues)` - Fill a supplied list with contour values. Make sure you've allocated memory of size GetNumberOfContours().
- `obj.SetNumberOfContours (int number)` - Set the number of contours to place into the list. You only really need to use this method to reduce list size. The method SetValue() will automatically increase list size as needed.
- `int = obj.GetNumberOfContours ()` - Return the number of contours in the
- `obj.GenerateValues (int numContours, double range[2])` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.

30.20 vtkCriticalSection

30.20.1 Usage

vtkCriticalSection allows the locking of variables which are accessed through different threads. This header file also defines vtkSimpleCriticalSection which is not a subclass of vtkObject. The API is identical to that of vtkMutexLock, and the behavior is identical as well, except on Windows 9x/NT platforms. The only difference on these platforms is that vtkMutexLock is more flexible, in that it works across processes as well as across threads, but also costs more, in that it evokes a 600-cycle x86 ring transition. The vtkCriticalSection provides a higher-performance equivalent (on Windows) but won't work across processes. Since it is unclear how, in vtk, an object at the vtk level can be shared across processes in the first place, one should use vtkCriticalSection unless one has a very good reason to use vtkMutexLock. If higher-performance equivalents for non-Windows platforms (Irix, SunOS, etc) are discovered, they should replace the implementations in this class

To create an instance of class vtkCriticalSection, simply invoke its constructor as follows

```
obj = vtkCriticalSection
```

30.20.2 Methods

The class vtkCriticalSection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCriticalSection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCriticalSection = obj.NewInstance ()`
- `vtkCriticalSection = obj.SafeDownCast (vtkObject o)`

- `obj.Lock ()` - Lock the vtkCriticalSection
- `obj.Unlock ()` - Unlock the vtkCriticalSection

30.21 vtkCylindricalTransform

30.21.1 Usage

vtkCylindricalTransform will convert (r,theta,z) coordinates to (x,y,z) coordinates and back again. The angles are given in radians. By default, it converts cylindrical coordinates to rectangular, but GetInverse() returns a transform that will do the opposite. The equation that is used is x = r*cos(theta), y = r*sin(theta), z = z.

To create an instance of class vtkCylindricalTransform, simply invoke its constructor as follows

```
obj = vtkCylindricalTransform
```

30.21.2 Methods

The class vtkCylindricalTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCylindricalTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCylindricalTransform = obj.NewInstance ()`
- `vtkCylindricalTransform = obj.SafeDownCast (vtkObject o)`
- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.

30.22 vtkDataArray

30.22.1 Usage

vtkDataArray is an abstract superclass for data array objects containing numeric data. It extends the API defined in vtkAbstractArray. vtkDataArray is an abstract superclass for data array objects. This class defines an API that all array objects must support. Note that the concrete subclasses of this class represent data in native form (char, int, etc.) and often have specialized more efficient methods for operating on this data (for example, getting pointers to data or getting/inserting data in native form). Subclasses of vtkDataArray are assumed to contain data whose components are meaningful when cast to and from double.

To create an instance of class vtkDataArray, simply invoke its constructor as follows

```
obj = vtkDataArray
```

30.22.2 Methods

The class vtkDataArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkDataArray = obj.NewInstance ()`
- `vtkDataArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.IsNumeric ()` - Return the size, in bytes, of the lowest-level element of an array. For vtkDataArray and subclasses this is the size of the data type.
- `int = obj.GetElementComponentSize ()` - Set the tuple at the ith location using the jth tuple in the source array. This method assumes that the two arrays have the same type and structure. Note that range checking and memory allocation is not performed; use in conjunction with SetNumberOfTuples() to allocate space.
- `obj.SetTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Set the tuple at the ith location using the jth tuple in the source array. This method assumes that the two arrays have the same type and structure. Note that range checking and memory allocation is not performed; use in conjunction with SetNumberOfTuples() to allocate space.
- `obj.InsertTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at ith location in this array. Note that memory allocation is performed as necessary to hold the data. This pure virtual function is redeclared here to avoid declaration hidden warnings.
- `vtkIdType = obj.InsertNextTuple (vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at the end in this array. Note that memory allocation is performed as necessary to hold the data. Returns the location at which the data was inserted. This pure virtual function is redeclared here to avoid declaration hidden warnings.
- `obj.GetTuples (vtkIdList ptIds, vtkAbstractArray output)` - Given a list of point ids, return an array of tuples. You must insure that the output array has been previously allocated with enough space to hold the data.
- `obj.GetTuples (vtkIdType p1, vtkIdType p2, vtkAbstractArray output)` - Get the tuples for the range of points ids specified (i.e., p1-¿p2 inclusive). You must insure that the output array has been previously allocated with enough space to hold the data.
- `obj.InterpolateTuple (vtkIdType i, vtkIdList ptIndices, vtkAbstractArray source, double weights)` - Set the ith tuple in this array as the interpolated tuple value, given the ptIndices in the source array and associated interpolation weights. This method assumes that the two arrays are of the same type and strcuture.
- `obj.InterpolateTuple (vtkIdType i, vtkIdType id1, vtkAbstractArray source1, vtkIdType id2, vtkAbstr`
- `obj.GetTuple (vtkIdType i, double tuple)` - Get the data tuple at ith location by filling in a user-provided array, Make sure that your array is large enough to hold the NumberOfComponents amount of data being returned.
- `double = obj.GetTuple1 (vtkIdType i)` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.
- `double = obj.GetTuple2 (vtkIdType i)` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.
- `double = obj.GetTuple3 (vtkIdType i)` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.
- `double = obj.GetTuple4 (vtkIdType i)` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.

- `double = obj.GetTuple9 (vtkIdType i)` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.SetTuple (vtkIdType i, float tuple)` - Set the data tuple at ith location. Note that range checking or memory allocation is not performed; use this method in conjunction with SetNumberOfTuples() to allocate space.

- `obj.SetTuple (vtkIdType i, double tuple)` - Set the data tuple at ith location. Note that range checking or memory allocation is not performed; use this method in conjunction with SetNumberOfTuples() to allocate space.

- `obj.SetTuple1 (vtkIdType i, double value)` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.SetTuple2 (vtkIdType i, double val0, double val1)` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.SetTuple3 (vtkIdType i, double val0, double val1, double val2)` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.SetTuple4 (vtkIdType i, double val0, double val1, double val2, double val3)` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.SetTuple9 (vtkIdType i, double val0, double val1, double val2, double val3, double val4, double` - These methods are included as convenience for the wrappers. GetTuple() and SetTuple() which return/take arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.InsertTuple (vtkIdType i, float tuple)` - Insert the data tuple at ith location. Note that memory allocation is performed as necessary to hold the data.

- `obj.InsertTuple (vtkIdType i, double tuple)` - Insert the data tuple at ith location. Note that memory allocation is performed as necessary to hold the data.

- `obj.InsertTuple1 (vtkIdType i, double value)` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.InsertTuple2 (vtkIdType i, double val0, double val1)` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.InsertTuple3 (vtkIdType i, double val0, double val1, double val2)` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.InsertTuple4 (vtkIdType i, double val0, double val1, double val2, double val3)` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.InsertTuple9 (vtkIdType i, double val0, double val1, double val2, double val3, double val4, dou` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `vtkIdType = obj.InsertNextTuple (float tuple)` - Insert the data tuple at the end of the array and return the location at which the data was inserted. Memory is allocated as necessary to hold the data.

- `vtkIdType = obj.InsertNextTuple (double tuple)` - Insert the data tuple at the end of the array and return the location at which the data was inserted. Memory is allocated as necessary to hold the data.

- `obj.InsertNextTuple1 (double value)` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.InsertNextTuple2 (double val0, double val1)` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.InsertNextTuple3 (double val0, double val1, double val2)` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.InsertNextTuple4 (double val0, double val1, double val2, double val3)` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.InsertNextTuple9 (double val0, double val1, double val2, double val3, double val4, double val5,` - These methods are included as convenience for the wrappers. InsertTuple() which takes arrays can not be used from wrapped languages. These methods can be used instead.

- `obj.RemoveTuple (vtkIdType id)` - These methods remove tuples from the data array. They shift data and resize array, so the data array is still valid after this operation. Note, this operation is fairly slow.

- `obj.RemoveFirstTuple ()` - These methods remove tuples from the data array. They shift data and resize array, so the data array is still valid after this operation. Note, this operation is fairly slow.

- `obj.RemoveLastTuple ()` - These methods remove tuples from the data array. They shift data and resize array, so the data array is still valid after this operation. Note, this operation is fairly slow.

- `double = obj.GetComponent (vtkIdType i, int j)` - Return the data component at the ith tuple and jth component location. Note that i is less than NumberOfTuples and j is less than NumberOfComponents.

- `obj.SetComponent (vtkIdType i, int j, double c)` - Set the data component at the ith tuple and jth component location. Note that i is less than NumberOfTuples and j is less than NumberOfComponents. Make sure enough memory has been allocated (use SetNumberOfTuples() and SetNumberOfComponents()).

- `obj.InsertComponent (vtkIdType i, int j, double c)` - Insert the data component at ith tuple and jth component location. Note that memory allocation is performed as necessary to hold the data.

- `obj.GetData (vtkIdType tupleMin, vtkIdType tupleMax, int compMin, int compMax, vtkDoubleArray data)` - Get the data as a double array in the range (tupleMin,tupleMax) and (compMin, compMax). The resulting double array consists of all data in the tuple range specified and only the component range specified. This process typically requires casting the data from native form into doubleing point values. This method is provided as a convenience for data exchange, and is not very fast.

- `obj.DeepCopy (vtkAbstractArray aa)` - Deep copy of data. Copies data from different data arrays even if they are different types (using doubleing-point exchange).

- `obj.DeepCopy (vtkDataArray da)` - Deep copy of data. Copies data from different data arrays even if they are different types (using doubleing-point exchange).

- `obj.FillComponent (int j, double c)` - Fill a component of a data array with a specified value. This method sets the specified component to specified value for all tuples in the data array. This methods can be used to initialize or reinitialize a single component of a multi-component array.

- `obj.CopyComponent (int j, vtkDataArray from, int fromComponent)` - Copy a component from one data array into a component on this data array. This method copies the specified component ("fromComponent") from the specified data array ("from") to the specified component ("j") over all the tuples in this data array. This method can be used to extract a component (column) from one data array and paste that data into a component on this data array.

- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this data array. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.

- `obj.CreateDefaultLookupTable ()` - Create default lookup table. Generally used to create one when none is available.

- `obj.SetLookupTable (vtkLookupTable lut)` - Set/get the lookup table associated with this scalar data, if any.

- `vtkLookupTable = obj.GetLookupTable ()` - Set/get the lookup table associated with this scalar data, if any.

- `obj.GetRange (double range[2], int comp)` - Return the range of the array values for the given component. Range is copied into the array provided. If comp is equal to -1, it returns the range of the magnitude (if the number of components is equal to 1 it still returns the range of component 0).

- `double = obj.GetRange (int comp)` - Return the range of the array values for the 0th component. Range is copied into the array provided.

- `double = obj.GetRange ()` - Return the range of the array values for the 0th component. Range is copied into the array provided.

- `obj.GetRange (double range[2])` - These methods return the Min and Max possible range of the native data type. For example if a vtkScalars consists of unsigned char data these will return (0,255).

- `obj.GetDataTypeRange (double range[2])` - These methods return the Min and Max possible range of the native data type. For example if a vtkScalars consists of unsigned char data these will return (0,255).

- `double = obj.GetDataTypeMin ()` - These methods return the Min and Max possible range of the native data type. For example if a vtkScalars consists of unsigned char data these will return (0,255).

- `double = obj.GetDataTypeMax ()` - These methods return the Min and Max possible range of the native data type. For example if a vtkScalars consists of unsigned char data these will return (0,255).

- `double = obj.GetMaxNorm ()` - Return the maximum norm for the tuples. Note that the max. is computed everytime GetMaxNorm is called.

- `int = obj.CopyInformation (vtkInformation infoFrom, int deep)` - Copy information instance. Arrays use information objects in a variety of ways. It is important to have flexibility in this regard because certain keys should not be coppied, while others must be. NOTE: Up to the implmeneter to make sure that keys not inteneded to be coppied are excluded here.

30.23 vtkDataArrayCollection

30.23.1 Usage

vtkDataArrayCollection is an object that creates and manipulates lists of datasets. See also vtkCollection and subclasses.

To create an instance of class vtkDataArrayCollection, simply invoke its constructor as follows

```
obj = vtkDataArrayCollection
```

30.23.2 Methods

The class vtkDataArrayCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataArrayCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataArrayCollection = obj.NewInstance ()`
- `vtkDataArrayCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkDataArray ds)` - Get the next dataarray in the list.
- `vtkDataArray = obj.GetNextItem ()` - Get the next dataarray in the list.
- `vtkDataArray = obj.GetItem (int i)` - Get the ith dataarray in the list.

30.24 vtkDataArrayCollectionIterator

30.24.1 Usage

vtkDataArrayCollectionIterator provides an implementation of vtkCollectionIterator which allows the items to be retrieved with the proper subclass pointer type for vtkDataArrayCollection.

To create an instance of class vtkDataArrayCollectionIterator, simply invoke its constructor as follows

```
obj = vtkDataArrayCollectionIterator
```

30.24.2 Methods

The class vtkDataArrayCollectionIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataArrayCollectionIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataArrayCollectionIterator = obj.NewInstance ()`
- `vtkDataArrayCollectionIterator = obj.SafeDownCast (vtkObject o)`
- `obj.SetCollection (vtkCollection )` - Set the collection over which to iterate.
- `obj.SetCollection (vtkDataArrayCollection )` - Set the collection over which to iterate.
- `vtkDataArray = obj.GetDataArray ()` - Get the item at the current iterator position. Valid only when IsDoneWithTraversal() returns 1.

30.25 vtkDataArraySelection

30.25.1 Usage

vtkDataArraySelection can be used by vtkSource subclasses to store on/off settings for whether each vtkDataArray in its input should be passed in the source's output. This is primarily intended to allow file readers to configure what data arrays are read from the file.

To create an instance of class vtkDataArraySelection, simply invoke its constructor as follows

```
obj = vtkDataArraySelection
```

30.25.2 Methods

The class vtkDataArraySelection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataArraySelection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataArraySelection = obj.NewInstance ()`
- `vtkDataArraySelection = obj.SafeDownCast (vtkObject o)`
- `obj.EnableArray (string name)` - Enable the array with the given name. Creates a new entry if none exists.
- `obj.DisableArray (string name)` - Disable the array with the given name. Creates a new entry if none exists.
- `int = obj.ArrayIsEnabled (string name)` - Return whether the array with the given name is enabled. If there is no entry, the array is assumed to be disabled.
- `int = obj.ArrayExists (string name)` - Return whether the array with the given name exists.
- `obj.EnableAllArrays ()` - Enable all arrays that currently have an entry.
- `obj.DisableAllArrays ()` - Disable all arrays that currently have an entry.
- `int = obj.GetNumberOfArrays ()` - Get the number of arrays that currently have an entry.
- `int = obj.GetNumberOfArraysEnabled ()` - Get the number of arrays that are enabled.
- `string = obj.GetArrayName (int index)` - Get the name of the array entry at the given index.
- `int = obj.GetArrayIndex (string name)` - Get an index of the array containing name within the enabled arrays
- `int = obj.GetEnabledArrayIndex (string name)` - Get the index of an array with the given name among those that are enabled. Returns -1 if the array is not enabled.
- `int = obj.GetArraySetting (string name)` - Get whether the array at the given index is enabled.
- `int = obj.GetArraySetting (int index)` - Get whether the array at the given index is enabled.
- `obj.RemoveAllArrays ()` - Remove all array entries.
- `obj.CopySelections (vtkDataArraySelection selections)` - Copy the selections from the given vtkDataArraySelection instance.

30.26 vtkDebugLeaks

30.26.1 Usage

vtkDebugLeaks is used to report memory leaks at the exit of the program. It uses the vtkObjectFactory to intercept the construction of all VTK objects. It uses the UnRegister method of vtkObject to intercept the destruction of all objects. A table of object name to number of instances is kept. At the exit of the program if there are still VTK objects around it will print them out. To enable this class add the flag -DVTK_DEBUG_LEAKS to the compile line, and rebuild vtkObject and vtkObjectFactory.

To create an instance of class vtkDebugLeaks, simply invoke its constructor as follows

```
obj = vtkDebugLeaks
```

30.26.2 Methods

The class vtkDebugLeaks has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDebugLeaks class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDebugLeaks = obj.NewInstance ()`
- `vtkDebugLeaks = obj.SafeDownCast (vtkObject o)`

30.27 vtkDirectory

30.27.1 Usage

vtkDirectory provides a portable way of finding the names of the files in a system directory. It also provides methods of manipulating directories.

To create an instance of class vtkDirectory, simply invoke its constructor as follows

```
obj = vtkDirectory
```

30.27.2 Methods

The class vtkDirectory has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDirectory class.

- `string = obj.GetClassName ()` - Return the class name as a string.
- `int = obj.IsA (string name)` - Return the class name as a string.
- `vtkDirectory = obj.NewInstance ()` - Return the class name as a string.
- `vtkDirectory = obj.SafeDownCast (vtkObject o)` - Return the class name as a string.
- `int = obj.Open (string dir)` - Open the specified directory and load the names of the files in that directory. 0 is returned if the directory can not be opened, 1 if it is opened.
- `vtkIdType = obj.GetNumberOfFiles ()` - Return the number of files in the current directory.
- `string = obj.GetFile (vtkIdType index)` - Return the file at the given index, the indexing is 0 based

- `int = obj.FileIsDirectory (string name)` - Return true if the file is a directory. If the file is not an absolute path, it is assumed to be relative to the opened directory. If no directory has been opened, it is assumed to be relative to the current working directory.
- `vtkStringArray = obj.GetFiles ()` - Get an array that contains all the file names.

30.28 vtkDoubleArray

30.28.1 Usage

vtkDoubleArray is an array of values of type double. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkDoubleArray, simply invoke its constructor as follows

```
obj = vtkDoubleArray
```

30.28.2 Methods

The class vtkDoubleArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDoubleArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDoubleArray = obj.NewInstance ()`
- `vtkDoubleArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, double tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, double tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, double tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (double tuple)` - Get the data at a particular index.
- `double = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, double value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, double f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (double f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.

- `obj.SetArray (double array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (double array, vtkIdType size, int save, int deleteMethod)`

30.29 vtkDynamicLoader

30.29.1 Usage

vtkDynamicLoader provides a portable interface to loading dynamic libraries into a process.

To create an instance of class vtkDynamicLoader, simply invoke its constructor as follows

```
obj = vtkDynamicLoader
```

30.29.2 Methods

The class vtkDynamicLoader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDynamicLoader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDynamicLoader = obj.NewInstance ()`
- `vtkDynamicLoader = obj.SafeDownCast (vtkObject o)`

30.30 vtkEdgeTable

30.30.1 Usage

vtkEdgeTable is a general object for keeping track of lists of edges. An edge is defined by the pair of point id's (p1,p2). Methods are available to insert edges, check if edges exist, and traverse the list of edges. Also, it's possible to associate attribute information with each edge. The attribute information may take the form of vtkIdType id's, void* pointers, or points. To store attributes, make sure that InitEdgeInsertion() is invoked with the storeAttributes flag set properly. If points are inserted, use the methods InitPointInsertion() and InsertUniquePoint().

To create an instance of class vtkEdgeTable, simply invoke its constructor as follows

```
obj = vtkEdgeTable
```

30.30.2 Methods

The class vtkEdgeTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEdgeTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEdgeTable = obj.NewInstance ()`

- `vtkEdgeTable = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Free memory and return to the initially instantiated state.
- `int = obj.InitEdgeInsertion (vtkIdType numPoints, int storeAttributes)` - Initialize the edge insertion process. Provide an estimate of the number of points in a dataset (the maximum range value of p1 or p2). The storeAttributes variable controls whether attributes are to be stored with the edge, and what type of attributes. If storeAttributes==1, then attributes of vtkIdType can be stored. If storeAttributes==2, then attributes of type void* can be stored. In either case, additional memory will be required by the data structure to store attribute data per each edge. This method is used in conjunction with one of the three InsertEdge() methods described below (don't mix the InsertEdge() methods—make sure that the one used is consistent with the storeAttributes flag set in InitEdgeInsertion()).
- `vtkIdType = obj.InsertEdge (vtkIdType p1, vtkIdType p2)` - Insert the edge (p1,p2) into the table. It is the user's responsibility to check if the edge has already been inserted (use IsEdge()). If the storeAttributes flag in InitEdgeInsertion() has been set, then the method returns a unique integer id (i.e., the edge id) that can be used to set and get edge attributes. Otherwise, the method will return 1. Do not mix this method with the InsertEdge() method that follows.
- `obj.InsertEdge (vtkIdType p1, vtkIdType p2, vtkIdType attributeId)` - Insert the edge (p1,p2) into the table with the attribute id specified (make sure the attributeId ¿= 0). Note that the attributeId is ignored if the storeAttributes variable was set to 0 in the InitEdgeInsertion() method. It is the user's responsibility to check if the edge has already been inserted (use IsEdge()). Do not mix this method with the other two InsertEdge() methods.
- `vtkIdType = obj.IsEdge (vtkIdType p1, vtkIdType p2)` - Return an integer id for the edge, or an attribute id of the edge (p1,p2) if the edge has been previously defined (it depends upon which version of InsertEdge() is being used); otherwise -1. The unique integer id can be used to set and retrieve attributes to the edge.
- `int = obj.InitPointInsertion (vtkPoints newPts, vtkIdType estSize)` - Initialize the point insertion process. The newPts is an object representing point coordinates into which incremental insertion methods place their data. The points are associated with the edge.
- `vtkIdType = obj.GetNumberOfEdges ()` - Return the number of edges that have been inserted thus far.
- `obj.InitTraversal ()` - Intialize traversal of edges in table.
- `obj.Reset ()` - Reset the object and prepare for reinsertion of edges. Does not delete memory like the Initialize() method.

30.31 vtkExtentSplitter

30.31.1 Usage

vtkExtentSplitter splits each input extent into non-overlapping sub-extents that are completely contained within other "source extents". A source extent corresponds to some resource providing an extent. Each source extent has an integer identifier, integer priority, and an extent. The input extents are split into sub-extents according to priority, availability, and amount of overlap of the source extents. This can be used by parallel data readers to read as few piece files as possible.

To create an instance of class vtkExtentSplitter, simply invoke its constructor as follows

```
obj = vtkExtentSplitter
```

30.31.2 Methods

The class vtkExtentSplitter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtentSplitter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtentSplitter = obj.NewInstance ()`
- `vtkExtentSplitter = obj.SafeDownCast (vtkObject o)`
- `obj.AddExtentSource (int id, int priority, int x0, int x1, int y0, int y1, int z0, int z1)` - Add/Remove a source providing the given extent. Sources with higher priority numbers are favored. Source id numbers and priorities must be non-negative.
- `obj.AddExtentSource (int id, int priority, int extent)` - Add/Remove a source providing the given extent. Sources with higher priority numbers are favored. Source id numbers and priorities must be non-negative.
- `obj.RemoveExtentSource (int id)` - Add/Remove a source providing the given extent. Sources with higher priority numbers are favored. Source id numbers and priorities must be non-negative.
- `obj.RemoveAllExtentSources ()` - Add/Remove a source providing the given extent. Sources with higher priority numbers are favored. Source id numbers and priorities must be non-negative.
- `obj.AddExtent (int x0, int x1, int y0, int y1, int z0, int z1)` - Add an extent to the queue of extents to be split among the available sources.
- `obj.AddExtent (int extent)` - Add an extent to the queue of extents to be split among the available sources.
- `int = obj.ComputeSubExtents ()` - Split the extents currently in the queue among the available sources. The queue is empty when this returns. Returns 1 if all extents could be read. Returns 0 if any portion of any extent was not available through any source.
- `int = obj.GetNumberOfSubExtents ()` - Get the number of sub-extents into which the original set of extents have been split across the available sources. Valid after a call to ComputeSubExtents.
- `int = obj.GetSubExtent (int index)` - Get the sub-extent associated with the given index. Use GetSubExtentSource to get the id of the source from which this sub-extent should be read. Valid after a call to ComputeSubExtents.
- `obj.GetSubExtent (int index, int extent)` - Get the sub-extent associated with the given index. Use GetSubExtentSource to get the id of the source from which this sub-extent should be read. Valid after a call to ComputeSubExtents.
- `int = obj.GetSubExtentSource (int index)` - Get the id of the source from which the sub-extent associated with the given index should be read. Returns -1 if no source provides the sub-extent.
- `int = obj.GetPointMode ()` - Get/Set whether "point mode" is on. In point mode, sub-extents are generated to ensure every point in the update request is read, but not necessarily every cell. This can be used when point data are stored in a planar slice per piece with no cell data. The default is OFF.
- `obj.SetPointMode (int )` - Get/Set whether "point mode" is on. In point mode, sub-extents are generated to ensure every point in the update request is read, but not necessarily every cell. This can be used when point data are stored in a planar slice per piece with no cell data. The default is OFF.

- `obj.PointModeOn ()` - Get/Set whether "point mode" is on. In point mode, sub-extents are generated to ensure every point in the update request is read, but not necessarily every cell. This can be used when point data are stored in a planar slice per piece with no cell data. The default is OFF.
- `obj.PointModeOff ()` - Get/Set whether "point mode" is on. In point mode, sub-extents are generated to ensure every point in the update request is read, but not necessarily every cell. This can be used when point data are stored in a planar slice per piece with no cell data. The default is OFF.

30.32 vtkExtentTranslator

30.32.1 Usage

vtkExtentTranslator generates a structured extent from an unstructured extent. It uses a recursive scheme that splits the largest axis. A hard coded extent can be used for a starting point.

To create an instance of class vtkExtentTranslator, simply invoke its constructor as follows

```
obj = vtkExtentTranslator
```

30.32.2 Methods

The class vtkExtentTranslator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtentTranslator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtentTranslator = obj.NewInstance ()`
- `vtkExtentTranslator = obj.SafeDownCast (vtkObject o)`
- `obj.SetWholeExtent (int , int , int , int , int , int )` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `obj.SetWholeExtent (int a[6])` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `int = obj. GetWholeExtent ()` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `obj.SetExtent (int , int , int , int , int , int )` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `obj.SetExtent (int a[6])` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `int = obj. GetExtent ()` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `obj.SetPiece (int )` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `int = obj.GetPiece ()` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `obj.SetNumberOfPieces (int )` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.

- `int = obj.GetNumberOfPieces ()` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `obj.SetGhostLevel (int )` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `int = obj.GetGhostLevel ()` - Set the Piece/NumPieces. Set the WholeExtent and then call PieceToExtent. The result can be obtained from the Extent ivar.
- `int = obj.PieceToExtent ()` - These are the main methods that should be called. These methods are responsible for converting a piece to an extent. The signatures without arguments are only thread safe when each thread accesses a different instance. The signatures with arguements are fully thread safe.
- `int = obj.PieceToExtentByPoints ()` - These are the main methods that should be called. These methods are responsible for converting a piece to an extent. The signatures without arguments are only thread safe when each thread accesses a different instance. The signatures with arguements are fully thread safe.
- `int = obj.PieceToExtentThreadSafe (int piece, int numPieces, int ghostLevel, int wholeExtent, int r` - These are the main methods that should be called. These methods are responsible for converting a piece to an extent. The signatures without arguments are only thread safe when each thread accesses a different instance. The signatures with arguements are fully thread safe.
- `obj.SetSplitModeToBlock ()` - How should the streamer break up extents. Block mode tries to break an extent up into cube blocks. It always chooses the largest axis to split. Slab mode first breaks up the Z axis. If it gets to one slice, then it starts breaking up other axes.
- `obj.SetSplitModeToXSlab ()` - How should the streamer break up extents. Block mode tries to break an extent up into cube blocks. It always chooses the largest axis to split. Slab mode first breaks up the Z axis. If it gets to one slice, then it starts breaking up other axes.
- `obj.SetSplitModeToYSlab ()` - How should the streamer break up extents. Block mode tries to break an extent up into cube blocks. It always chooses the largest axis to split. Slab mode first breaks up the Z axis. If it gets to one slice, then it starts breaking up other axes.
- `obj.SetSplitModeToZSlab ()` - How should the streamer break up extents. Block mode tries to break an extent up into cube blocks. It always chooses the largest axis to split. Slab mode first breaks up the Z axis. If it gets to one slice, then it starts breaking up other axes.
- `int = obj.GetSplitMode ()` - How should the streamer break up extents. Block mode tries to break an extent up into cube blocks. It always chooses the largest axis to split. Slab mode first breaks up the Z axis. If it gets to one slice, then it starts breaking up other axes.
- `obj.SetSplitPath (int len, int splitpath)`

30.33 vtkFastNumericConversion

30.33.1 Usage

vtkFastNumericConversion uses a portable (assuming IEEE format) method for converting single and double precision floating point values to a fixed point representation. This allows fast integer floor operations on platforms, such as Intel X86, in which CPU floating point conversion algorithms are very slow. It is based on the techniques described in Chris Hecker's article, "Let's Get to the (Floating) Point", in Game Developer Magazine, Feb/Mar 1996, and the techniques described in Michael Herf's website, http://www.stereopsis.com/FPU.html. The Hecker article can be found at http://www.d6.com/users/checker/pdfs/gdmfp.pd Unfortunately, each of these techniques is incomplete, and doesn't convert properly, in a way that depends on how many bits are reserved for fixed point fractional use, due to failing to properly account for the default

round-towards-even rounding mode of the X86. Thus, my implementation incorporates some rounding correction that undoes the rounding that the FPU performs during denormalization of the floating point value. Note that the rounding affect I'm talking about here is not the effect on the fistp instruction, but rather the effect that occurs during the denormalization of a value that occurs when adding it to a much larger value. The bits must be shifted to the right, and when a "1" bit falls off the edge, the rounding mode determines what happens next, in order to avoid completely "losing" the 1-bit. Furthermore, my implementation works on Linux, where the default precision mode is 64-bit extended precision.

To create an instance of class vtkFastNumericConversion, simply invoke its constructor as follows

```
obj = vtkFastNumericConversion
```

30.33.2 Methods

The class vtkFastNumericConversion has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFastNumericConversion class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFastNumericConversion = obj.NewInstance ()`
- `vtkFastNumericConversion = obj.SafeDownCast (vtkObject o)`
- `int = obj.TestQuickFloor (double val)` - Wrappable method for script-testing of correct cross-platform functionality
- `int = obj.TestSafeFloor (double val)` - Wrappable method for script-testing of correct cross-platform functionality
- `int = obj.TestRound (double val)` - Wrappable method for script-testing of correct cross-platform functionality
- `int = obj.TestConvertFixedPointIntPart (double val)` - Wrappable method for script-testing of correct cross-platform functionality
- `int = obj.TestConvertFixedPointFracPart (double val)` - Wrappable method for script-testing of correct cross-platform functionality
- `obj.SetReservedFracBits (int bits)` - Set the number of bits reserved for fractional precision that are maintained as part of the flooring process. This number affects the flooring arithmetic. It may be useful if the factional part is to be used to index into a lookup table of some sort. However, if you are only interested in knowing the fractional remainder after flooring, there doesn't appear to be any advantage to using these bits, either in terms of a lookup table, or by directly multiplying by some unit fraction, over simply subtracting the floored value from the original value. Note that since only 32 bits are used for the entire fixed point representation, increasing the number of reserved fractional bits reduces the range of integer values that can be floored to. Add one to the requested number of fractional bits, to make the conversion safe with respect to rounding mode. This is the same as the difference between QuickFloor and SafeFloor.
- `obj.PerformanceTests (void )` - Conduct timing tests so that the usefulness of this class can be ascertained on whatever platform it is being used. Output can be retrieved via Print method.

30.34 vtkFileOutputWindow

30.34.1 Usage

Writes debug/warning/error output to a log file instead of the console. To use this class, instantiate it and then call SetInstance(this).

To create an instance of class vtkFileOutputWindow, simply invoke its constructor as follows

```
obj = vtkFileOutputWindow
```

30.34.2 Methods

The class vtkFileOutputWindow has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFileOutputWindow class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFileOutputWindow = obj.NewInstance ()`
- `vtkFileOutputWindow = obj.SafeDownCast (vtkObject o)`
- `obj.DisplayText (string )` - Put the text into the log file. New lines are converted to carriage return new lines.
- `obj.SetFileName (string )` - Sets the name for the log file.
- `string = obj.GetFileName ()` - Sets the name for the log file.
- `obj.SetFlush (int )` - Turns on buffer flushing for the output to the log file.
- `int = obj.GetFlush ()` - Turns on buffer flushing for the output to the log file.
- `obj.FlushOn ()` - Turns on buffer flushing for the output to the log file.
- `obj.FlushOff ()` - Turns on buffer flushing for the output to the log file.
- `obj.SetAppend (int )` - Setting append will cause the log file to be opened in append mode. Otherwise, if the log file exists, it will be overwritten each time the vtkFileOutputWindow is created.
- `int = obj.GetAppend ()` - Setting append will cause the log file to be opened in append mode. Otherwise, if the log file exists, it will be overwritten each time the vtkFileOutputWindow is created.
- `obj.AppendOn ()` - Setting append will cause the log file to be opened in append mode. Otherwise, if the log file exists, it will be overwritten each time the vtkFileOutputWindow is created.
- `obj.AppendOff ()` - Setting append will cause the log file to be opened in append mode. Otherwise, if the log file exists, it will be overwritten each time the vtkFileOutputWindow is created.

30.35 vtkFloatArray

30.35.1 Usage

vtkFloatArray is an array of values of type float. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkFloatArray, simply invoke its constructor as follows

```
obj = vtkFloatArray
```

30.35.2 Methods

The class vtkFloatArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFloatArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFloatArray = obj.NewInstance ()`
- `vtkFloatArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, float tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, float tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, float tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (float tuple)` - Get the data at a particular index.
- `float = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, float value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, float f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (float f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `obj.SetArray (float array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (float array, vtkIdType size, int save, int deleteMethod)`

30.36 vtkFunctionParser

30.36.1 Usage

vtkFunctionParser is a class that takes in a mathematical expression as a char string, parses it, and evaluates it at the specified values of the variables in the input string.

You can use the "if" operator to create conditional expressions such as if (test, trueresult, falseresult). These evaluate the boolean valued test expression and then evaluate either the trueresult or the falseresult

expression to produce a final (scalar or vector valued) value. "test" may contain ¡,¿,=,—,&, and () and all three subexpressions can evaluate arbitrary function operators (ln, cos, +, if, etc)

.SECTION Thanks Thomas Dunne (thomas.dunne@iwr.uni-heidelberg.de) for adding code for two-parameter-parsing and a few functions (sign, min, max).

Sid Sydoriak (sxs@lanl.gov) for adding boolean operations and conditional expressions and for fixing a variety of bugs.

To create an instance of class vtkFunctionParser, simply invoke its constructor as follows

```
obj = vtkFunctionParser
```

30.36.2 Methods

The class vtkFunctionParser has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFunctionParser class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFunctionParser = obj.NewInstance ()`
- `vtkFunctionParser = obj.SafeDownCast (vtkObject o)`
- `obj.SetFunction (string function)`
- `string = obj.GetFunction ()`
- `int = obj.IsScalarResult ()` - Check whether the result is a scalar result. If it isn't, then either the result is a vector or an error has occurred.
- `int = obj.IsVectorResult ()` - Check whether the result is a vector result. If it isn't, then either the result is scalar or an error has occurred.
- `double = obj.GetScalarResult ()` - Get a scalar result from evaluating the input function.
- `double = obj.GetVectorResult ()` - Get a vector result from evaluating the input function.
- `obj.GetVectorResult (double result[3])` - Get a vector result from evaluating the input function.
- `obj.SetScalarVariableValue (string variableName, double value)` - Set the value of a scalar variable. If a variable with this name exists, then its value will be set to the new value. If there is not already a variable with this name, variableName will be added to the list of variables, and its value will be set to the new value.
- `obj.SetScalarVariableValue (int i, double value)` - Set the value of a scalar variable. If a variable with this name exists, then its value will be set to the new value. If there is not already a variable with this name, variableName will be added to the list of variables, and its value will be set to the new value.
- `double = obj.GetScalarVariableValue (string variableName)` - Get the value of a scalar variable.
- `double = obj.GetScalarVariableValue (int i)` - Get the value of a scalar variable.
- `obj.SetVectorVariableValue (string variableName, double xValue, double yValue, double zValue)` - Set the value of a vector variable. If a variable with this name exists, then its value will be set to the new value. If there is not already a variable with this name, variableName will be added to the list of variables, and its value will be set to the new value.

- `obj.SetVectorVariableValue (string variableName, double values[3])` - Set the value of a vector variable. If a variable with this name exists, then its value will be set to the new value. If there is not already a variable with this name, variableName will be added to the list of variables, and its value will be set to the new value.
- `obj.SetVectorVariableValue (int i, double xValue, double yValue, double zValue)` - Set the value of a vector variable. If a variable with this name exists, then its value will be set to the new value. If there is not already a variable with this name, variableName will be added to the list of variables, and its value will be set to the new value.
- `obj.SetVectorVariableValue (int i, double values[3])` - Set the value of a vector variable. If a variable with this name exists, then its value will be set to the new value. If there is not already a variable with this name, variableName will be added to the list of variables, and its value will be set to the new value.
- `double = obj.GetVectorVariableValue (string variableName)` - Get the value of a vector variable.
- `obj.GetVectorVariableValue (string variableName, double value[3])` - Get the value of a vector variable.
- `double = obj.GetVectorVariableValue (int i)` - Get the value of a vector variable.
- `obj.GetVectorVariableValue (int i, double value[3])` - Get the value of a vector variable.
- `int = obj.GetNumberOfScalarVariables ()` - Get the number of scalar variables.
- `int = obj.GetNumberOfVectorVariables ()` - Get the number of vector variables.
- `string = obj.GetScalarVariableName (int i)` - Get the ith scalar variable name.
- `string = obj.GetVectorVariableName (int i)` - Get the ith vector variable name.
- `obj.RemoveAllVariables ()` - Remove all the current variables.
- `obj.RemoveScalarVariables ()` - Remove all the scalar variables.
- `obj.RemoveVectorVariables ()` - Remove all the vector variables.
- `obj.SetReplaceInvalidValues (int )` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `int = obj.GetReplaceInvalidValues ()` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `obj.ReplaceInvalidValuesOn ()` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `obj.ReplaceInvalidValuesOff ()` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `obj.SetReplacementValue (double )` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `double = obj.GetReplacementValue ()` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported

30.37 vtkFunctionSet

30.37.1 Usage

vtkFunctionSet specifies an abstract interface for set of functions of the form F_i = F_i(x_j) where F (with i=1..m) are the functions and x (with j=1..n) are the independent variables. The only supported operation is the function evaluation at x_j.

To create an instance of class vtkFunctionSet, simply invoke its constructor as follows

```
obj = vtkFunctionSet
```

30.37.2 Methods

The class vtkFunctionSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFunctionSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFunctionSet = obj.NewInstance ()`
- `vtkFunctionSet = obj.SafeDownCast (vtkObject o)`
- `int = obj.FunctionValues (double x, double f)` - Evaluate functions at x_j. x and f have to point to valid double arrays of appropriate sizes obtained with GetNumberOfFunctions() and GetNumberOfIndependentVariables.
- `int = obj.GetNumberOfFunctions ()` - Return the number of independent variables. Note that this is constant for a given type of set of functions and can not be changed at run time.
- `int = obj.GetNumberOfIndependentVariables ()`

30.38 vtkGarbageCollector

30.38.1 Usage

vtkGarbageCollector is used by VTK classes that may be involved in reference counting loops (such as Algorithm ¡-¿ Executive). It detects strongly connected components of the reference graph that have been leaked deletes them. The garbage collector uses the ReportReferences method to search the reference graph and construct a net reference count for each connected component. If the net reference count is zero the entire set of objects is deleted. Deleting each component may leak other components, which are then collected recursively.

To enable garbage collection for a class, add these members:

```
public:
 virtual void Register(vtkObjectBase* o)
   {
   this->RegisterInternal(o, 1);
   }
 virtual void UnRegister(vtkObjectBase* o)
   {
   this->UnRegisterInternal(o, 1);
   }
```

```
protected:

 virtual void ReportReferences(vtkGarbageCollector* collector)
   {
   // Report references held by this object that may be in a loop.
   this->Superclass::ReportReferences(collector);
   vtkGarbageCollectorReport(collector, this->OtherObject, ''Other Object'');
   }
```

The implementations should be in the .cxx file in practice. It is important that the reference be reported using the real pointer or smart pointer instance that holds the reference. When collecting the garbage collector will actually set this pointer to NULL. The destructor of the class should be written to deal with this. It is also expected that an invariant is maintained for any reference that is reported. The variable holding the reference must always either be NULL or refer to a fully constructed valid object. Therefore code like "this-¿Object-¿UnRegister(this)" must be avoided if "this-¿Object" is a reported reference because it is possible that the object is deleted before UnRegister returns but then "this-¿Object" will be left as a dangling pointer. Instead use code like

```
vtkObjectBase* obj = this->Object;
this->Object = 0;
obj->UnRegister(this);
```

so that the reported reference maintains the invariant.

If subclassing from a class that already supports garbage collection, one need only provide the ReportReferences method.

To create an instance of class vtkGarbageCollector, simply invoke its constructor as follows

```
obj = vtkGarbageCollector
```

30.38.2 Methods

The class vtkGarbageCollector has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGarbageCollector class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGarbageCollector = obj.NewInstance ()`
- `vtkGarbageCollector = obj.SafeDownCast (vtkObject o)`

30.39 vtkGaussianRandomSequence

30.39.1 Usage

vtkGaussianRandomSequence is a sequence of pseudo random numbers distributed according to the Gaussian/normal distribution (mean=0 and standard deviation=1)

This is just an interface.

To create an instance of class vtkGaussianRandomSequence, simply invoke its constructor as follows

```
obj = vtkGaussianRandomSequence
```

30.39.2 Methods

The class vtkGaussianRandomSequence has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGaussianRandomSequence class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGaussianRandomSequence = obj.NewInstance ()`
- `vtkGaussianRandomSequence = obj.SafeDownCast (vtkObject o)`
- `double = obj.GetScaledValue (double mean, double standardDeviation)` - Convenient method to return a value given the mean and standard deviation of the Gaussian distribution from the the Gaussian distribution of mean=0 and standard deviation=1.0. There is an initial implementation that can be overridden by a subclass.

30.40 vtkGeneralTransform

30.40.1 Usage

vtkGeneralTransform is like vtkTransform and vtkPerspectiveTransform, but it will work with any vtkAbstractTransform as input. It is not as efficient as the other two, however, because arbitrary transformations cannot be concatenated by matrix multiplication. Transform concatenation is simulated by passing each input point through each transform in turn.

To create an instance of class vtkGeneralTransform, simply invoke its constructor as follows

```
obj = vtkGeneralTransform
```

30.40.2 Methods

The class vtkGeneralTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeneralTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeneralTransform = obj.NewInstance ()`
- `vtkGeneralTransform = obj.SafeDownCast (vtkObject o)`
- `obj.Identity ()` - Set this transformation to the identity transformation. If the transform has an Input, then the transformation will be reset so that it is the same as the Input.
- `obj.Inverse ()` - Invert the transformation. This will also set a flag so that the transformation will use the inverse of its Input, if an Input has been set.
- `obj.Translate (double x, double y, double z)` - Create a translation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Translate (double x[3])` - Create a translation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.

- `obj.Translate (float x[3])` - Create a translation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.RotateWXYZ (double angle, double x, double y, double z)` - Create a rotation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is in degrees, and (x,y,z) specifies the axis that the rotation will be performed around.
- `obj.RotateWXYZ (double angle, double axis[3])` - Create a rotation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is in degrees, and (x,y,z) specifies the axis that the rotation will be performed around.
- `obj.RotateWXYZ (double angle, float axis[3])` - Create a rotation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is in degrees, and (x,y,z) specifies the axis that the rotation will be performed around.
- `obj.RotateX (double angle)` - Create a rotation matrix about the X, Y, or Z axis and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is expressed in degrees.
- `obj.RotateY (double angle)` - Create a rotation matrix about the X, Y, or Z axis and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is expressed in degrees.
- `obj.RotateZ (double angle)` - Create a rotation matrix about the X, Y, or Z axis and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is expressed in degrees.
- `obj.Scale (double x, double y, double z)` - Create a scale matrix (i.e. set the diagonal elements to x, y, z) and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Scale (double s[3])` - Create a scale matrix (i.e. set the diagonal elements to x, y, z) and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Scale (float s[3])` - Create a scale matrix (i.e. set the diagonal elements to x, y, z) and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Concatenate (vtkMatrix4x4 matrix)` - Concatenates the matrix with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Concatenate (double elements[16])` - Concatenates the matrix with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Concatenate (vtkAbstractTransform transform)` - Concatenate the specified transform with the current transformation according to PreMultiply or PostMultiply semantics. The concatenation is pipelined, meaning that if any of the transformations are changed, even after Concatenate() is called, those changes will be reflected when you call TransformPoint().
- `obj.PreMultiply ()` - Sets the internal state of the transform to PreMultiply. All subsequent operations will occur before those already represented in the current transformation. In homogeneous matrix notation, M = M*A where M is the current transformation matrix and A is the applied matrix. The default is PreMultiply.
- `obj.PostMultiply ()` - Sets the internal state of the transform to PostMultiply. All subsequent operations will occur after those already represented in the current transformation. In homogeneous matrix notation, M = A*M where M is the current transformation matrix and A is the applied matrix. The default is PreMultiply.
- `int = obj.GetNumberOfConcatenatedTransforms ()` - Get the total number of transformations that are linked into this one via Concatenate() operations or via SetInput().

- `vtkAbstractTransform = obj.GetConcatenatedTransform (int i)`
- `obj.SetInput (vtkAbstractTransform input)` - Set the input for this transformation. This will be used as the base transformation if it is set. This method allows you to build a transform pipeline: if the input is modified, then this transformation will automatically update accordingly. Note that the InverseFlag, controlled via Inverse(), determines whether this transformation will use the Input or the inverse of the Input.
- `vtkAbstractTransform = obj.GetInput ()` - Set the input for this transformation. This will be used as the base transformation if it is set. This method allows you to build a transform pipeline: if the input is modified, then this transformation will automatically update accordingly. Note that the InverseFlag, controlled via Inverse(), determines whether this transformation will use the Input or the inverse of the Input.
- `int = obj.GetInverseFlag ()` - Get the inverse flag of the transformation. This controls whether it is the Input or the inverse of the Input that is used as the base transformation. The InverseFlag is flipped every time Inverse() is called. The InverseFlag is off when a transform is first created.
- `obj.Push ()` - Pushes the current transformation onto the transformation stack.
- `obj.Pop ()` - Deletes the transformation on the top of the stack and sets the top to the next transformation on the stack.
- `obj.InternalTransformPoint (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `int = obj.CircuitCheck (vtkAbstractTransform transform)` - Check for self-reference. Will return true if concatenating with the specified transform, setting it to be our inverse, or setting it to be our input will create a circular reference. CircuitCheck is automatically called by SetInput(), SetInverse(), and Concatenate(vtkXTransform *). Avoid using this function, it is experimental.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.
- `long = obj.GetMTime ()` - Override GetMTime to account for input and concatenation.

30.41 vtkHeap

30.41.1 Usage

This class is a replacement for malloc/free and new/delete for software that has inherent memory leak or performance problems. For example, external software such as the PLY library (vtkPLY) and VRML importer (vtkVRMLImporter) are often written with lots of malloc() calls but without the corresponding free() invocations. The class vtkOrderedTriangulator may create and delete millions of new/delete calls. This class allows the overloading of the C++ new operator (or other memory allocation requests) by using the method AllocateMemory(). Memory is deleted with an invocation of CleanAll() (which deletes ALL memory; any given memory allocation cannot be deleted). Note: a block size can be used to control the size of each memory allocation. Requests for memory are fulfilled from the block until the block runs out, then a new block is created.

To create an instance of class vtkHeap, simply invoke its constructor as follows

```
obj = vtkHeap
```

30.41.2 Methods

The class vtkHeap has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHeap class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHeap = obj.NewInstance ()`
- `vtkHeap = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfBlocks ()` - Get the number of allocations thus far.
- `int = obj.GetNumberOfAllocations ()` - Get the number of allocations thus far.
- `obj.Reset ()` - This methods resets the current allocation location back to the beginning of the heap. This allows reuse of previously allocated memory which may be beneficial to performance in many cases.
- `string = obj.StringDup (string str)` - Convenience method performs string duplication.

30.42 vtkHomogeneousTransform

30.42.1 Usage

vtkHomogeneousTransform provides a generic interface for homogeneous transformations, i.e. transformations which can be represented by multiplying a 4x4 matrix with a homogeneous coordinate.

To create an instance of class vtkHomogeneousTransform, simply invoke its constructor as follows

```
obj = vtkHomogeneousTransform
```

30.42.2 Methods

The class vtkHomogeneousTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHomogeneousTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHomogeneousTransform = obj.NewInstance ()`
- `vtkHomogeneousTransform = obj.SafeDownCast (vtkObject o)`
- `obj.TransformPoints (vtkPoints inPts, vtkPoints outPts)` - Apply the transformation to a series of points, and append the results to outPts.
- `obj.TransformPointsNormalsVectors (vtkPoints inPts, vtkPoints outPts, vtkDataArray inNms, vtkDataAr` - Apply the transformation to a combination of points, normals and vectors.
- `obj.GetMatrix (vtkMatrix4x4 m)` - Get a copy of the internal transformation matrix. The transform is Updated first, to guarantee that the matrix is valid.
- `vtkMatrix4x4 = obj.GetMatrix ()` - Get a pointer to an internal vtkMatrix4x4 that represents the transformation. An Update() is called on the transform to ensure that the matrix is up-to-date when you get it. You should not store the matrix pointer anywhere because it might become stale.

- `vtkHomogeneousTransform = obj.GetHomogeneousInverse ()` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.

30.43 vtkIdentityTransform

30.43.1 Usage

vtkIdentityTransform is a transformation which will simply pass coordinate data unchanged. All other transform types can also do this, however, the vtkIdentityTransform does so with much greater efficiency.

To create an instance of class vtkIdentityTransform, simply invoke its constructor as follows

```
obj = vtkIdentityTransform
```

30.43.2 Methods

The class vtkIdentityTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIdentityTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIdentityTransform = obj.NewInstance ()`
- `vtkIdentityTransform = obj.SafeDownCast (vtkObject o)`
- `obj.TransformPoints (vtkPoints inPts, vtkPoints outPts)` - Apply the transformation to a series of points, and append the results to outPts.
- `obj.TransformNormals (vtkDataArray inNms, vtkDataArray outNms)` - Apply the transformation to a series of normals, and append the results to outNms.
- `obj.TransformVectors (vtkDataArray inVrs, vtkDataArray outVrs)` - Apply the transformation to a series of vectors, and append the results to outVrs.
- `obj.TransformPointsNormalsVectors (vtkPoints inPts, vtkPoints outPts, vtkDataArray inNms, vtkDataAr` - Apply the transformation to a combination of points, normals and vectors.
- `obj.Inverse ()`
- `obj.InternalTransformPoint (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformNormal (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformNormal (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.

- `obj.InternalTransformVector (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformVector (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make a transform of the same type. This will actually return the same transform.

30.44 vtkIdList

30.44.1 Usage

vtkIdList is used to represent and pass data id's between objects. vtkIdList may represent any type of integer id, but usually represents point and cell ids.

To create an instance of class vtkIdList, simply invoke its constructor as follows

```
obj = vtkIdList
```

30.44.2 Methods

The class vtkIdList has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIdList class.

- `obj.Initialize ()`
- `int = obj.Allocate (vtkIdType sz, int strategy)`
- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIdList = obj.NewInstance ()`
- `vtkIdList = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetNumberOfIds ()` - Return the number of id's in the list.
- `vtkIdType = obj.GetId (vtkIdType i)` - Return the id at location i.
- `obj.SetNumberOfIds (vtkIdType number)` - Specify the number of ids for this object to hold. Does an allocation as well as setting the number of ids.
- `obj.SetId (vtkIdType i, vtkIdType vtkid)` - Set the id at location i. Doesn't do range checking so it's a bit faster than InsertId. Make sure you use SetNumberOfIds() to allocate memory prior to using SetId().
- `obj.InsertId (vtkIdType i, vtkIdType vtkid)` - Set the id at location i. Does range checking and allocates memory as necessary.
- `vtkIdType = obj.InsertNextId (vtkIdType vtkid)` - Add the id specified to the end of the list. Range checking is performed.
- `vtkIdType = obj.InsertUniqueId (vtkIdType vtkid)` - If id is not already in list, insert it and return location in list. Otherwise return just location in list.
- `vtkIdType = obj.GetPointer (vtkIdType i)` - Get a pointer to a particular data index.

- `vtkIdType = obj.WritePointer (vtkIdType i, vtkIdType number)` - Get a pointer to a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `obj.Reset ()` - Reset to an empty state.
- `obj.Squeeze ()` - Free any unused memory.
- `obj.DeepCopy (vtkIdList ids)` - Copy an id list by explicitly copying the internal array.
- `obj.DeleteId (vtkIdType vtkid)` - Delete specified id from list. Will remove all occurrences of id in list.
- `vtkIdType = obj.IsId (vtkIdType vtkid)` - Return -1 if id specified is not contained in the list; otherwise return the position in the list.
- `obj.IntersectWith (vtkIdList \&otherIds)` - Intersect this list with another vtkIdList. Updates current list according to result of intersection operation.

30.45 vtkIdListCollection

30.45.1 Usage

vtkIdListCollection is an object that creates and manipulates lists of datasets. See also vtkCollection and subclasses.

To create an instance of class vtkIdListCollection, simply invoke its constructor as follows

```
obj = vtkIdListCollection
```

30.45.2 Methods

The class vtkIdListCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIdListCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIdListCollection = obj.NewInstance ()`
- `vtkIdListCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkIdList ds)` - Get the next dataset in the list.
- `vtkIdList = obj.GetNextItem ()` - Get the next dataset in the list.
- `vtkIdList = obj.GetItem (int i)` - Get the ith dataset in the list.

30.46 vtkIdTypeArray

30.46.1 Usage

vtkIdTypeArray is an array of values of type vtkIdType. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkIdTypeArray, simply invoke its constructor as follows

```
obj = vtkIdTypeArray
```

30.46.2 Methods

The class vtkIdTypeArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIdTypeArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIdTypeArray = obj.NewInstance ()`
- `vtkIdTypeArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `vtkIdType = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, vtkIdType value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, vtkIdType f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (vtkIdType f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `vtkIdType = obj.WritePointer (vtkIdType id, vtkIdType number)` - Get the address of a particular data index. Performs no checks to verify that the memory has been allocated etc.
- `vtkIdType = obj.GetPointer (vtkIdType id)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.

30.47 vtkImplicitFunction

30.47.1 Usage

vtkImplicitFunction specifies an abstract interface for implicit functions. Implicit functions are real valued functions defined in 3D space, w = F(x,y,z). Two primitive operations are required: the ability to evaluate the function, and the function gradient at a given point. The implicit function divides space into three regions: on the surface (F(x,y,z)=w), outside of the surface (F(x,y,z)¿c), and inside the surface (F(x,y,z)¡c). (When c is zero, positive values are outside, negative values are inside, and zero is on the surface. Note also that the function gradient points from inside to outside.)

Implicit functions are very powerful. It is possible to represent almost any type of geometry with the level sets w = const, especially if you use boolean combinations of implicit functions (see vtkImplicitBoolean).

vtkImplicitFunction provides a mechanism to transform the implicit function(s) via a vtkAbstractTransform. This capability can be used to translate, orient, scale, or warp implicit functions. For example, a sphere implicit function can be transformed into an oriented ellipse.

To create an instance of class vtkImplicitFunction, simply invoke its constructor as follows

```
obj = vtkImplicitFunction
```

30.47.2 Methods

The class vtkImplicitFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitFunction = obj.NewInstance ()`
- `vtkImplicitFunction = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Overload standard modified time function. If Transform is modified, then this object is modified as well.
- `double = obj.FunctionValue (double x[3])` - Evaluate function at position x-y-z and return value. Point x[3] is transformed through transform (if provided).
- `double = obj.FunctionValue (double x, double y, double z)` - Evaluate function at position x-y-z and return value. Point x[3] is transformed through transform (if provided).
- `obj.FunctionGradient (double x[3], double g[3])` - Evaluate function gradient at position x-y-z and pass back vector. Point x[3] is transformed through transform (if provided).
- `double = obj.FunctionGradient (double x[3])` - Evaluate function gradient at position x-y-z and pass back vector. Point x[3] is transformed through transform (if provided).
- `double = obj.FunctionGradient (double x, double y, double z)` - Evaluate function gradient at position x-y-z and pass back vector. Point x[3] is transformed through transform (if provided).
- `obj.SetTransform (vtkAbstractTransform )` - Set/Get a transformation to apply to input points before executing the implicit function.
- `obj.SetTransform (double elements[16])` - Set/Get a transformation to apply to input points before executing the implicit function.
- `vtkAbstractTransform = obj.GetTransform ()` - Set/Get a transformation to apply to input points before executing the implicit function.
- `double = obj.EvaluateFunction (double x[3])` - Evaluate function at position x-y-z and return value. You should generally not call this method directly, you should use FunctionValue() instead. This method must be implemented by any derived class.
- `double = obj.EvaluateFunction (double x, double y, double z)` - Evaluate function at position x-y-z and return value. You should generally not call this method directly, you should use FunctionValue() instead. This method must be implemented by any derived class.
- `obj.EvaluateGradient (double x[3], double g[3])` - Evaluate function gradient at position x-y-z and pass back vector. You should generally not call this method directly, you should use FunctionGradient() instead. This method must be implemented by any derived class.

30.48 vtkImplicitFunctionCollection

30.48.1 Usage

vtkImplicitFunctionCollection is an object that creates and manipulates lists of objects of type vtkImplicitFunction.

To create an instance of class vtkImplicitFunctionCollection, simply invoke its constructor as follows

```
obj = vtkImplicitFunctionCollection
```

30.48.2 Methods

The class vtkImplicitFunctionCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitFunctionCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitFunctionCollection = obj.NewInstance ()`
- `vtkImplicitFunctionCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkImplicitFunction )` - Add an implicit function to the list.
- `vtkImplicitFunction = obj.GetNextItem ()` - Get the next implicit function in the list.

30.49 vtkInformation

30.49.1 Usage

vtkInformation represents information and/or data for one input or one output of a vtkAlgorithm. It maps from keys to values of several data types. Instances of this class are collected in vtkInformationVector instances and passed to vtkAlgorithm::ProcessRequest calls. The information and data referenced by the instance on a particular input or output define the request made to the vtkAlgorithm instance.

To create an instance of class vtkInformation, simply invoke its constructor as follows

```
obj = vtkInformation
```

30.49.2 Methods

The class vtkInformation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformation = obj.NewInstance ()`
- `vtkInformation = obj.SafeDownCast (vtkObject o)`
- `obj.Modified ()` - Modified signature with no arguments that calls Modified on vtkObject superclass.
- `obj.Modified (vtkInformationKey key)` - Modified signature that takes an information key as an argument. Sets the new MTime and invokes a modified event with the information key as call data.
- `obj.Clear ()` - Clear all information entries.
- `int = obj.GetNumberOfKeys ()` - Return the number of keys in this information object (as would be returned by iterating over the keys).
- `obj.Copy (vtkInformation from, int deep)` - Copy all information entries from the given vtkInformation instance. Any previously existing entries are removed. If deep==1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).

- `obj.CopyEntry (vtkInformation from, vtkInformationKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationDataObjectKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationDoubleVectorKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationInformationKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationInformationVectorKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationIntegerKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationIntegerVectorKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationRequestKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationStringKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationStringVectorKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntry (vtkInformation from, vtkInformationUnsignedLongKey key, int deep)` - Copy the key/value pair associated with the given key in the given information object. If deep=1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.CopyEntries (vtkInformation from, vtkInformationKeyVectorKey key, int deep)` - Use the given key to lookup a list of other keys in the given information object. The key/value pairs associated with these other keys will be copied. If deep==1, a deep copy of the information structure is performed.

- `int = obj.Has (vtkInformationKey key)` - Check whether the given key appears in this information object.
- `obj.Remove (vtkInformationKey key)` - Remove the given key and its data from this information object.
- `obj.Set (vtkInformationRequestKey key)` - Get/Set a request-valued entry.
- `obj.Remove (vtkInformationRequestKey key)` - Get/Set a request-valued entry.
- `int = obj.Has (vtkInformationRequestKey key)` - Get/Set a request-valued entry.
- `obj.Set (vtkInformationIntegerKey key, int value)` - Get/Set an integer-valued entry.
- `int = obj.Get (vtkInformationIntegerKey key)` - Get/Set an integer-valued entry.
- `obj.Remove (vtkInformationIntegerKey key)` - Get/Set an integer-valued entry.
- `int = obj.Has (vtkInformationIntegerKey key)` - Get/Set an integer-valued entry.
- `obj.Set (vtkInformationIdTypeKey key, vtkIdType value)` - Get/Set a vtkIdType-valued entry.
- `vtkIdType = obj.Get (vtkInformationIdTypeKey key)` - Get/Set a vtkIdType-valued entry.
- `obj.Remove (vtkInformationIdTypeKey key)` - Get/Set a vtkIdType-valued entry.
- `int = obj.Has (vtkInformationIdTypeKey key)` - Get/Set a vtkIdType-valued entry.
- `obj.Set (vtkInformationDoubleKey key, double value)` - Get/Set an double-valued entry.
- `double = obj.Get (vtkInformationDoubleKey key)` - Get/Set an double-valued entry.
- `obj.Remove (vtkInformationDoubleKey key)` - Get/Set an double-valued entry.
- `int = obj.Has (vtkInformationDoubleKey key)` - Get/Set an double-valued entry.
- `obj.Append (vtkInformationIntegerVectorKey key, int value)` - Get/Set an integer-vector-valued entry.
- `obj.Set (vtkInformationIntegerVectorKey key, int value, int length)` - Get/Set an integer-vector-valued entry.
- `obj.Set (vtkInformationIntegerVectorKey key, int value1, int value2, int value3)` - Get/Set an integer-vector-valued entry.
- `obj.Set (vtkInformationIntegerVectorKey key, int value1, int value2, int value3, int value4, int va` - Get/Set an integer-vector-valued entry.
- `int = obj.Get (vtkInformationIntegerVectorKey key, int idx)` - Get/Set an integer-vector-valued entry.
- `obj.Get (vtkInformationIntegerVectorKey key, int value)` - Get/Set an integer-vector-valued entry.
- `int = obj.Length (vtkInformationIntegerVectorKey key)` - Get/Set an integer-vector-valued entry.
- `obj.Remove (vtkInformationIntegerVectorKey key)` - Get/Set an integer-vector-valued entry.
- `int = obj.Has (vtkInformationIntegerVectorKey key)` - Get/Set an integer-vector-valued entry.
- `obj.Append (vtkInformationStringVectorKey key, string value)` - Get/Set a string-vector-valued entry.

- `obj.Set (vtkInformationStringVectorKey key, string value, int idx)` - Get/Set a string-vector-valued entry.
- `string = obj.Get (vtkInformationStringVectorKey key, int idx)` - Get/Set a string-vector-valued entry.
- `int = obj.Length (vtkInformationStringVectorKey key)` - Get/Set a string-vector-valued entry.
- `obj.Remove (vtkInformationStringVectorKey key)` - Get/Set a string-vector-valued entry.
- `int = obj.Has (vtkInformationStringVectorKey key)` - Get/Set a string-vector-valued entry.
- `obj.Set (vtkInformationIntegerPointerKey key, int value, int length)` - Get/Set an integer-pointer-valued entry.
- `obj.Get (vtkInformationIntegerPointerKey key, int value)` - Get/Set an integer-pointer-valued entry.
- `int = obj.Length (vtkInformationIntegerPointerKey key)` - Get/Set an integer-pointer-valued entry.
- `obj.Remove (vtkInformationIntegerPointerKey key)` - Get/Set an integer-pointer-valued entry.
- `int = obj.Has (vtkInformationIntegerPointerKey key)` - Get/Set an integer-pointer-valued entry.
- `obj.Set (vtkInformationUnsignedLongKey key, long value)` - Get/Set an unsigned-long-valued entry.
- `long = obj.Get (vtkInformationUnsignedLongKey key)` - Get/Set an unsigned-long-valued entry.
- `obj.Remove (vtkInformationUnsignedLongKey key)` - Get/Set an unsigned-long-valued entry.
- `int = obj.Has (vtkInformationUnsignedLongKey key)` - Get/Set an unsigned-long-valued entry.
- `obj.Append (vtkInformationDoubleVectorKey key, double value)` - Get/Set an double-vector-valued entry.
- `obj.Set (vtkInformationDoubleVectorKey key, double value, int length)` - Get/Set an double-vector-valued entry.
- `obj.Set (vtkInformationDoubleVectorKey key, double value1, double value2, double value3)` - Get/Set an double-vector-valued entry.
- `obj.Set (vtkInformationDoubleVectorKey key, double value1, double value2, double value3, double val` - Get/Set an double-vector-valued entry.
- `double = obj.Get (vtkInformationDoubleVectorKey key, int idx)` - Get/Set an double-vector-valued entry.
- `obj.Get (vtkInformationDoubleVectorKey key, double value)` - Get/Set an double-vector-valued entry.
- `int = obj.Length (vtkInformationDoubleVectorKey key)` - Get/Set an double-vector-valued entry.
- `obj.Remove (vtkInformationDoubleVectorKey key)` - Get/Set an double-vector-valued entry.
- `int = obj.Has (vtkInformationDoubleVectorKey key)` - Get/Set an double-vector-valued entry.
- `obj.Append (vtkInformationKeyVectorKey key, vtkInformationKey value)` - Get/Set an InformationKey-vector-valued entry.

- `obj.AppendUnique (vtkInformationKeyVectorKey key, vtkInformationKey value)` - Get/Set an InformationKey-vector-valued entry.
- `obj.Remove (vtkInformationKeyVectorKey key, vtkInformationKey value)` - Get/Set an InformationKey-vector-valued entry.
- `vtkInformationKey = obj.Get (vtkInformationKeyVectorKey key, int idx)` - Get/Set an InformationKey-vector-valued entry.
- `int = obj.Length (vtkInformationKeyVectorKey key)` - Get/Set an InformationKey-vector-valued entry.
- `obj.Remove (vtkInformationKeyVectorKey key)` - Get/Set an InformationKey-vector-valued entry.
- `int = obj.Has (vtkInformationKeyVectorKey key)` - Get/Set an InformationKey-vector-valued entry.
- `obj.Set (vtkInformationStringKey key, string )` - Get/Set a string-valued entry.
- `string = obj.Get (vtkInformationStringKey key)` - Get/Set a string-valued entry.
- `obj.Remove (vtkInformationStringKey key)` - Get/Set a string-valued entry.
- `int = obj.Has (vtkInformationStringKey key)` - Get/Set a string-valued entry.
- `obj.Set (vtkInformationInformationKey key, vtkInformation )` - Get/Set an entry storing another vtkInformation instance.
- `vtkInformation = obj.Get (vtkInformationInformationKey key)` - Get/Set an entry storing another vtkInformation instance.
- `obj.Remove (vtkInformationInformationKey key)` - Get/Set an entry storing another vtkInformation instance.
- `int = obj.Has (vtkInformationInformationKey key)` - Get/Set an entry storing another vtkInformation instance.
- `obj.Set (vtkInformationInformationVectorKey key, vtkInformationVector )` - Get/Set an entry storing a vtkInformationVector instance.
- `vtkInformationVector = obj.Get (vtkInformationInformationVectorKey key)` - Get/Set an entry storing a vtkInformationVector instance.
- `obj.Remove (vtkInformationInformationVectorKey key)` - Get/Set an entry storing a vtkInformationVector instance.
- `int = obj.Has (vtkInformationInformationVectorKey key)` - Get/Set an entry storing a vtkInformationVector instance.
- `obj.Set (vtkInformationObjectBaseKey key, vtkObjectBase )` - Get/Set an entry storing a vtkObjectBase instance.
- `vtkObjectBase = obj.Get (vtkInformationObjectBaseKey key)` - Get/Set an entry storing a vtkObjectBase instance.
- `obj.Remove (vtkInformationObjectBaseKey key)` - Get/Set an entry storing a vtkObjectBase instance.
- `int = obj.Has (vtkInformationObjectBaseKey key)` - Get/Set an entry storing a vtkObjectBase instance.

- `obj.Set (vtkInformationDataObjectKey key, vtkDataObject )` - Get/Set an entry storing a vtkDataObject instance.
- `vtkDataObject = obj.Get (vtkInformationDataObjectKey key)` - Get/Set an entry storing a vtkDataObject instance.
- `obj.Remove (vtkInformationDataObjectKey key)` - Get/Set an entry storing a vtkDataObject instance.
- `int = obj.Has (vtkInformationDataObjectKey key)` - Get/Set an entry storing a vtkDataObject instance.
- `obj.Register (vtkObjectBase o)` - Initiate garbage collection when a reference is removed.
- `obj.UnRegister (vtkObjectBase o)` - Initiate garbage collection when a reference is removed.
- `obj.SetRequest (vtkInformationRequestKey request)` - Get/Set the Request ivar
- `vtkInformationRequestKey = obj.GetRequest ()` - Get/Set the Request ivar

30.50 vtkInformationDataObjectKey

30.50.1 Usage

vtkInformationDataObjectKey is used to represent keys in vtkInformation for values that are vtkDataObject instances.

To create an instance of class vtkInformationDataObjectKey, simply invoke its constructor as follows

```
obj = vtkInformationDataObjectKey
```

30.50.2 Methods

The class vtkInformationDataObjectKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationDataObjectKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationDataObjectKey = obj.NewInstance ()`
- `vtkInformationDataObjectKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationDataObjectKey = obj.(string name, string location)`
- `~vtkInformationDataObjectKey = obj.()`
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.
- `obj.Report (vtkInformation info, vtkGarbageCollector collector)` - Report a reference this key has in the given information object.

30.51 vtkInformationDoubleKey

30.51.1 Usage

vtkInformationDoubleKey is used to represent keys for double values in vtkInformation.

To create an instance of class vtkInformationDoubleKey, simply invoke its constructor as follows

```
obj = vtkInformationDoubleKey
```

30.51.2 Methods

The class vtkInformationDoubleKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationDoubleKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationDoubleKey = obj.NewInstance ()`
- `vtkInformationDoubleKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationDoubleKey = obj.(string name, string location)`
- `~vtkInformationDoubleKey = obj.()`
- `obj.Set (vtkInformation info, double )` - Get/Set the value associated with this key in the given information object.
- `double = obj.Get (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.52 vtkInformationDoubleVectorKey

30.52.1 Usage

vtkInformationDoubleVectorKey is used to represent keys for double vector values in vtkInformation.h

To create an instance of class vtkInformationDoubleVectorKey, simply invoke its constructor as follows

```
obj = vtkInformationDoubleVectorKey
```

30.52.2 Methods

The class vtkInformationDoubleVectorKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationDoubleVectorKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationDoubleVectorKey = obj.NewInstance ()`

- `vtkInformationDoubleVectorKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationDoubleVectorKey = obj.(string name, string location, int length)`
- `~vtkInformationDoubleVectorKey = obj.()`
- `obj.Append (vtkInformation info, double value)` - Get/Set the value associated with this key in the given information object.
- `obj.Set (vtkInformation info, double value, int length)` - Get/Set the value associated with this key in the given information object.
- `double = obj.Get (vtkInformation info, int idx)` - Get/Set the value associated with this key in the given information object.
- `obj.Get (vtkInformation info, double value)` - Get/Set the value associated with this key in the given information object.
- `int = obj.Length (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.53 vtkInformationIdTypeKey

30.53.1 Usage

vtkInformationIdTypeKey is used to represent keys for vtkIdType values in vtkInformation.

To create an instance of class vtkInformationIdTypeKey, simply invoke its constructor as follows

```
obj = vtkInformationIdTypeKey
```

30.53.2 Methods

The class vtkInformationIdTypeKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationIdTypeKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationIdTypeKey = obj.NewInstance ()`
- `vtkInformationIdTypeKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationIdTypeKey = obj.(string name, string location)`
- `~vtkInformationIdTypeKey = obj.()`
- `obj.Set (vtkInformation info, vtkIdType )` - Get/Set the value associated with this key in the given information object.
- `vtkIdType = obj.Get (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.54 vtkInformationInformationKey

30.54.1 Usage

vtkInformationInformationKey is used to represent keys in vtkInformation for other information objects.

To create an instance of class vtkInformationInformationKey, simply invoke its constructor as follows

```
obj = vtkInformationInformationKey
```

30.54.2 Methods

The class vtkInformationInformationKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationInformationKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationInformationKey = obj.NewInstance ()`
- `vtkInformationInformationKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationInformationKey = obj.(string name, string location)`
- `~vtkInformationInformationKey = obj.()`
- `obj.Set (vtkInformation info, vtkInformation )` - Get/Set the value associated with this key in the given information object.
- `vtkInformation = obj.Get (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.
- `obj.DeepCopy (vtkInformation from, vtkInformation to)` - Duplicate (new instance created) the entry associated with this key from one information object to another (new instances of any contained vtkInformation and vtkInformationVector objects are created).

30.55 vtkInformationInformationVectorKey

30.55.1 Usage

vtkInformationInformationVectorKey is used to represent keys in vtkInformation for vectors of other vtkInformation objects.

To create an instance of class vtkInformationInformationVectorKey, simply invoke its constructor as follows

```
obj = vtkInformationInformationVectorKey
```

30.55.2 Methods

The class vtkInformationInformationVectorKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationInformationVectorKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationInformationVectorKey = obj.NewInstance ()`
- `vtkInformationInformationVectorKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationInformationVectorKey = obj.(string name, string location)`
- `~vtkInformationInformationVectorKey = obj.()`
- `obj.Set (vtkInformation info, vtkInformationVector )` - Get/Set the value associated with this key in the given information object.
- `vtkInformationVector = obj.Get (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.
- `obj.DeepCopy (vtkInformation from, vtkInformation to)` - Duplicate (new instance created) the entry associated with this key from one information object to another (new instances of any contained vtkInformation and vtkInformationVector objects are created).
- `obj.Report (vtkInformation info, vtkGarbageCollector collector)` - Report a reference this key has in the given information object.

30.56 vtkInformationIntegerKey

30.56.1 Usage

vtkInformationIntegerKey is used to represent keys for integer values in vtkInformation.

To create an instance of class vtkInformationIntegerKey, simply invoke its constructor as follows

```
obj = vtkInformationIntegerKey
```

30.56.2 Methods

The class vtkInformationIntegerKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationIntegerKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationIntegerKey = obj.NewInstance ()`
- `vtkInformationIntegerKey = obj.SafeDownCast (vtkObject o)`

- `vtkInformationIntegerKey = obj.(string name, string location)`
- `~vtkInformationIntegerKey = obj.()`
- `obj.Set (vtkInformation info, int )` - Get/Set the value associated with this key in the given information object.
- `int = obj.Get (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.57 vtkInformationIntegerPointerKey

30.57.1 Usage

vtkInformationIntegerPointerKey is used to represent keys for pointer to integer values in vtkInformation.h
To create an instance of class vtkInformationIntegerPointerKey, simply invoke its constructor as follows

```
obj = vtkInformationIntegerPointerKey
```

30.57.2 Methods

The class vtkInformationIntegerPointerKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationIntegerPointerKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationIntegerPointerKey = obj.NewInstance ()`
- `vtkInformationIntegerPointerKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationIntegerPointerKey = obj.(string name, string location, int length)`
- `~vtkInformationIntegerPointerKey = obj.()`
- `obj.Set (vtkInformation info, int value, int length)` - Get/Set the value associated with this key in the given information object.
- `obj.Get (vtkInformation info, int value)` - Get/Set the value associated with this key in the given information object.
- `int = obj.Length (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.58 vtkInformationIntegerVectorKey

30.58.1 Usage

vtkInformationIntegerVectorKey is used to represent keys for integer vector values in vtkInformation.h

To create an instance of class vtkInformationIntegerVectorKey, simply invoke its constructor as follows

```
obj = vtkInformationIntegerVectorKey
```

30.58.2 Methods

The class vtkInformationIntegerVectorKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationIntegerVectorKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationIntegerVectorKey = obj.NewInstance ()`
- `vtkInformationIntegerVectorKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationIntegerVectorKey = obj.(string name, string location, int length)`
- `~vtkInformationIntegerVectorKey = obj.()`
- `obj.Append (vtkInformation info, int value)` - Get/Set the value associated with this key in the given information object.
- `obj.Set (vtkInformation info, int value, int length)` - Get/Set the value associated with this key in the given information object.
- `int = obj.Get (vtkInformation info, int idx)` - Get/Set the value associated with this key in the given information object.
- `obj.Get (vtkInformation info, int value)` - Get/Set the value associated with this key in the given information object.
- `int = obj.Length (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.59 vtkInformationIterator

30.59.1 Usage

vtkInformationIterator can be used to iterate over the keys of an information object. The corresponding values can then be directly obtained from the information object using the keys.

To create an instance of class vtkInformationIterator, simply invoke its constructor as follows

```
obj = vtkInformationIterator
```

30.59.2 Methods

The class vtkInformationIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationIterator = obj.NewInstance ()`
- `vtkInformationIterator = obj.SafeDownCast (vtkObject o)`
- `obj.SetInformation (vtkInformation )` - Set/Get the information to iterator over.
- `vtkInformation = obj.GetInformation ()` - Set/Get the information to iterator over.
- `obj.InitTraversal ()` - Move the iterator to the beginning of the collection.
- `obj.GoToFirstItem ()` - Move the iterator to the beginning of the collection.
- `obj.GoToNextItem ()` - Move the iterator to the next item in the collection.
- `int = obj.IsDoneWithTraversal ()` - Test whether the iterator is currently pointing to a valid item. Returns 1 for yes, 0 for no.
- `vtkInformationKey = obj.GetCurrentKey ()` - Get the current item. Valid only when IsDoneWith-Traversal() returns 1.

30.60 vtkInformationKey

30.60.1 Usage

vtkInformationKey is the superclass for all keys used to access the map represented by vtkInformation. The vtkInformation::Set and vtkInformation::Get methods of vtkInformation are accessed by information keys. A key is a pointer to an instance of a subclass of vtkInformationKey. The type of the subclass determines the overload of Set/Get that is selected. This ensures that the type of value stored in a vtkInformation instance corresponding to a given key matches the type expected for that key.

To create an instance of class vtkInformationKey, simply invoke its constructor as follows

```
obj = vtkInformationKey
```

30.60.2 Methods

The class vtkInformationKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationKey = obj.NewInstance ()`
- `vtkInformationKey = obj.SafeDownCast (vtkObject o)`
- `obj.Register (vtkObjectBase )` - Prevent normal vtkObject reference counting behavior.

- `obj.UnRegister (vtkObjectBase )` - Prevent normal vtkObject reference counting behavior.
- `string = obj.GetName ()` - Get the name of the key. This is not the type of the key, but the name of the key instance.
- `string = obj.GetLocation ()` - Get the location of the key. This is the name of the class in which the key is defined.
- `vtkInformationKey = obj.(string name, string location)` - Key instances are static data that need to be created and destroyed. The constructor and destructor must be public. The name of the static instance and the class in which it is defined should be passed to the constructor. They must be string literals because the strings are not copied.
- `~vtkInformationKey = obj.()` - Key instances are static data that need to be created and destroyed. The constructor and destructor must be public. The name of the static instance and the class in which it is defined should be passed to the constructor. They must be string literals because the strings are not copied.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.
- `obj.DeepCopy (vtkInformation from, vtkInformation to)` - Check whether this key appears in the given information object.
- `int = obj.Has (vtkInformation info)` - Check whether this key appears in the given information object.
- `obj.Remove (vtkInformation info)` - Remove this key from the given information object.
- `obj.Report (vtkInformation info, vtkGarbageCollector collector)` - Report a reference this key has in the given information object.

30.61 vtkInformationKeyVectorKey

30.61.1 Usage

vtkInformationKeyVectorKey is used to represent keys for vector-of-keys values in vtkInformation.

To create an instance of class vtkInformationKeyVectorKey, simply invoke its constructor as follows

```
obj = vtkInformationKeyVectorKey
```

30.61.2 Methods

The class vtkInformationKeyVectorKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationKeyVectorKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationKeyVectorKey = obj.NewInstance ()`
- `vtkInformationKeyVectorKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationKeyVectorKey = obj.(string name, string location)`
- `~vtkInformationKeyVectorKey = obj.()`

- `obj.Append (vtkInformation info, vtkInformationKey value)` - Get/Set the value associated with this key in the given information object.
- `obj.AppendUnique (vtkInformation info, vtkInformationKey value)` - Get/Set the value associated with this key in the given information object.
- `obj.RemoveItem (vtkInformation info, vtkInformationKey value)` - Get/Set the value associated with this key in the given information object.
- `vtkInformationKey = obj.Get (vtkInformation info, int idx)` - Get/Set the value associated with this key in the given information object.
- `int = obj.Length (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.62 vtkInformationObjectBaseKey

30.62.1 Usage

vtkInformationObjectBaseKey is used to represent keys in vtkInformation for values that are vtkObjectBase instances.

To create an instance of class vtkInformationObjectBaseKey, simply invoke its constructor as follows

```
obj = vtkInformationObjectBaseKey
```

30.62.2 Methods

The class vtkInformationObjectBaseKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationObjectBaseKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationObjectBaseKey = obj.NewInstance ()`
- `vtkInformationObjectBaseKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationObjectBaseKey = obj.(string name, string location, string requiredClass)`
- `~vtkInformationObjectBaseKey = obj.()`
- `obj.Set (vtkInformation info, vtkObjectBase )` - Get/Set the value associated with this key in the given information object.
- `vtkObjectBase = obj.Get (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.
- `obj.Report (vtkInformation info, vtkGarbageCollector collector)` - Report a reference this key has in the given information object.

30.63 vtkInformationObjectBaseVectorKey

30.63.1 Usage

vtkInformationObjectBaseVectorKey is used to represent keys for double vector values in vtkInformation.h. NOTE the interface in this key differs from that in other similar keys because of our internal use of smart pointers.

To create an instance of class vtkInformationObjectBaseVectorKey, simply invoke its constructor as follows

```
obj = vtkInformationObjectBaseVectorKey
```

30.63.2 Methods

The class vtkInformationObjectBaseVectorKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationObjectBaseVectorKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationObjectBaseVectorKey = obj.NewInstance ()`
- `vtkInformationObjectBaseVectorKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationObjectBaseVectorKey = obj.(string name, string location, string requiredClass)` - The name of the static instance and the class in which it is defined(location) should be passed to the constructor. Providing "requiredClass" name one can insure that only objects of type "requiredClass" are stored in vectors associated with the instance of this key type created. These should be string literals as they are not coppied.
- `~vtkInformationObjectBaseVectorKey = obj.()` - The name of the static instance and the class in which it is defined(location) should be passed to the constructor. Providing "requiredClass" name one can insure that only objects of type "requiredClass" are stored in vectors associated with the instance of this key type created. These should be string literals as they are not coppied.
- `obj.Clear (vtkInformation info)` - Clear the vector.
- `obj.Resize (vtkInformation info, int n)` - Resize (extend) the vector to hold n objects. Any new elements created will be null initialized.
- `int = obj.Size (vtkInformation info)` - Get the vector's length.
- `int = obj.Length (vtkInformation info)` - Put the value on the back of the vector, with ref counting.
- `obj.Append (vtkInformation info, vtkObjectBase value)` - Put the value on the back of the vector, with ref counting.
- `obj.Set (vtkInformation info, vtkObjectBase value, int i)` - Set element i of the vector to value. Resizes the vector if needed.
- `vtkObjectBase = obj.Get (vtkInformation info, int idx)` - Get the vtkObjectBase at a specific location in the vector.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.64 vtkInformationQuadratureSchemeDefinitionVectorKey

30.64.1 Usage

vtkInformationQuadratureSchemeDefinitionVectorKey is used to represent keys for double vector values in vtkInformation.h. NOTE the interface in this key differs from that in other similar keys because of our internal use of smart pointers.

To create an instance of class vtkInformationQuadratureSchemeDefinitionVectorKey, simply invoke its constructor as follows

```
obj = vtkInformationQuadratureSchemeDefinitionVectorKey
```

30.64.2 Methods

The class vtkInformationQuadratureSchemeDefinitionVectorKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationQuadratureSchemeDefinitionVectorKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationQuadratureSchemeDefinitionVectorKey = obj.NewInstance ()`
- `vtkInformationQuadratureSchemeDefinitionVectorKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationQuadratureSchemeDefinitionVectorKey = obj.(string name, string location)` - The name of the static instance and the class in which it is defined(location) should be passed to the constructor.
- `~vtkInformationQuadratureSchemeDefinitionVectorKey = obj.()` - The name of the static instance and the class in which it is defined(location) should be passed to the constructor.
- `obj.Clear (vtkInformation info)` - Clear the vector.
- `obj.Resize (vtkInformation info, int n)` - Resize (extend) the vector to hold n objects. Any new elements created will be null initialized.
- `int = obj.Size (vtkInformation info)` - Get the vector's length.
- `int = obj.Length (vtkInformation info)` - Put the value on the back of the vector, with reference counting.
- `obj.Append (vtkInformation info, vtkQuadratureSchemeDefinition value)` - Put the value on the back of the vector, with reference counting.
- `obj.Set (vtkInformation info, vtkQuadratureSchemeDefinition value, int i)` - Set element i of the vector to value. Resizes the vector if needed.
- `vtkQuadratureSchemeDefinition = obj.Get (vtkInformation info, int idx)` - Get the vtkQuadratureSchemeDefinition at a specific location in the vector.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.
- `obj.DeepCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

- `int = obj.SaveState (vtkInformation info, vtkXMLDataElement element)` - Generate an XML representation of the object. Each key/value pair will be nested in the resulting XML hierarchy. The element passed in is assumed to be empty.
- `int = obj.RestoreState (vtkInformation info, vtkXMLDataElement element)` - Load key/value pairs from an XML state representation created with SaveState. Duplicate keys will generate a fatal error.

30.65 vtkInformationRequestKey

30.65.1 Usage

vtkInformationRequestKey is used to represent keys for pointer to pointer values in vtkInformation.h

To create an instance of class vtkInformationRequestKey, simply invoke its constructor as follows

```
obj = vtkInformationRequestKey
```

30.65.2 Methods

The class vtkInformationRequestKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationRequestKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationRequestKey = obj.NewInstance ()`
- `vtkInformationRequestKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationRequestKey = obj.(string name, string location)`
- `~vtkInformationRequestKey = obj.()`
- `obj.Set (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.Remove (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `int = obj.Has (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.66 vtkInformationStringKey

30.66.1 Usage

vtkInformationStringKey is used to represent keys for string values in vtkInformation.

To create an instance of class vtkInformationStringKey, simply invoke its constructor as follows

```
obj = vtkInformationStringKey
```

30.66.2 Methods

The class vtkInformationStringKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationStringKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationStringKey = obj.NewInstance ()`
- `vtkInformationStringKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationStringKey = obj.(string name, string location)`
- `~vtkInformationStringKey = obj.()`
- `obj.Set (vtkInformation info, string )` - Get/Set the value associated with this key in the given information object.
- `string = obj.Get (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.67 vtkInformationStringVectorKey

30.67.1 Usage

vtkInformationStringVectorKey is used to represent keys for String vector values in vtkInformation.h

To create an instance of class vtkInformationStringVectorKey, simply invoke its constructor as follows

```
obj = vtkInformationStringVectorKey
```

30.67.2 Methods

The class vtkInformationStringVectorKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationStringVectorKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationStringVectorKey = obj.NewInstance ()`
- `vtkInformationStringVectorKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationStringVectorKey = obj.(string name, string location, int length)`
- `~vtkInformationStringVectorKey = obj.()`
- `obj.Append (vtkInformation info, string value)` - Get/Set the value associated with this key in the given information object.

- `obj.Set (vtkInformation info, string value, int idx)` - Get/Set the value associated with this key in the given information object.
- `string = obj.Get (vtkInformation info, int idx)` - Get/Set the value associated with this key in the given information object.
- `int = obj.Length (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.68 vtkInformationUnsignedLongKey

30.68.1 Usage

vtkInformationUnsignedLongKey is used to represent keys for unsigned long values in vtkInformation.

To create an instance of class vtkInformationUnsignedLongKey, simply invoke its constructor as follows

```
obj = vtkInformationUnsignedLongKey
```

30.68.2 Methods

The class vtkInformationUnsignedLongKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationUnsignedLongKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationUnsignedLongKey = obj.NewInstance ()`
- `vtkInformationUnsignedLongKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationUnsignedLongKey = obj.(string name, string location)`
- `~vtkInformationUnsignedLongKey = obj.()`
- `obj.Set (vtkInformation info, long )` - Get/Set the value associated with this key in the given information object.
- `long = obj.Get (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.

30.69 vtkInformationVector

30.69.1 Usage

To create an instance of class vtkInformationVector, simply invoke its constructor as follows

```
obj = vtkInformationVector
```

30.69.2 Methods

The class vtkInformationVector has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationVector class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationVector = obj.NewInstance ()`
- `vtkInformationVector = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfInformationObjects ()` - Get/Set the number of information objects in the vector. Setting the number to larger than the current number will create empty vtkInformation instances. Setting the number to smaller than the current number will remove entries from higher indices.
- `obj.SetNumberOfInformationObjects (int n)` - Get/Set the number of information objects in the vector. Setting the number to larger than the current number will create empty vtkInformation instances. Setting the number to smaller than the current number will remove entries from higher indices.
- `obj.SetInformationObject (int index, vtkInformation info)` - Get/Set the vtkInformation instance stored at the given index in the vector. The vector will automatically expand to include the index given if necessary. Missing entries in-between will be filled with empty vtkInformation instances.
- `vtkInformation = obj.GetInformationObject (int index)` - Get/Set the vtkInformation instance stored at the given index in the vector. The vector will automatically expand to include the index given if necessary. Missing entries in-between will be filled with empty vtkInformation instances.
- `obj.Append (vtkInformation info)` - Append/Remove an information object.
- `obj.Remove (vtkInformation info)` - Append/Remove an information object.
- `obj.Register (vtkObjectBase o)` - Initiate garbage collection when a reference is removed.
- `obj.UnRegister (vtkObjectBase o)` - Initiate garbage collection when a reference is removed.
- `obj.Copy (vtkInformationVector from, int deep)` - Copy all information entries from the given vtkInformation instance. Any previously existing entries are removed. If deep==1, a deep copy of the information structure is performed (new instances of any contained vtkInformation and vtkInformationVector objects are created).

30.70 vtkInitialValueProblemSolver

30.70.1 Usage

Given a vtkFunctionSet which returns dF_i(x_j, t)/dt given x_j and t, vtkInitialValueProblemSolver computes the value of F_i at t+deltat.

To create an instance of class vtkInitialValueProblemSolver, simply invoke its constructor as follows

```
obj = vtkInitialValueProblemSolver
```

30.70.2 Methods

The class vtkInitialValueProblemSolver has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInitialValueProblemSolver class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInitialValueProblemSolver = obj.NewInstance ()`
- `vtkInitialValueProblemSolver = obj.SafeDownCast (vtkObject o)`
- `obj.SetFunctionSet (vtkFunctionSet functionset)` - Set / get the dataset used for the implicit function evaluation.
- `vtkFunctionSet = obj.GetFunctionSet ()` - Set / get the dataset used for the implicit function evaluation.
- `int = obj.IsAdaptive ()`

30.71 vtkInstantiator

30.71.1 Usage

vtkInstantiator provides an interface to create an instance of any VTK class from its name. Instances are created through registered pointers to functions returning the objects. New classes can also be registered with the creator. VTK libraries automatically register their classes with the creator when they are loaded. Instances are created using the static New() method, so the normal vtkObjectFactory mechanism is still invoked.

When using this class from language wrappers (Tcl, Python, or Java), the vtkInstantiator should be able to create any class from any kit that has been loaded.

In C++ code, one should include the header for each kit from which one wishes to create instances through vtkInstantiator. This is necessary to ensure proper linking when building static libraries. Be careful, though, because including each kit's header means every class from that kit will be linked into your executable whether or not the class is used. The headers are:

vtkCommon - vtkCommonInstantiator.h vtkFiltering - vtkFilteringInstantiator.h vtkIO - vtkIOInstantiator.h vtkImaging - vtkImagingInstantiator.h vtkGraphics - vtkGraphicsInstantiator.h vtkRendering - vtkRenderingInstantiator.h vtkVolumeRendering - vtkVolumeRenderingInstantiator.h vtkHybrid - vtkHybridInstantiator.h vtkParallel - vtkParallelInstantiator.h

The VTK_MAKE_INSTANTIATOR() command in CMake is used to automatically generate the creator registration for each VTK library. It can also be used to create registration code for VTK-style user libraries that are linked to vtkCommon. After using this command to register classes from a new library, the generated header must be included.

To create an instance of class vtkInstantiator, simply invoke its constructor as follows

```
obj = vtkInstantiator
```

30.71.2 Methods

The class vtkInstantiator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInstantiator class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkInstantiator = obj.NewInstance ()`
- `vtkInstantiator = obj.SafeDownCast (vtkObject o)`

30.72 vtkIntArray

30.72.1 Usage

vtkIntArray is an array of values of type int. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkIntArray, simply invoke its constructor as follows

```
obj = vtkIntArray
```

30.72.2 Methods

The class vtkIntArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIntArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIntArray = obj.NewInstance ()`
- `vtkIntArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, int tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, int tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, int tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (int tuple)` - Get the data at a particular index.
- `int = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, int value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, int f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (int f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.

- `obj.SetArray (int array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (int array, vtkIdType size, int save, int deleteMethod)`

30.73 vtkLinearTransform

30.73.1 Usage

vtkLinearTransform provides a generic interface for linear (affine or 12 degree-of-freedom) geometric transformations.

To create an instance of class vtkLinearTransform, simply invoke its constructor as follows

```
obj = vtkLinearTransform
```

30.73.2 Methods

The class vtkLinearTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLinearTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLinearTransform = obj.NewInstance ()`
- `vtkLinearTransform = obj.SafeDownCast (vtkObject o)`
- `obj.TransformNormal (float in[3], float out[3])` - Apply the transformation to a normal. You can use the same array to store both the input and output.
- `obj.TransformNormal (double in[3], double out[3])` - Apply the transformation to a double-precision normal. You can use the same array to store both the input and output.
- `double = obj.TransformNormal (double x, double y, double z)` - Synonymous with TransformDoubleNormal(x,y,z). Use this if you are programming in python, tcl or Java.
- `double = obj.TransformNormal (double normal[3])` - Synonymous with TransformDoubleNormal(x,y,z). Use this if you are programming in python, tcl or Java.
- `float = obj.TransformFloatNormal (float x, float y, float z)` - Apply the transformation to an (x,y,z) normal. Use this if you are programming in python, tcl or Java.
- `float = obj.TransformFloatNormal (float normal[3])` - Apply the transformation to an (x,y,z) normal. Use this if you are programming in python, tcl or Java.
- `double = obj.TransformDoubleNormal (double x, double y, double z)` - Apply the transformation to a double-precision (x,y,z) normal. Use this if you are programming in python, tcl or Java.
- `double = obj.TransformDoubleNormal (double normal[3])` - Apply the transformation to a double-precision (x,y,z) normal. Use this if you are programming in python, tcl or Java.
- `double = obj.TransformVector (double x, double y, double z)` - Synonymous with TransformDoubleVector(x,y,z). Use this if you are programming in python, tcl or Java.

- `double = obj.TransformVector (double normal[3])` - Synonymous with TransformDoubleVector(x,y,z). Use this if you are programming in python, tcl or Java.
- `obj.TransformVector (float in[3], float out[3])` - Apply the transformation to a vector. You can use the same array to store both the input and output.
- `obj.TransformVector (double in[3], double out[3])` - Apply the transformation to a double-precision vector. You can use the same array to store both the input and output.
- `float = obj.TransformFloatVector (float x, float y, float z)` - Apply the transformation to an (x,y,z) vector. Use this if you are programming in python, tcl or Java.
- `float = obj.TransformFloatVector (float vec[3])` - Apply the transformation to an (x,y,z) vector. Use this if you are programming in python, tcl or Java.
- `double = obj.TransformDoubleVector (double x, double y, double z)` - Apply the transformation to a double-precision (x,y,z) vector. Use this if you are programming in python, tcl or Java.
- `double = obj.TransformDoubleVector (double vec[3])` - Apply the transformation to a double-precision (x,y,z) vector. Use this if you are programming in python, tcl or Java.
- `obj.TransformPoints (vtkPoints inPts, vtkPoints outPts)` - Apply the transformation to a series of points, and append the results to outPts.
- `obj.TransformNormals (vtkDataArray inNms, vtkDataArray outNms)` - Apply the transformation to a series of normals, and append the results to outNms.
- `obj.TransformVectors (vtkDataArray inVrs, vtkDataArray outVrs)` - Apply the transformation to a series of vectors, and append the results to outVrs.
- `obj.TransformPointsNormalsVectors (vtkPoints inPts, vtkPoints outPts, vtkDataArray inNms, vtkDataAr` - Apply the transformation to a combination of points, normals and vectors.
- `vtkLinearTransform = obj.GetLinearInverse ()` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformNormal (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformNormal (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformVector (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformVector (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.

30.74 vtkLogLookupTable

30.74.1 Usage

This class is an empty shell. Use vtkLookupTable with SetScaleToLog10() instead.

To create an instance of class vtkLogLookupTable, simply invoke its constructor as follows

```
obj = vtkLogLookupTable
```

30.74.2 Methods

The class vtkLogLookupTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLogLookupTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLogLookupTable = obj.NewInstance ()`
- `vtkLogLookupTable = obj.SafeDownCast (vtkObject o)`

30.75 vtkLongArray

30.75.1 Usage

vtkLongArray is an array of values of type long. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkLongArray, simply invoke its constructor as follows

```
obj = vtkLongArray
```

30.75.2 Methods

The class vtkLongArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLongArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLongArray = obj.NewInstance ()`
- `vtkLongArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, long tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, long tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, long tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (long tuple)` - Get the data at a particular index.
- `long = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, long value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.

- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, long f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (long f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `obj.SetArray (long array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (long array, vtkIdType size, int save, int deleteMethod)`

30.76 vtkLongLongArray

30.76.1 Usage

vtkLongLongArray is an array of values of type long long. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkLongLongArray, simply invoke its constructor as follows

```
obj = vtkLongLongArray
```

30.76.2 Methods

The class vtkLongLongArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLongLongArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLongLongArray = obj.NewInstance ()`
- `vtkLongLongArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `long = obj.long GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, long long value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, long long f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (long long f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.

- `long = obj.long WritePointer (vtkIdType id, vtkIdType number)` - Get the address of a particular data index. Performs no checks to verify that the memory has been allocated etc.
- `long = obj.long GetPointer (vtkIdType id)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.

30.77 vtkLookupTable

30.77.1 Usage

vtkLookupTable is an object that is used by mapper objects to map scalar values into rgba (red-green-blue-alpha transparency) color specification, or rgba into scalar values. The color table can be created by direct insertion of color values, or by specifying hue, saturation, value, and alpha range and generating a table.

To create an instance of class vtkLookupTable, simply invoke its constructor as follows

```
obj = vtkLookupTable
```

30.77.2 Methods

The class vtkLookupTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLookupTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLookupTable = obj.NewInstance ()`
- `vtkLookupTable = obj.SafeDownCast (vtkObject o)`
- `int = obj.IsOpaque ()` - Return true if all of the values defining the mapping have an opacity equal to 1. Default implementation return true.
- `int = obj.Allocate (int sz, int ext)` - Allocate a color table of specified size.
- `obj.Build ()` - Generate lookup table from hue, saturation, value, alpha min/max values. Table is built from linear ramp of each value.
- `obj.ForceBuild ()` - Force the lookup table to regenerate from hue, saturation, value, and alpha min/max values. Table is built from a linear ramp of each value. ForceBuild() is useful if a lookup table has been defined manually (using SetTableValue) and then an application decides to rebuild the lookup table using the implicit process.
- `obj.SetRamp (int )` - Set the shape of the table ramp to either linear or S-curve. The default is S-curve, which tails off gradually at either end. The equation used for the S-curve is y = (sin((x - 1/2)*pi) + 1)/2, while the equation for the linear ramp is simply y = x. For an S-curve greyscale ramp, you should set NumberOfTableValues to 402 (which is 256*pi/2) to provide room for the tails of the ramp. The equation for the SQRT is y = sqrt(x).
- `obj.SetRampToLinear ()` - Set the shape of the table ramp to either linear or S-curve. The default is S-curve, which tails off gradually at either end. The equation used for the S-curve is y = (sin((x - 1/2)*pi) + 1)/2, while the equation for the linear ramp is simply y = x. For an S-curve greyscale ramp, you should set NumberOfTableValues to 402 (which is 256*pi/2) to provide room for the tails of the ramp. The equation for the SQRT is y = sqrt(x).

- `obj.SetRampToSCurve ()` - Set the shape of the table ramp to either linear or S-curve. The default is S-curve, which tails off gradually at either end. The equation used for the S-curve is y = (sin((x - 1/2)*pi) + 1)/2, while the equation for the linear ramp is simply y = x. For an S-curve greyscale ramp, you should set NumberOfTableValues to 402 (which is 256*pi/2) to provide room for the tails of the ramp. The equation for the SQRT is y = sqrt(x).

- `obj.SetRampToSQRT ()` - Set the shape of the table ramp to either linear or S-curve. The default is S-curve, which tails off gradually at either end. The equation used for the S-curve is y = (sin((x - 1/2)*pi) + 1)/2, while the equation for the linear ramp is simply y = x. For an S-curve greyscale ramp, you should set NumberOfTableValues to 402 (which is 256*pi/2) to provide room for the tails of the ramp. The equation for the SQRT is y = sqrt(x).

- `int = obj.GetRamp ()` - Set the shape of the table ramp to either linear or S-curve. The default is S-curve, which tails off gradually at either end. The equation used for the S-curve is y = (sin((x - 1/2)*pi) + 1)/2, while the equation for the linear ramp is simply y = x. For an S-curve greyscale ramp, you should set NumberOfTableValues to 402 (which is 256*pi/2) to provide room for the tails of the ramp. The equation for the SQRT is y = sqrt(x).

- `obj.SetScale (int scale)` - Set the type of scale to use, linear or logarithmic. The default is linear. If the scale is logarithmic, then the TableRange must not cross the value zero.

- `obj.SetScaleToLinear ()` - Set the type of scale to use, linear or logarithmic. The default is linear. If the scale is logarithmic, then the TableRange must not cross the value zero.

- `obj.SetScaleToLog10 ()` - Set the type of scale to use, linear or logarithmic. The default is linear. If the scale is logarithmic, then the TableRange must not cross the value zero.

- `int = obj.GetScale ()` - Set the type of scale to use, linear or logarithmic. The default is linear. If the scale is logarithmic, then the TableRange must not cross the value zero.

- `obj.SetTableRange (double r[2])` - Set/Get the minimum/maximum scalar values for scalar mapping. Scalar values less than minimum range value are clamped to minimum range value. Scalar values greater than maximum range value are clamped to maximum range value.

- `obj.SetTableRange (double min, double max)` - Set/Get the minimum/maximum scalar values for scalar mapping. Scalar values less than minimum range value are clamped to minimum range value. Scalar values greater than maximum range value are clamped to maximum range value.

- `double = obj. GetTableRange ()` - Set/Get the minimum/maximum scalar values for scalar mapping. Scalar values less than minimum range value are clamped to minimum range value. Scalar values greater than maximum range value are clamped to maximum range value.

- `obj.SetHueRange (double , double )` - Set the range in hue (using automatic generation). Hue ranges between [0,1].

- `obj.SetHueRange (double a[2])` - Set the range in hue (using automatic generation). Hue ranges between [0,1].

- `double = obj. GetHueRange ()` - Set the range in hue (using automatic generation). Hue ranges between [0,1].

- `obj.SetSaturationRange (double , double )` - Set the range in saturation (using automatic generation). Saturation ranges between [0,1].

- `obj.SetSaturationRange (double a[2])` - Set the range in saturation (using automatic generation). Saturation ranges between [0,1].

- `double = obj. GetSaturationRange ()` - Set the range in saturation (using automatic generation). Saturation ranges between [0,1].

- `obj.SetValueRange (double , double )` - Set the range in value (using automatic generation). Value ranges between [0,1].
- `obj.SetValueRange (double  a[2])` - Set the range in value (using automatic generation). Value ranges between [0,1].
- `double = obj. GetValueRange ()` - Set the range in value (using automatic generation). Value ranges between [0,1].
- `obj.SetAlphaRange (double , double )` - Set the range in alpha (using automatic generation). Alpha ranges from [0,1].
- `obj.SetAlphaRange (double  a[2])` - Set the range in alpha (using automatic generation). Alpha ranges from [0,1].
- `double = obj. GetAlphaRange ()` - Set the range in alpha (using automatic generation). Alpha ranges from [0,1].
- `obj.GetColor (double x, double rgb[3])` - Map one value through the lookup table and return the color as an RGB array of doubles between 0 and 1.
- `double = obj.GetOpacity (double v)` - Map one value through the lookup table and return the alpha value (the opacity) as a double between 0 and 1.
- `vtkIdType = obj.GetIndex (double v)` - Return the table index associated with a particular value.
- `obj.SetNumberOfTableValues (vtkIdType number)` - Specify the number of values (i.e., colors) in the lookup table.
- `vtkIdType = obj.GetNumberOfTableValues ()` - Specify the number of values (i.e., colors) in the lookup table.
- `obj.SetTableValue (vtkIdType indx, double rgba[4])` - Directly load color into lookup table. Use [0,1] double values for color component specification. Make sure that you've either used the Build() method or used SetNumberOfTableValues() prior to using this method.
- `obj.SetTableValue (vtkIdType indx, double r, double g, double b, double a)` - Directly load color into lookup table. Use [0,1] double values for color component specification.
- `double = obj.GetTableValue (vtkIdType id)` - Return a rgba color value for the given index into the lookup table. Color components are expressed as [0,1] double values.
- `obj.GetTableValue (vtkIdType id, double rgba[4])` - Return a rgba color value for the given index into the lookup table. Color components are expressed as [0,1] double values.
- `double = obj.GetRange ()` - Sets/Gets the range of scalars which will be mapped. This is a duplicate of Get/SetTableRange.
- `obj.SetRange (double min, double max)` - Sets/Gets the range of scalars which will be mapped. This is a duplicate of Get/SetTableRange.
- `obj.SetRange (double rng[2])` - Sets/Gets the range of scalars which will be mapped. This is a duplicate of Get/SetTableRange.
- `obj.SetNumberOfColors (vtkIdType )` - Set the number of colors in the lookup table. Use SetNumberOfTableValues() instead, it can be used both before and after the table has been built whereas SetNumberOfColors() has no effect after the table has been built.
- `vtkIdType = obj.GetNumberOfColorsMinValue ()` - Set the number of colors in the lookup table. Use SetNumberOfTableValues() instead, it can be used both before and after the table has been built whereas SetNumberOfColors() has no effect after the table has been built.

- `vtkIdType = obj.GetNumberOfColorsMaxValue ()` - Set the number of colors in the lookup table. Use SetNumberOfTableValues() instead, it can be used both before and after the table has been built whereas SetNumberOfColors() has no effect after the table has been built.

- `vtkIdType = obj.GetNumberOfColors ()` - Set the number of colors in the lookup table. Use SetNumberOfTableValues() instead, it can be used both before and after the table has been built whereas SetNumberOfColors() has no effect after the table has been built.

- `obj.SetTable (vtkUnsignedCharArray )` - Set/Get the internal table array that is used to map the scalars to colors. The table array is an unsigned char array with 4 components representing RGBA.

- `vtkUnsignedCharArray = obj.GetTable ()` - Set/Get the internal table array that is used to map the scalars to colors. The table array is an unsigned char array with 4 components representing RGBA.

- `obj.DeepCopy (vtkLookupTable lut)` - Copy the contents from another LookupTable

- `int = obj.UsingLogScale ()`

30.78 vtkLookupTableWithEnabling

30.78.1 Usage

vtkLookupTableWithEnabling "disables" or "grays out" output colors based on whether the given value in EnabledArray is "0" or not.

To create an instance of class vtkLookupTableWithEnabling, simply invoke its constructor as follows

```
obj = vtkLookupTableWithEnabling
```

30.78.2 Methods

The class vtkLookupTableWithEnabling has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLookupTableWithEnabling class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkLookupTableWithEnabling = obj.NewInstance ()`

- `vtkLookupTableWithEnabling = obj.SafeDownCast (vtkObject o)`

- `vtkDataArray = obj.GetEnabledArray ()` - This must be set before MapScalars() is called. Indices of this array must map directly to those in the scalars array passed to MapScalars(). Values of 0 in the array indicate the color should be desaturatated.

- `obj.SetEnabledArray (vtkDataArray enabledArray)` - This must be set before MapScalars() is called. Indices of this array must map directly to those in the scalars array passed to MapScalars(). Values of 0 in the array indicate the color should be desaturatated.

- `obj.DisableColor (char r, char g, char b, string rd, string gd, string bd)` - A convenience method for taking a color and desaturating it.

30.79 vtkMath

30.79.1 Usage

vtkMath provides methods to perform common math operations. These include providing constants such as Pi; conversion from degrees to radians; vector operations such as dot and cross products and vector norm; matrix determinant for 2x2 and 3x3 matrices; univariate polynomial solvers; and for random number generation (for backward compatibility only).

To create an instance of class vtkMath, simply invoke its constructor as follows

```
obj = vtkMath
```

30.79.2 Methods

The class vtkMath has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMath class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMath = obj.NewInstance ()`
- `vtkMath = obj.SafeDownCast (vtkObject o)`

30.80 vtkMatrix3x3

30.80.1 Usage

vtkMatrix3x3 is a class to represent and manipulate 3x3 matrices. Specifically, it is designed to work on 3x3 transformation matrices found in 2D rendering using homogeneous coordinates [x y w].

To create an instance of class vtkMatrix3x3, simply invoke its constructor as follows

```
obj = vtkMatrix3x3
```

30.80.2 Methods

The class vtkMatrix3x3 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMatrix3x3 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMatrix3x3 = obj.NewInstance ()`
- `vtkMatrix3x3 = obj.SafeDownCast (vtkObject o)`
- `obj.DeepCopy (vtkMatrix3x3 source)` - Non-static member function. Assigns *from* elements array
- `obj.DeepCopy (double Elements[9])` - Set all of the elements to zero.
- `obj.Zero ()` - Set equal to Identity matrix
- `obj.Identity ()` - Matrix Inversion (adapted from Richard Carling in "Graphics Gems," Academic Press, 1990).

- `obj.Invert ()` - Transpose the matrix and put it into out.
- `obj.Transpose ()` - Multiply a homogeneous coordinate by this matrix, i.e. out = A*in. The in[3] and out[3] can be the same array.
- `obj.MultiplyPoint (float in[3], float out[3])` - Multiply a homogeneous coordinate by this matrix, i.e. out = A*in. The in[3] and out[3] can be the same array.
- `obj.MultiplyPoint (double in[3], double out[3])` - Multiplies matrices a and b and stores the result in c (c=a*b).
- `obj.Adjoint (vtkMatrix3x3 in, vtkMatrix3x3 out)` - Compute the determinant of the matrix and return it.
- `double = obj.Determinant ()` - Sets the element i,j in the matrix.
- `obj.SetElement (int i, int j, double value)` - Sets the element i,j in the matrix.
- `double = obj.GetElement (int i, int j) const`
- `bool = obj.IsIdentity ()`

30.81 vtkMatrix4x4

30.81.1 Usage

vtkMatrix4x4 is a class to represent and manipulate 4x4 matrices. Specifically, it is designed to work on 4x4 transformation matrices found in 3D rendering using homogeneous coordinates [x y z w].

To create an instance of class vtkMatrix4x4, simply invoke its constructor as follows

```
obj = vtkMatrix4x4
```

30.81.2 Methods

The class vtkMatrix4x4 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMatrix4x4 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMatrix4x4 = obj.NewInstance ()`
- `vtkMatrix4x4 = obj.SafeDownCast (vtkObject o)`
- `obj.DeepCopy (vtkMatrix4x4 source)` - Non-static member function. Assigns *from* elements array
- `obj.DeepCopy (double Elements[16])` - Set all of the elements to zero.
- `obj.Zero ()` - Set equal to Identity matrix
- `obj.Identity ()` - Matrix Inversion (adapted from Richard Carling in "Graphics Gems," Academic Press, 1990).
- `obj.Invert ()` - Transpose the matrix and put it into out.
- `obj.Transpose ()` - Multiply a homogeneous coordinate by this matrix, i.e. out = A*in. The in[4] and out[4] can be the same array.

- `obj.MultiplyPoint (float in[4], float out[4])` - Multiply a homogeneous coordinate by this matrix, i.e. out = A*in. The in[4] and out[4] can be the same array.
- `obj.MultiplyPoint (double in[4], double out[4])` - For use in Java, Python or Tcl. The default MultiplyPoint() uses a single-precision point.
- `float = obj.MultiplyPoint (float in[4])` - For use in Java, Python or Tcl. The default MultiplyPoint() uses a single-precision point.
- `float = obj.MultiplyFloatPoint (float in[4])` - For use in Java, Python or Tcl. The default MultiplyPoint() uses a single-precision point.
- `double = obj.MultiplyDoublePoint (double in[4])` - Multiplies matrices a and b and stores the result in c.
- `obj.Adjoint (vtkMatrix4x4 in, vtkMatrix4x4 out)` - Compute the determinant of the matrix and return it.
- `double = obj.Determinant ()` - Sets the element i,j in the matrix.
- `obj.SetElement (int i, int j, double value)` - Sets the element i,j in the matrix.
- `double = obj.GetElement (int i, int j) const`

30.82 vtkMatrixToHomogeneousTransform

30.82.1 Usage

This is a very simple class which allows a vtkMatrix4x4 to be used in place of a vtkHomogeneousTransform or vtkAbstractTransform. For example, if you use it as a proxy between a matrix and vtkTransformPolyDataFilter then any modifications to the matrix will automatically be reflected in the output of the filter.

To create an instance of class vtkMatrixToHomogeneousTransform, simply invoke its constructor as follows

```
obj = vtkMatrixToHomogeneousTransform
```

30.82.2 Methods

The class vtkMatrixToHomogeneousTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMatrixToHomogeneousTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMatrixToHomogeneousTransform = obj.NewInstance ()`
- `vtkMatrixToHomogeneousTransform = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkMatrix4x4 )`
- `vtkMatrix4x4 = obj.GetInput ()`
- `obj.Inverse ()` - The input matrix is left as-is, but the transformation matrix is inverted.
- `long = obj.GetMTime ()` - Get the MTime: this is the bit of magic that makes everything work.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make a new transform of the same type.
- `obj.SetMatrix (vtkMatrix4x4 matrix)` - @deprecated This method is deprecated.

30.83 vtkMatrixToLinearTransform

30.83.1 Usage

This is a very simple class which allows a vtkMatrix4x4 to be used in place of a vtkLinearTransform or vtkAbstractTransform. For example, if you use it as a proxy between a matrix and vtkTransformPolyDataFilter then any modifications to the matrix will automatically be reflected in the output of the filter.

To create an instance of class vtkMatrixToLinearTransform, simply invoke its constructor as follows

```
obj = vtkMatrixToLinearTransform
```

30.83.2 Methods

The class vtkMatrixToLinearTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMatrixToLinearTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMatrixToLinearTransform = obj.NewInstance ()`
- `vtkMatrixToLinearTransform = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkMatrix4x4 )` - Set the input matrix. Any modifications to the matrix will be reflected in the transformation.
- `vtkMatrix4x4 = obj.GetInput ()` - Set the input matrix. Any modifications to the matrix will be reflected in the transformation.
- `obj.Inverse ()` - The input matrix is left as-is, but the transformation matrix is inverted.
- `long = obj.GetMTime ()` - Get the MTime: this is the bit of magic that makes everything work.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make a new transform of the same type.
- `obj.SetMatrix (vtkMatrix4x4 matrix)` - @deprecated This method is deprecated.

30.84 vtkMinimalStandardRandomSequence

30.84.1 Usage

vtkMinimalStandardRandomSequence is a sequence of statistically independent pseudo random numbers uniformly distributed between 0.0 and 1.0.

The sequence is generated by a prime modulus multiplicative linear congruential generator (PMMLCG) or "Lehmer generator" with multiplier 16807 and prime modulus 2$\hat{}$(31)-1. The authors calls it "minimal standard random number generator"

ref: "Random Number Generators: Good Ones are Hard to Find," by Stephen K. Park and Keith W. Miller in Communications of the ACM, 31, 10 (Oct. 1988) pp. 1192-1201. Code is at page 1195, "Integer version 2"

Correctness test is described in first column, page 1195: A seed of 1 at step 1 should give a seed of 1043618065 at step 10001.

To create an instance of class vtkMinimalStandardRandomSequence, simply invoke its constructor as follows

```
obj = vtkMinimalStandardRandomSequence
```

30.84.2 Methods

The class vtkMinimalStandardRandomSequence has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMinimalStandardRandomSequence class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMinimalStandardRandomSequence = obj.NewInstance ()`
- `vtkMinimalStandardRandomSequence = obj.SafeDownCast (vtkObject o)`
- `obj.SetSeed (int value)` - Set the seed of the random sequence. The following pre-condition is stated page 1197, second column: valid_seed: value¿=1 && value¡=2147483646 2147483646=(2$\hat{}$31)-2 This method does not have this criterium as a pre-condition (ie it will not fail if an incorrect seed value is passed) but the value is silently changed to fit in the valid range [1,2147483646]. 2147483646 is added to a null or negative value. 2147483647 is changed to be 1 (ie 2147483646 is substracted). Implementation note: it also performs 3 calls to Next() to avoid the bad property that the first random number is proportional to the seed value.
- `obj.SetSeedOnly (int value)` - Set the seed of the random sequence. There is no extra internal ajustment. Only useful for writing correctness test. The following pre-condition is stated page 1197, second column 2147483646=(2$\hat{}$31)-2 This method does not have this criterium as a pre-condition (ie it will not fail if an incorrect seed value is passed) but the value is silently changed to fit in the valid range [1,2147483646]. 2147483646 is added to a null or negative value. 2147483647 is changed to be 1 (ie 2147483646 is substracted).
- `int = obj.GetSeed ()` - Get the seed of the random sequence. Only useful for writing correctness test.
- `double = obj.GetValue ()` - Current value
- `obj.Next ()` - Move to the next number in the random sequence.
- `double = obj.GetRangeValue (double rangeMin, double rangeMax)` - Convenient method to return a value in a specific range from the range [0,1. There is an initial implementation that can be overridden by a subclass. There is no pre-condition on the range: - it can be in increasing order: rangeMin¡rangeMax - it can be empty: rangeMin=rangeMax - it can be in decreasing order: rangeMin¿rangeMax

  (rangeMin¡=rangeMax && result¿=rangeMin && result¡=rangeMax) —— (rangeMax¡=rangeMin && result¿=rangeMax && result¡=rangeMin)

30.85 vtkMultiThreader

30.85.1 Usage

vtkMultithreader is a class that provides support for multithreaded execution using sproc() on an SGI, or pthread_create on any platform supporting POSIX threads. This class can be used to execute a single method on multiple threads, or to specify a method per thread.

To create an instance of class vtkMultiThreader, simply invoke its constructor as follows

```
obj = vtkMultiThreader
```

30.85.2 Methods

The class vtkMultiThreader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiThreader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiThreader = obj.NewInstance ()`
- `vtkMultiThreader = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfThreads (int )` - Get/Set the number of threads to create. It will be clamped to the range 1 - VTK_MAX_THREADS, so the caller of this method should check that the requested number of threads was accepted.
- `int = obj.GetNumberOfThreadsMinValue ()` - Get/Set the number of threads to create. It will be clamped to the range 1 - VTK_MAX_THREADS, so the caller of this method should check that the requested number of threads was accepted.
- `int = obj.GetNumberOfThreadsMaxValue ()` - Get/Set the number of threads to create. It will be clamped to the range 1 - VTK_MAX_THREADS, so the caller of this method should check that the requested number of threads was accepted.
- `int = obj.GetNumberOfThreads ()` - Get/Set the number of threads to create. It will be clamped to the range 1 - VTK_MAX_THREADS, so the caller of this method should check that the requested number of threads was accepted.

30.86 vtkMutexLock

30.86.1 Usage

vtkMutexLock allows the locking of variables which are accessed through different threads. This header file also defines vtkSimpleMutexLock which is not a subclass of vtkObject.

To create an instance of class vtkMutexLock, simply invoke its constructor as follows

```
obj = vtkMutexLock
```

30.86.2 Methods

The class vtkMutexLock has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMutexLock class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMutexLock = obj.NewInstance ()`
- `vtkMutexLock = obj.SafeDownCast (vtkObject o)`
- `obj.Lock (void )` - Lock the vtkMutexLock
- `obj.Unlock (void )` - Unlock the vtkMutexLock

30.87 vtkObject

30.87.1 Usage

vtkObject is the base class for most objects in the visualization toolkit. vtkObject provides methods for tracking modification time, debugging, printing, and event callbacks. Most objects created within the VTK framework should be a subclass of vtkObject or one of its children. The few exceptions tend to be very small helper classes that usually never get instantiated or situations where multiple inheritance gets in the way. vtkObject also performs reference counting: objects that are reference counted exist as long as another object uses them. Once the last reference to a reference counted object is removed, the object will spontaneously destruct.

To create an instance of class vtkObject, simply invoke its constructor as follows

```
obj = vtkObject
```

30.87.2 Methods

The class vtkObject has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkObject class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkObject = obj.NewInstance ()`
- `vtkObject = obj.SafeDownCast (vtkObject o)`
- `obj.DebugOn ()` - Turn debugging output on.
- `obj.DebugOff ()` - Turn debugging output off.
- `char = obj.GetDebug ()` - Get the value of the debug flag.
- `obj.SetDebug (char debugFlag)` - Set the value of the debug flag. A non-zero value turns debugging on.
- `obj.Modified ()` - Update the modification time for this object. Many filters rely on the modification time to determine if they need to recompute their data. The modification time is a unique monotonically increasing unsigned long integer.
- `long = obj.GetMTime ()` - Return this object's modified time.
- `obj.RemoveObserver (long tag)` - Allow people to add/remove/invoke observers (callbacks) to any VTK object. This is an implementation of the subject/observer design pattern. An observer is added by specifying an event to respond to and a vtkCommand to execute. It returns an unsigned long tag which can be used later to remove the event or retrieve the command. When events are invoked, the observers are called in the order they were added. If a priority value is specified, then the higher priority commands are called first. A command may set an abort flag to stop processing of the event. (See vtkCommand.h for more information.)
- `obj.RemoveObservers (long event)` - Allow people to add/remove/invoke observers (callbacks) to any VTK object. This is an implementation of the subject/observer design pattern. An observer is added by specifying an event to respond to and a vtkCommand to execute. It returns an unsigned long tag which can be used later to remove the event or retrieve the command. When events are invoked, the observers are called in the order they were added. If a priority value is specified, then the higher priority commands are called first. A command may set an abort flag to stop processing of the event. (See vtkCommand.h for more information.)

- `obj.RemoveObservers (string event)` - Allow people to add/remove/invoke observers (callbacks) to any VTK object. This is an implementation of the subject/observer design pattern. An observer is added by specifying an event to respond to and a vtkCommand to execute. It returns an unsigned long tag which can be used later to remove the event or retrieve the command. When events are invoked, the observers are called in the order they were added. If a priority value is specified, then the higher priority commands are called first. A command may set an abort flag to stop processing of the event. (See vtkCommand.h for more information.)
- `obj.RemoveAllObservers ()` - Allow people to add/remove/invoke observers (callbacks) to any VTK object. This is an implementation of the subject/observer design pattern. An observer is added by specifying an event to respond to and a vtkCommand to execute. It returns an unsigned long tag which can be used later to remove the event or retrieve the command. When events are invoked, the observers are called in the order they were added. If a priority value is specified, then the higher priority commands are called first. A command may set an abort flag to stop processing of the event. (See vtkCommand.h for more information.)
- `int = obj.HasObserver (long event)` - Allow people to add/remove/invoke observers (callbacks) to any VTK object. This is an implementation of the subject/observer design pattern. An observer is added by specifying an event to respond to and a vtkCommand to execute. It returns an unsigned long tag which can be used later to remove the event or retrieve the command. When events are invoked, the observers are called in the order they were added. If a priority value is specified, then the higher priority commands are called first. A command may set an abort flag to stop processing of the event. (See vtkCommand.h for more information.)
- `int = obj.HasObserver (string event)` - Allow people to add/remove/invoke observers (callbacks) to any VTK object. This is an implementation of the subject/observer design pattern. An observer is added by specifying an event to respond to and a vtkCommand to execute. It returns an unsigned long tag which can be used later to remove the event or retrieve the command. When events are invoked, the observers are called in the order they were added. If a priority value is specified, then the higher priority commands are called first. A command may set an abort flag to stop processing of the event. (See vtkCommand.h for more information.)
- `int = obj.InvokeEvent (long event)`
- `int = obj.InvokeEvent (string event)`

30.88 vtkObjectBase

30.88.1 Usage

vtkObjectBase is the base class for all reference counted classes in the VTK. These classes include vtkCommand classes, vtkInformationKey classes, and vtkObject classes.

vtkObjectBase performs reference counting: objects that are reference counted exist as long as another object uses them. Once the last reference to a reference counted object is removed, the object will spontaneously destruct.

Constructor and destructor of the subclasses of vtkObjectBase should be protected, so that only New() and UnRegister() actually call them. Debug leaks can be used to see if there are any objects left with nonzero reference count.

To create an instance of class vtkObjectBase, simply invoke its constructor as follows

```
obj = vtkObjectBase
```

30.88.2 Methods

The class vtkObjectBase has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When

in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkObjectBase class.

- `string = obj.GetClassName () const` - Return the class name as a string. This method is defined in all subclasses of vtkObjectBase with the vtkTypeRevisionMacro found in vtkSetGet.h.
- `int = obj.IsA (string name)` - Return 1 if this class is the same type of (or a subclass of) the named class. Returns 0 otherwise. This method works in combination with vtkTypeRevisionMacro found in vtkSetGet.h.
- `obj.Delete ()` - Delete a VTK object. This method should always be used to delete an object when the New() method was used to create it. Using the C++ delete method will not work with reference counting.
- `obj.FastDelete ()` - Delete a reference to this object. This version will not invoke garbage collection and can potentially leak the object if it is part of a reference loop. Use this method only when it is known that the object has another reference and would not be collected if a full garbage collection check were done.
- `obj.Register (vtkObjectBase o)` - Increase the reference count (mark as used by another object).
- `obj.UnRegister (vtkObjectBase o)` - Decrease the reference count (release by another object). This has the same effect as invoking Delete() (i.e., it reduces the reference count by 1).
- `int = obj.GetReferenceCount ()` - Sets the reference count. (This is very dangerous, use with care.)
- `obj.SetReferenceCount (int )` - Sets the reference count. (This is very dangerous, use with care.)

30.89 vtkObjectFactory

30.89.1 Usage

vtkObjectFactory is used to create vtk objects. The base class vtkObjectFactory contains a static method CreateInstance which is used to create vtk objects from the list of registered vtkObjectFactory sub-classes. The first time CreateInstance is called, all dll's or shared libraries in the environment variable VTK_AUTOLOAD_PATH are loaded into the current process. The C functions vtkLoad, vtkGetFactoryCompilerUsed, and vtkGetFactoryVersion are called on each dll. To implement these functions in a shared library or dll, use the macro: VTK_FACTORY_INTERFACE_IMPLEMENT. VTK_AUTOLOAD_PATH is an environment variable containing a colon separated (semi-colon on win32) list of paths.

The vtkObjectFactory can be use to override the creation of any object in VTK with a sub-class of that object. The factories can be registered either at run time with the VTK_AUTOLOAD_PATH, or at compile time with the vtkObjectFactory::RegisterFactory method.

To create an instance of class vtkObjectFactory, simply invoke its constructor as follows

```
obj = vtkObjectFactory
```

30.89.2 Methods

The class vtkObjectFactory has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkObjectFactory class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkObjectFactory = obj.NewInstance ()`

- `vtkObjectFactory = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetVTKSourceVersion ()` - All sub-classes of vtkObjectFactory should must return the version of VTK they were built with. This should be implemented with the macro VTK_SOURCE_VERSION and NOT a call to vtkVersion::GetVTKSourceVersion. As the version needs to be compiled into the file as a string constant. This is critical to determine possible incompatible dynamic factory loads.
- `string = obj.GetDescription ()` - Return a descriptive string describing the factory.
- `int = obj.GetNumberOfOverrides ()` - Return number of overrides this factory can create.
- `string = obj.GetClassOverrideName (int index)` - Return the name of a class override at the given index.
- `string = obj.GetClassOverrideWithName (int index)` - Return the name of the class that will override the class at the given index
- `int = obj.GetEnableFlag (int index)` - Return the enable flag for the class at the given index.
- `string = obj.GetOverrideDescription (int index)` - Return the description for a the class override at the given index.
- `obj.SetEnableFlag (int flag, string className, string subclassName)` - Set and Get the Enable flag for the specific override of className. if subclassName is null, then it is ignored.
- `int = obj.GetEnableFlag (string className, string subclassName)` - Set and Get the Enable flag for the specific override of className. if subclassName is null, then it is ignored.
- `int = obj.HasOverride (string className)` - Return 1 if this factory overrides the given class name, 0 otherwise.
- `int = obj.HasOverride (string className, string subclassName)` - Return 1 if this factory overrides the given class name, 0 otherwise.
- `obj.Disable (string className)` - Set all enable flags for the given class to 0. This will mean that the factory will stop producing class with the given name.
- `string = obj.GetLibraryPath ()` - This returns the path to a dynamically loaded factory.

30.90 vtkObjectFactoryCollection

30.90.1 Usage

vtkObjectFactoryCollection is an object that creates and manipulates lists of object of type vtkObjectFactory.

To create an instance of class vtkObjectFactoryCollection, simply invoke its constructor as follows

```
obj = vtkObjectFactoryCollection
```

30.90.2 Methods

The class vtkObjectFactoryCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkObjectFactoryCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkObjectFactoryCollection = obj.NewInstance ()`
- `vtkObjectFactoryCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkObjectFactory t)` - Get the next ObjectFactory in the list. Return NULL when the end of the list is reached.
- `vtkObjectFactory = obj.GetNextItem ()`

30.91 vtkOutputWindow

30.91.1 Usage

This class is used to encapsulate all text output, so that it will work with operating systems that have a stdout and stderr, and ones that do not. (i.e windows does not). Sub-classes can be provided which can redirect the output to a window.

To create an instance of class vtkOutputWindow, simply invoke its constructor as follows

```
obj = vtkOutputWindow
```

30.91.2 Methods

The class vtkOutputWindow has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOutputWindow class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOutputWindow = obj.NewInstance ()`
- `vtkOutputWindow = obj.SafeDownCast (vtkObject o)`
- `obj.DisplayText (string )` - Display the text. Four virtual methods exist, depending on the type of message to display. This allows redirection or reformatting of the messages. The default implementation uses DisplayText for all.
- `obj.DisplayErrorText (string )` - Display the text. Four virtual methods exist, depending on the type of message to display. This allows redirection or reformatting of the messages. The default implementation uses DisplayText for all.
- `obj.DisplayWarningText (string )` - Display the text. Four virtual methods exist, depending on the type of message to display. This allows redirection or reformatting of the messages. The default implementation uses DisplayText for all.
- `obj.DisplayGenericWarningText (string )` - Display the text. Four virtual methods exist, depending on the type of message to display. This allows redirection or reformatting of the messages. The default implementation uses DisplayText for all.
- `obj.DisplayDebugText (string )`
- `obj.PromptUserOn ()` - If PromptUser is set to true then each time a line of text is displayed, the user is asked if they want to keep getting messages.
- `obj.PromptUserOff ()` - If PromptUser is set to true then each time a line of text is displayed, the user is asked if they want to keep getting messages.
- `obj.SetPromptUser (int )` - If PromptUser is set to true then each time a line of text is displayed, the user is asked if they want to keep getting messages.

30.92 vtkOverrideInformation

30.92.1 Usage

vtkOverrideInformation is used to represent the information about a class which is overriden in a vtkObjectFactory.

To create an instance of class vtkOverrideInformation, simply invoke its constructor as follows

```
obj = vtkOverrideInformation
```

30.92.2 Methods

The class vtkOverrideInformation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOverrideInformation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOverrideInformation = obj.NewInstance ()`
- `vtkOverrideInformation = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetClassOverrideName ()` - Returns the name of the class that will override the class. For example, if you had a factory that provided an override for vtkVertex called vtkMyVertex, then this would return "vtkMyVertex"
- `string = obj.GetClassOverrideWithName ()` - Return a human readable or GUI displayable description of this override.
- `string = obj.GetDescription ()` - Return the specific object factory that this override occurs in.
- `vtkObjectFactory = obj.GetObjectFactory ()` - Set the class override name
- `obj.SetClassOverrideName (string )` - Set the class override name
- `obj.SetClassOverrideWithName (string )` - Set the class override name Set the class override with name
- `obj.SetDescription (string )` - Set the class override name Set the class override with name Set the description

30.93 vtkOverrideInformationCollection

30.93.1 Usage

vtkOverrideInformationCollection is an object that creates and manipulates lists of objects of type vtkOverrideInformation.

To create an instance of class vtkOverrideInformationCollection, simply invoke its constructor as follows

```
obj = vtkOverrideInformationCollection
```

30.93.2 Methods

The class vtkOverrideInformationCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOverrideInformationCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOverrideInformationCollection = obj.NewInstance ()`
- `vtkOverrideInformationCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkOverrideInformation )` - Add a OverrideInformation to the list.
- `vtkOverrideInformation = obj.GetNextItem ()` - Get the next OverrideInformation in the list.

30.94 vtkParametricBoy

30.94.1 Usage

vtkParametricBoy generates Boy's surface. This is a Model of the projective plane without singularities. It was found by Werner Boy on assignment from David Hilbert.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricBoy, simply invoke its constructor as follows

```
obj = vtkParametricBoy
```

30.94.2 Methods

The class vtkParametricBoy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricBoy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricBoy = obj.NewInstance ()`
- `vtkParametricBoy = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Set/Get the scale factor for the z-coordinate. Default = 1/8, giving a nice shape.
- `obj.SetZScale (double )` - Set/Get the scale factor for the z-coordinate. Default = 1/8, giving a nice shape.
- `double = obj.GetZScale ()` - Set/Get the scale factor for the z-coordinate. Default = 1/8, giving a nice shape.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - Boy's surface.

  This function performs the mapping $f(u, v) \rightarrow (x, y, x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x, y, z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.

- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.95 vtkParametricConicSpiral

30.95.1 Usage

vtkParametricConicSpiral generates conic spiral surfaces. These can resemble sea shells, or may look like a torus "eating" its own tail.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricConicSpiral, simply invoke its constructor as follows

```
obj = vtkParametricConicSpiral
```

30.95.2 Methods

The class vtkParametricConicSpiral has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricConicSpiral class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricConicSpiral = obj.NewInstance ()`
- `vtkParametricConicSpiral = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Set/Get the scale factor. Default = 0.2
- `obj.SetA (double )` - Set/Get the scale factor. Default = 0.2
- `double = obj.GetA ()` - Set/Get the scale factor. Default = 0.2
- `obj.SetB (double )` - Set/Get the A function coefficient (see equation below). Default = 1.
- `double = obj.GetB ()` - Set/Get the A function coefficient (see equation below). Default = 1.
- `obj.SetC (double )` - Set/Get the B function coefficient (see equation below). Default = 0.1.
- `double = obj.GetC ()` - Set/Get the B function coefficient (see equation below). Default = 0.1.
- `obj.SetN (double )` - Set/Get the C function coefficient (see equation below). Default = 2.
- `double = obj.GetN ()` - Set/Get the C function coefficient (see equation below). Default = 2.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - A conic spiral surface.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.

- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.96 vtkParametricCrossCap

30.96.1 Usage

vtkParametricCrossCap generates a cross-cap which is a non-orientable self-intersecting single-sided surface. This is one possible image of a projective plane in three-space.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricCrossCap, simply invoke its constructor as follows

```
obj = vtkParametricCrossCap
```

30.96.2 Methods

The class vtkParametricCrossCap has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricCrossCap class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricCrossCap = obj.NewInstance ()`
- `vtkParametricCrossCap = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - A cross-cap.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.

- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - A cross-cap.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.

- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.97 vtkParametricDini

30.97.1 Usage

vtkParametricDini generates Dini's surface. Dini's surface is a surface that posesses constant negative Gaussian curvature

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricDini, simply invoke its constructor as follows

```
obj = vtkParametricDini
```

30.97.2 Methods

The class vtkParametricDini has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricDini class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricDini = obj.NewInstance ()`
- `vtkParametricDini = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Set/Get the scale factor. Default = 1.
- `obj.SetA (double )` - Set/Get the scale factor. Default = 1.
- `double = obj.GetA ()` - Set/Get the scale factor. Default = 1.
- `obj.SetB (double )` - Set/Get the scale factor. Default = 0.2
- `double = obj.GetB ()` - Set/Get the scale factor. Default = 0.2
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - Dini's surface.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.

- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.98 vtkParametricEllipsoid

30.98.1 Usage

vtkParametricEllipsoid generates an ellipsoid. If all the radii are the same, we have a sphere. An oblate spheroid occurs if RadiusX = RadiusY ¿ RadiusZ. Here the Z-axis forms the symmetry axis. To a first approximation, this is the shape of the earth. A prolate spheroid occurs if RadiusX = RadiusY ¡ RadiusZ.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricEllipsoid, simply invoke its constructor as follows

```
obj = vtkParametricEllipsoid
```

30.98.2 Methods

The class vtkParametricEllipsoid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricEllipsoid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricEllipsoid = obj.NewInstance ()`
- `vtkParametricEllipsoid = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Set/Get the scaling factor for the x-axis. Default = 1.
- `obj.SetXRadius (double )` - Set/Get the scaling factor for the x-axis. Default = 1.
- `double = obj.GetXRadius ()` - Set/Get the scaling factor for the x-axis. Default = 1.
- `obj.SetYRadius (double )` - Set/Get the scaling factor for the y-axis. Default = 1.
- `double = obj.GetYRadius ()` - Set/Get the scaling factor for the y-axis. Default = 1.
- `obj.SetZRadius (double )` - Set/Get the scaling factor for the z-axis. Default = 1.
- `double = obj.GetZRadius ()` - Set/Get the scaling factor for the z-axis. Default = 1.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - An ellipsoid.

  This function performs the mapping $f(u, v) \rightarrow (x, y, x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x, y, z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.

- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.99 vtkParametricEnneper

30.99.1 Usage

vtkParametricEnneper generates Enneper's surface. Enneper's surface is a a self-intersecting minimal surface posessing constant negative Gaussian curvature

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricEnneper, simply invoke its constructor as follows

```
obj = vtkParametricEnneper
```

30.99.2 Methods

The class vtkParametricEnneper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricEnneper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricEnneper = obj.NewInstance ()`
- `vtkParametricEnneper = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Enneper's surface.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.

- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - Enneper's surface.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.

- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uv are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.100 vtkParametricFigure8Klein

30.100.1 Usage

vtkParametricFigure8Klein generates a figure-8 Klein bottle. A Klein bottle is a closed surface with no interior and only one surface. It is unrealisable in 3 dimensions without intersecting surfaces. It can be realised in 4 dimensions by considering the map $F : R^2 \rightarrow R^4$ given by:

- $f(u,v) = ((r * cos(v) + a) * cos(u), (r * cos(v) + a) * sin(u), r * sin(v) * cos(u/2), r * sin(v) * sin(u/2))$

This representation of the immersion in R^3 is formed by taking two Mobius strips and joining them along their boundaries, this is the so called "Figure-8 Klein Bottle"

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricFigure8Klein, simply invoke its constructor as follows

```
obj = vtkParametricFigure8Klein
```

30.100.2 Methods

The class vtkParametricFigure8Klein has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricFigure8Klein class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricFigure8Klein = obj.NewInstance ()`
- `vtkParametricFigure8Klein = obj.SafeDownCast (vtkObject o)`
- `obj.SetRadius (double )` - Set/Get the radius of the bottle.
- `double = obj.GetRadius ()` - Set/Get the radius of the bottle.
- `int = obj.GetDimension ()` - A Figure-8 Klein bottle.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - A Figure-8 Klein bottle.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.101 vtkParametricFunction

30.101.1 Usage

vtkParametricFunction is an abstract interface for functions defined by parametric mapping i.e. f(u,v,w)-¿(x,y,z) where u_min ¡= u ¡ u_max, v_min ¡= v ¡ v_max, w_min ¡= w ¡ w_max. (For notational convenience, we will write f(u)-¿x and assume that u means (u,v,w) and x means (x,y,z).)

The interface contains the pure virtual function, Evaluate(), that generates a point and the derivatives at that point which are then used to construct the surface. A second pure virtual function, EvaluateScalar(), can be used to generate a scalar for the surface. Finally, the GetDimension() virtual function is used to differentiate 1D, 2D, and 3D parametric functions. Since this abstract class defines a pure virtual API, its subclasses must implement the pure virtual functions GetDimension(), Evaluate() and EvaluateScalar().

This class has also methods for defining a range of parametric values (u,v,w).
.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.
To create an instance of class vtkParametricFunction, simply invoke its constructor as follows

```
obj = vtkParametricFunction
```

30.101.2 Methods

The class vtkParametricFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricFunction = obj.NewInstance ()`
- `vtkParametricFunction = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()`
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - Performs the mapping $f(uvw)-¿(Pt,Duvw)$f. This is a pure virtual function that must be instantiated in a derived class.

  uvw are the parameters, with u corresponding to uvw[0], v to uvw[1] and w to uvw[2] respectively. Pt is the returned Cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Note that the first three values in Duvw are Du, the next three are Dv, and the final three are Dw. Du Dv Dw are the partial derivatives of the function at the point Pt with respect to u, v and w respectively.
- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw. This is a pure virtual function that must be instantiated in a derived class.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v, and w. Pt, Duvw are obtained from Evaluate().
- `obj.SetMinimumU (double )` - Set/Get the minimum u-value.
- `double = obj.GetMinimumU ()` - Set/Get the minimum u-value.
- `obj.SetMaximumU (double )` - Set/Get the maximum u-value.
- `double = obj.GetMaximumU ()` - Set/Get the maximum u-value.
- `obj.SetMinimumV (double )` - Set/Get the minimum v-value.
- `double = obj.GetMinimumV ()` - Set/Get the minimum v-value.
- `obj.SetMaximumV (double )` - Set/Get the maximum v-value.
- `double = obj.GetMaximumV ()` - Set/Get the maximum v-value.
- `obj.SetMinimumW (double )` - Set/Get the minimum w-value.
- `double = obj.GetMinimumW ()` - Set/Get the minimum w-value.
- `obj.SetMaximumW (double )` - Set/Get the maximum w-value.
- `double = obj.GetMaximumW ()` - Set/Get the maximum w-value.
- `obj.SetJoinU (int )` - Set/Get the flag which joins the first triangle strip to the last one.

- `int = obj.GetJoinU ()` - Set/Get the flag which joins the first triangle strip to the last one.
- `obj.JoinUOn ()` - Set/Get the flag which joins the first triangle strip to the last one.
- `obj.JoinUOff ()` - Set/Get the flag which joins the first triangle strip to the last one.
- `obj.SetJoinV (int )` - Set/Get the flag which joins the the ends of the triangle strips.
- `int = obj.GetJoinV ()` - Set/Get the flag which joins the the ends of the triangle strips.
- `obj.JoinVOn ()` - Set/Get the flag which joins the the ends of the triangle strips.
- `obj.JoinVOff ()` - Set/Get the flag which joins the the ends of the triangle strips.
- `obj.SetTwistU (int )` - Set/Get the flag which joins the first triangle strip to the last one with a twist. JoinU must also be set if this is set. Used when building some non-orientable surfaces.
- `int = obj.GetTwistU ()` - Set/Get the flag which joins the first triangle strip to the last one with a twist. JoinU must also be set if this is set. Used when building some non-orientable surfaces.
- `obj.TwistUOn ()` - Set/Get the flag which joins the first triangle strip to the last one with a twist. JoinU must also be set if this is set. Used when building some non-orientable surfaces.
- `obj.TwistUOff ()` - Set/Get the flag which joins the first triangle strip to the last one with a twist. JoinU must also be set if this is set. Used when building some non-orientable surfaces.
- `obj.SetTwistV (int )` - Set/Get the flag which joins the ends of the triangle strips with a twist. JoinV must also be set if this is set. Used when building some non-orientable surfaces.
- `int = obj.GetTwistV ()` - Set/Get the flag which joins the ends of the triangle strips with a twist. JoinV must also be set if this is set. Used when building some non-orientable surfaces.
- `obj.TwistVOn ()` - Set/Get the flag which joins the ends of the triangle strips with a twist. JoinV must also be set if this is set. Used when building some non-orientable surfaces.
- `obj.TwistVOff ()` - Set/Get the flag which joins the ends of the triangle strips with a twist. JoinV must also be set if this is set. Used when building some non-orientable surfaces.
- `obj.SetClockwiseOrdering (int )` - Set/Get the flag which determines the ordering of the the vertices forming the triangle strips. The ordering of the points being inserted into the triangle strip is important because it determines the direction of the normals for the lighting. If set, the ordering is clockwise, otherwise the ordering is anti-clockwise. Default is true (i.e. clockwise ordering).
- `int = obj.GetClockwiseOrdering ()` - Set/Get the flag which determines the ordering of the the vertices forming the triangle strips. The ordering of the points being inserted into the triangle strip is important because it determines the direction of the normals for the lighting. If set, the ordering is clockwise, otherwise the ordering is anti-clockwise. Default is true (i.e. clockwise ordering).
- `obj.ClockwiseOrderingOn ()` - Set/Get the flag which determines the ordering of the the vertices forming the triangle strips. The ordering of the points being inserted into the triangle strip is important because it determines the direction of the normals for the lighting. If set, the ordering is clockwise, otherwise the ordering is anti-clockwise. Default is true (i.e. clockwise ordering).
- `obj.ClockwiseOrderingOff ()` - Set/Get the flag which determines the ordering of the the vertices forming the triangle strips. The ordering of the points being inserted into the triangle strip is important because it determines the direction of the normals for the lighting. If set, the ordering is clockwise, otherwise the ordering is anti-clockwise. Default is true (i.e. clockwise ordering).
- `obj.SetDerivativesAvailable (int )` - Set/Get the flag which determines whether derivatives are available from the parametric function (i.e., whether the Evaluate() method returns valid derivatives).

- `int = obj.GetDerivativesAvailable ()` - Set/Get the flag which determines whether derivatives are available from the parametric function (i.e., whether the Evaluate() method returns valid derivatives).
- `obj.DerivativesAvailableOn ()` - Set/Get the flag which determines whether derivatives are available from the parametric function (i.e., whether the Evaluate() method returns valid derivatives).
- `obj.DerivativesAvailableOff ()` - Set/Get the flag which determines whether derivatives are available from the parametric function (i.e., whether the Evaluate() method returns valid derivatives).

30.102 vtkParametricKlein

30.102.1 Usage

vtkParametricKlein generates a "classical" representation of a Klein bottle. A Klein bottle is a closed surface with no interior and only one surface. It is unrealisable in 3 dimensions without intersecting surfaces. It can be realised in 4 dimensions by considering the map $F : R^2 \rightarrow R^4$ given by:

- $f(u, v) = ((r * cos(v) + a) * cos(u), (r * cos(v) + a) * sin(u), r * sin(v) * cos(u/2), r * sin(v) * sin(u/2))$

The classical representation of the immersion in R^3 is returned by this function.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricKlein, simply invoke its constructor as follows

```
obj = vtkParametricKlein
```

30.102.2 Methods

The class vtkParametricKlein has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricKlein class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricKlein = obj.NewInstance ()`
- `vtkParametricKlein = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - A Klein bottle.

  This function performs the mapping $f(u, v) \rightarrow (x, y, x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x, y, z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - A Klein bottle.

  This function performs the mapping $f(u, v) \rightarrow (x, y, x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x, y, z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.103 vtkParametricMobius

30.103.1 Usage

vtkParametricMobius generates a Mobius strip.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricMobius, simply invoke its constructor as follows

```
obj = vtkParametricMobius
```

30.103.2 Methods

The class vtkParametricMobius has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricMobius class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricMobius = obj.NewInstance ()`
- `vtkParametricMobius = obj.SafeDownCast (vtkObject o)`
- `obj.SetRadius (double )` - Set/Get the radius of the Mobius strip.
- `double = obj.GetRadius ()` - Set/Get the radius of the Mobius strip.
- `int = obj.GetDimension ()` - The Mobius strip.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - The Mobius strip.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Du, Dv are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.104 vtkParametricRandomHills

30.104.1 Usage

vtkParametricRandomHills generates a surface covered with randomly placed hills.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.
To create an instance of class vtkParametricRandomHills, simply invoke its constructor as follows

```
obj = vtkParametricRandomHills
```

30.104.2 Methods

The class vtkParametricRandomHills has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricRandomHills class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricRandomHills = obj.NewInstance ()`
- `vtkParametricRandomHills = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Construct a surface of random hills with the following parameters: MinimumU = -10, MaximumU = 10, MinimumV = -10, MaximumV = 10, JoinU = 0, JoinV = 0, TwistU = 0, TwistV = 0; ClockwiseOrdering = 1, DerivativesAvailable = 0, Number of hills = 30, Variance of the hills 2.5 in both x- and y- directions, Scaling factor for the variances 1/3 in both x- and y- directions, Amplitude of each hill = 1, Scaling factor for the amplitude = 1/3, RandomSeed = 1, AllowRandomGeneration = 1.
- `obj.SetNumberOfHills (int )` - Set/Get the number of hills. Default is 30.
- `int = obj.GetNumberOfHills ()` - Set/Get the number of hills. Default is 30.
- `obj.SetHillXVariance (double )` - Set/Get the hill variance in the x-direction. Default is 2.5.
- `double = obj.GetHillXVariance ()` - Set/Get the hill variance in the x-direction. Default is 2.5.
- `obj.SetHillYVariance (double )` - Set/Get the hill variance in the y-direction. Default is 2.5.
- `double = obj.GetHillYVariance ()` - Set/Get the hill variance in the y-direction. Default is 2.5.
- `obj.SetHillAmplitude (double )` - Set/Get the hill amplitude (height). Default is 2.
- `double = obj.GetHillAmplitude ()` - Set/Get the hill amplitude (height). Default is 2.
- `obj.SetRandomSeed (int )` - Set/Get the Seed for the random number generator, a value of 1 will initialize the random number generator, a negative value will initialize it with the system time. Default is 1.
- `int = obj.GetRandomSeed ()` - Set/Get the Seed for the random number generator, a value of 1 will initialize the random number generator, a negative value will initialize it with the system time. Default is 1.
- `obj.SetAllowRandomGeneration (int )` - Set/Get the random generation flag. A value of 0 will disable the generation of random hills on the surface. This allows a reproducible shape to be generated. Any other value means that the generation of the hills will be done randomly. Default is 1.
- `int = obj.GetAllowRandomGeneration ()` - Set/Get the random generation flag. A value of 0 will disable the generation of random hills on the surface. This allows a reproducible shape to be generated. Any other value means that the generation of the hills will be done randomly. Default is 1.
- `obj.AllowRandomGenerationOn ()` - Set/Get the random generation flag. A value of 0 will disable the generation of random hills on the surface. This allows a reproducible shape to be generated. Any other value means that the generation of the hills will be done randomly. Default is 1.

- `obj.AllowRandomGenerationOff ()` - Set/Get the random generation flag. A value of 0 will disable the generation of random hills on the surface. This allows a reproducible shape to be generated. Any other value means that the generation of the hills will be done randomly. Default is 1.
- `obj.SetXVarianceScaleFactor (double )` - Set/Get the scaling factor for the variance in the x-direction. Default is 1/3.
- `double = obj.GetXVarianceScaleFactor ()` - Set/Get the scaling factor for the variance in the x-direction. Default is 1/3.
- `obj.SetYVarianceScaleFactor (double )` - Set/Get the scaling factor for the variance in the y-direction. Default is 1/3.
- `double = obj.GetYVarianceScaleFactor ()` - Set/Get the scaling factor for the variance in the y-direction. Default is 1/3.
- `obj.SetAmplitudeScaleFactor (double )` - Set/Get the scaling factor for the amplitude. Default is 1/3.
- `double = obj.GetAmplitudeScaleFactor ()` - Set/Get the scaling factor for the amplitude. Default is 1/3.
- `obj.GenerateTheHills (void )` - Generate the centers of the hills, their standard deviations and their amplitudes. This function creates a series of vectors representing the u, v coordinates of each hill, its variance in the u, v directions and the amplitude.

  NOTE: This function must be called whenever any of the parameters are changed.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - Construct a terrain consisting of randomly placed hills on a surface.

  It is assumed that the function GenerateTheHills() has been executed to build the vectors of coordinates required to generate the point Pt. Pt represents the sum of all the amplitudes over the space.

  This function performs the mapping $f(u, v) \rightarrow (x, y, x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x, y, z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the Cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.105 vtkParametricRoman

30.105.1 Usage

vtkParametricRoman generates Steiner's Roman Surface.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricRoman, simply invoke its constructor as follows

```
obj = vtkParametricRoman
```

30.105.2 Methods

The class vtkParametricRoman has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricRoman class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricRoman = obj.NewInstance ()`
- `vtkParametricRoman = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Construct Steiner's Roman Surface with the following parameters: MinimumU = 0, MaximumU = Pi, MinimumV = 0, MaximumV = Pi, JoinU = 1, JoinV = 1, TwistU = 1, TwistV = 0; ClockwiseOrdering = 1, DerivativesAvailable = 1, Radius = 1
- `obj.SetRadius (double )` - Set/Get the radius.
- `double = obj.GetRadius ()` - Set/Get the radius.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - Steiner's Roman Surface

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the Cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.106 vtkParametricSuperEllipsoid

30.106.1 Usage

vtkParametricSuperEllipsoid generates a superellipsoid. A superellipsoid is a versatile primitive that is controlled by two parameters n1 and n2. As special cases it can represent a sphere, square box, and closed cylindrical can.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

Also see: http://astronomy.swin.edu.au/ pbourke/surfaces/

To create an instance of class vtkParametricSuperEllipsoid, simply invoke its constructor as follows

```
obj = vtkParametricSuperEllipsoid
```

30.106.2 Methods

The class vtkParametricSuperEllipsoid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricSuperEllipsoid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricSuperEllipsoid = obj.NewInstance ()`
- `vtkParametricSuperEllipsoid = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Set/Get the scaling factor for the x-axis. Default = 1.
- `obj.SetXRadius (double )` - Set/Get the scaling factor for the x-axis. Default = 1.
- `double = obj.GetXRadius ()` - Set/Get the scaling factor for the x-axis. Default = 1.
- `obj.SetYRadius (double )` - Set/Get the scaling factor for the y-axis. Default = 1.
- `double = obj.GetYRadius ()` - Set/Get the scaling factor for the y-axis. Default = 1.
- `obj.SetZRadius (double )` - Set/Get the scaling factor for the z-axis. Default = 1.
- `double = obj.GetZRadius ()` - Set/Get the scaling factor for the z-axis. Default = 1.
- `obj.SetN1 (double )` - Set/Get the "squareness" parameter in the z axis. Default = 1.
- `double = obj.GetN1 ()` - Set/Get the "squareness" parameter in the z axis. Default = 1.
- `obj.SetN2 (double )` - Set/Get the "squareness" parameter in the x-y plane. Default = 1.
- `double = obj.GetN2 ()` - Set/Get the "squareness" parameter in the x-y plane. Default = 1.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - A superellipsoid.

  This function performs the mapping $f(u, v) \rightarrow (x, y, x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x, y, z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.107 vtkParametricSuperToroid

30.107.1 Usage

vtkParametricSuperToroid generates a supertoroid. Essentially a supertoroid is a torus with the sine and cosine terms raised to a power. A supertoroid is a versatile primitive that is controlled by four parameters r0, r1, n1 and n2. r0, r1 determine the type of torus whilst the value of n1 determines the shape of the torus ring and n2 determines the shape of the cross section of the ring. It is the different values of these powers which give rise to a family of 3D shapes that are all basically toroidal in shape.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

Also see: http://astronomy.swin.edu.au/ pbourke/surfaces/.

To create an instance of class vtkParametricSuperToroid, simply invoke its constructor as follows

```
obj = vtkParametricSuperToroid
```

30.107.2 Methods

The class vtkParametricSuperToroid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricSuperToroid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricSuperToroid = obj.NewInstance ()`
- `vtkParametricSuperToroid = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Set/Get the radius from the center to the middle of the ring of the supertoroid. Default = 1.
- `obj.SetRingRadius (double )` - Set/Get the radius from the center to the middle of the ring of the supertoroid. Default = 1.
- `double = obj.GetRingRadius ()` - Set/Get the radius from the center to the middle of the ring of the supertoroid. Default = 1.
- `obj.SetCrossSectionRadius (double )` - Set/Get the radius of the cross section of ring of the supertoroid. Default = 0.5.
- `double = obj.GetCrossSectionRadius ()` - Set/Get the radius of the cross section of ring of the supertoroid. Default = 0.5.
- `obj.SetXRadius (double )` - Set/Get the scaling factor for the x-axis. Default = 1.
- `double = obj.GetXRadius ()` - Set/Get the scaling factor for the x-axis. Default = 1.
- `obj.SetYRadius (double )` - Set/Get the scaling factor for the y-axis. Default = 1.
- `double = obj.GetYRadius ()` - Set/Get the scaling factor for the y-axis. Default = 1.
- `obj.SetZRadius (double )` - Set/Get the scaling factor for the z-axis. Default = 1.
- `double = obj.GetZRadius ()` - Set/Get the scaling factor for the z-axis. Default = 1.
- `obj.SetN1 (double )` - Set/Get the shape of the torus ring. Default = 1.
- `double = obj.GetN1 ()` - Set/Get the shape of the torus ring. Default = 1.
- `obj.SetN2 (double )` - Set/Get the shape of the cross section of the ring. Default = 1.
- `double = obj.GetN2 ()` - Set/Get the shape of the cross section of the ring. Default = 1.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - A supertoroid.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.108 vtkParametricTorus

30.108.1 Usage

vtkParametricTorus generates a torus.

For further information about this surface, please consult the technical description "Parametric surfaces" in http://www.vtk.org/documents.php in the "VTK Technical Documents" section in the VTk.org web pages.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricTorus, simply invoke its constructor as follows

```
obj = vtkParametricTorus
```

30.108.2 Methods

The class vtkParametricTorus has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricTorus class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricTorus = obj.NewInstance ()`
- `vtkParametricTorus = obj.SafeDownCast (vtkObject o)`
- `obj.SetRingRadius (double )` - Set/Get the radius from the center to the middle of the ring of the torus. The default value is 1.0.
- `double = obj.GetRingRadius ()` - Set/Get the radius from the center to the middle of the ring of the torus. The default value is 1.0.
- `obj.SetCrossSectionRadius (double )` - Set/Get the radius of the cross section of ring of the torus. The default value is 0.5.
- `double = obj.GetCrossSectionRadius ()` - Set/Get the radius of the cross section of ring of the torus. The default value is 0.5.
- `int = obj.GetDimension ()` - A torus.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `obj.Evaluate (double uvw[3], double Pt[3], double Duvw[9])` - A torus.

  This function performs the mapping $f(u,v) \rightarrow (x,y,x)$, returning it as Pt. It also returns the partial derivatives Du and Dv. $Pt = (x,y,z), Du = (dx/du, dy/du, dz/du), Dv = (dx/dv, dy/dv, dz/dv)$. Then the normal is $N = DuXDv$.
- `double = obj.EvaluateScalar (double uvw[3], double Pt[3], double Duvw[9])` - Calculate a user defined scalar using one or all of uvw, Pt, Duvw.

  uvw are the parameters with Pt being the the Cartesian point, Duvw are the derivatives of this point with respect to u, v and w. Pt, Duvw are obtained from Evaluate().

  This function is only called if the ScalarMode has the value vtkParametricFunctionSource::SCALAR_FUNCTION_DEFI

  If the user does not need to calculate a scalar, then the instantiated function should return zero.

30.109 vtkPerspectiveTransform

30.109.1 Usage

A vtkPerspectiveTransform can be used to describe the full range of homogeneous transformations. It was designed in particular to describe a camera-view of a scene. ¡P¿The order in which you set up the display coordinates (via AdjustZBuffer() and AdjustViewport()), the projection (via Perspective(), Frustum(), or Ortho()) and the camera view (via SetupCamera()) are important. If the transform is in PreMultiply mode, which is the default, set the Viewport and ZBuffer first, then the projection, and finally the camera view. Once the view is set up, the Translate and Rotate methods can be used to move the camera around in world coordinates. If the Oblique() or Stereo() methods are used, they should be called just before SetupCamera(). ¡P¿In PostMultiply mode, you must perform all transformations in the opposite order. This is necessary, for example, if you already have a perspective transformation set up but must adjust the viewport. Another example is if you have a view transformation, and wish to perform translations and rotations in the camera's coordinate system rather than in world coordinates. ¡P¿The SetInput and Concatenate methods can be used to create a transformation pipeline with vtkPerspectiveTransform. See vtkTransform for more information on the transformation pipeline.

To create an instance of class vtkPerspectiveTransform, simply invoke its constructor as follows

```
obj = vtkPerspectiveTransform
```

30.109.2 Methods

The class vtkPerspectiveTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPerspectiveTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPerspectiveTransform = obj.NewInstance ()`
- `vtkPerspectiveTransform = obj.SafeDownCast (vtkObject o)`
- `obj.Identity ()` - Set this transformation to the identity transformation. If the transform has an Input, then the transformation will be reset so that it is the same as the Input.
- `obj.Inverse ()` - Invert the transformation. This will also set a flag so that the transformation will use the inverse of its Input, if an Input has been set.
- `obj.AdjustViewport (double oldXMin, double oldXMax, double oldYMin, double oldYMax, double newXMin,` - Perform an adjustment to the viewport coordinates. By default Ortho, Frustum, and Perspective provide a window of ([-1,+1],[-1,+1]). In PreMultiply mode, you call this method before calling Ortho, Frustum, or Perspective. In PostMultiply mode you can call it after. Note that if you must apply both AdjustZBuffer and AdjustViewport, it makes no difference which order you apply them in.
- `obj.AdjustZBuffer (double oldNearZ, double oldFarZ, double newNearZ, double newFarZ)` - Perform an adjustment to the Z-Buffer range that the near and far clipping planes map to. By default Ortho, Frustum, and Perspective map the near clipping plane to -1 and the far clipping plane to +1. In PreMultiply mode, you call this method before calling Ortho, Frustum, or Perspective. In PostMultiply mode you can call it after.
- `obj.Ortho (double xmin, double xmax, double ymin, double ymax, double znear, double zfar)` - Create an orthogonal projection matrix and concatenate it by the current transformation. The matrix maps [xmin,xmax], [ymin,ymax], [-znear,-zfar] to [-1,+1], [-1,+1], [+1,-1].

- `obj.Frustum (double xmin, double xmax, double ymin, double ymax, double znear, double zfar)` - Create an perspective projection matrix and concatenate it by the current transformation. The matrix maps a frustum with a back plane at -zfar and a front plane at -znear with extent [xmin,xmax],[ymin,ymax] to [-1,+1], [-1,+1], [+1,-1].
- `obj.Perspective (double angle, double aspect, double znear, double zfar)` - Create a perspective projection matrix by specifying the view angle (this angle is in the y direction), the aspect ratio, and the near and far clipping range. The projection matrix is concatenated with the current transformation. This method works via Frustum.
- `obj.Shear (double dxdz, double dydz, double zplane)` - Create a shear transformation about a plane at distance z from the camera. The values dxdz (i.e. dx/dz) and dydz specify the amount of shear in the x and y directions. The 'zplane' specifies the distance from the camera to the plane at which the shear causes zero displacement. Generally you want this plane to be the focal plane. This transformation can be used in combination with Ortho to create an oblique projection. It can also be used in combination with Perspective to provide correct stereo views when the eye is at arbitrary but known positions relative to the center of a flat viewing screen.
- `obj.Stereo (double angle, double focaldistance)` - Create a stereo shear matrix and concatenate it with the current transformation. This can be applied in conjunction with either a perspective transformation (via Frustum or Projection) or an orthographic projection. You must specify the distance from the camera plane to the focal plane, and the angle between the distance vector and the eye. The angle should be negative for the left eye, and positive for the right. This method works via Oblique.
- `obj.SetupCamera (double position[3], double focalpoint[3], double viewup[3])` - Set a view transformation matrix for the camera (this matrix does not contain any perspective) and concatenate it with the current transformation.
- `obj.SetupCamera (double p0, double p1, double p2, double fp0, double fp1, double fp2, double vup0,`
- `obj.Translate (double x, double y, double z)` - Create a translation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Translate (double x[3])` - Create a translation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Translate (float x[3])` - Create a translation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.RotateWXYZ (double angle, double x, double y, double z)` - Create a rotation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is in degrees, and (x,y,z) specifies the axis that the rotation will be performed around.
- `obj.RotateWXYZ (double angle, double axis[3])` - Create a rotation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is in degrees, and (x,y,z) specifies the axis that the rotation will be performed around.
- `obj.RotateWXYZ (double angle, float axis[3])` - Create a rotation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is in degrees, and (x,y,z) specifies the axis that the rotation will be performed around.
- `obj.RotateX (double angle)` - Create a rotation matrix about the X, Y, or Z axis and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is expressed in degrees.
- `obj.RotateY (double angle)` - Create a rotation matrix about the X, Y, or Z axis and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is expressed in degrees.

- `obj.RotateZ (double angle)` - Create a rotation matrix about the X, Y, or Z axis and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is expressed in degrees.
- `obj.Scale (double x, double y, double z)` - Create a scale matrix (i.e. set the diagonal elements to x, y, z) and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Scale (double s[3])` - Create a scale matrix (i.e. set the diagonal elements to x, y, z) and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Scale (float s[3])` - Create a scale matrix (i.e. set the diagonal elements to x, y, z) and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.SetMatrix (vtkMatrix4x4 matrix)` - Set the current matrix directly. This actually calls Identity(), followed by Concatenate(matrix).
- `obj.SetMatrix (double elements[16])` - Set the current matrix directly. This actually calls Identity(), followed by Concatenate(matrix).
- `obj.Concatenate (vtkMatrix4x4 matrix)` - Concatenates the matrix with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Concatenate (double elements[16])` - Concatenates the matrix with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Concatenate (vtkHomogeneousTransform transform)` - Concatenate the specified transform with the current transformation according to PreMultiply or PostMultiply semantics. The concatenation is pipelined, meaning that if any of the transformations are changed, even after Concatenate() is called, those changes will be reflected when you call TransformPoint().
- `obj.PreMultiply ()` - Sets the internal state of the transform to PreMultiply. All subsequent operations will occur before those already represented in the current transformation. In homogeneous matrix notation, M = M*A where M is the current transformation matrix and A is the applied matrix. The default is PreMultiply.
- `obj.PostMultiply ()` - Sets the internal state of the transform to PostMultiply. All subsequent operations will occur after those already represented in the current transformation. In homogeneous matrix notation, M = A*M where M is the current transformation matrix and A is the applied matrix. The default is PreMultiply.
- `int = obj.GetNumberOfConcatenatedTransforms ()` - Get the total number of transformations that are linked into this one via Concatenate() operations or via SetInput().
- `vtkHomogeneousTransform = obj.GetConcatenatedTransform (int i)` - Set the input for this transformation. This will be used as the base transformation if it is set. This method allows you to build a transform pipeline: if the input is modified, then this transformation will automatically update accordingly. Note that the InverseFlag, controlled via Inverse(), determines whether this transformation will use the Input or the inverse of the Input.
- `obj.SetInput (vtkHomogeneousTransform input)` - Set the input for this transformation. This will be used as the base transformation if it is set. This method allows you to build a transform pipeline: if the input is modified, then this transformation will automatically update accordingly. Note that the InverseFlag, controlled via Inverse(), determines whether this transformation will use the Input or the inverse of the Input.
- `vtkHomogeneousTransform = obj.GetInput ()` - Set the input for this transformation. This will be used as the base transformation if it is set. This method allows you to build a transform pipeline: if the input is modified, then this transformation will automatically update accordingly. Note that the

InverseFlag, controlled via Inverse(), determines whether this transformation will use the Input or the inverse of the Input.

- `int = obj.GetInverseFlag ()` - Get the inverse flag of the transformation. This controls whether it is the Input or the inverse of the Input that is used as the base transformation. The InverseFlag is flipped every time Inverse() is called. The InverseFlag is off when a transform is first created.
- `obj.Push ()` - Pushes the current transformation onto the transformation stack.
- `obj.Pop ()` - Deletes the transformation on the top of the stack and sets the top to the next transformation on the stack.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make a new transform of the same type – you are responsible for deleting the transform when you are done with it.
- `int = obj.CircuitCheck (vtkAbstractTransform transform)` - Check for self-reference. Will return true if concatenating with the specified transform, setting it to be our inverse, or setting it to be our input will create a circular reference. CircuitCheck is automatically called by SetInput(), SetInverse(), and Concatenate(vtkXTransform *). Avoid using this function, it is experimental.
- `long = obj.GetMTime ()` - Override GetMTime to account for input and concatenation.

30.110 vtkPlane

30.110.1 Usage

vtkPlane provides methods for various plane computations. These include projecting points onto a plane, evaluating the plane equation, and returning plane normal. vtkPlane is a concrete implementation of the abstract class vtkImplicitFunction.

To create an instance of class vtkPlane, simply invoke its constructor as follows

```
obj = vtkPlane
```

30.110.2 Methods

The class vtkPlane has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPlane class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPlane = obj.NewInstance ()`
- `vtkPlane = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double g[3])`
- `obj.SetNormal (double , double , double )` - Set/get plane normal. Plane is defined by point and normal.
- `obj.SetNormal (double  a[3])` - Set/get plane normal. Plane is defined by point and normal.
- `double = obj. GetNormal ()` - Set/get plane normal. Plane is defined by point and normal.

- `obj.SetOrigin (double , double , double )` - Set/get point through which plane passes. Plane is defined by point and normal.
- `obj.SetOrigin (double a[3])` - Set/get point through which plane passes. Plane is defined by point and normal.
- `double = obj. GetOrigin ()` - Set/get point through which plane passes. Plane is defined by point and normal.
- `obj.Push (double distance)` - Translate the plane in the direction of the normal by the distance specified. Negative values move the plane in the opposite direction.
- `obj.ProjectPoint (double x[3], double xproj[3])`
- `obj.GeneralizedProjectPoint (double x[3], double xproj[3])`
- `double = obj.DistanceToPlane (double x[3])` - Return the distance of a point x to a plane defined by n(x-p0) = 0. The normal n[3] must be magnitude=1.

30.111 vtkPlaneCollection

30.111.1 Usage

vtkPlaneCollection is an object that creates and manipulates lists of objects of type vtkPlane.

To create an instance of class vtkPlaneCollection, simply invoke its constructor as follows

```
obj = vtkPlaneCollection
```

30.111.2 Methods

The class vtkPlaneCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPlaneCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPlaneCollection = obj.NewInstance ()`
- `vtkPlaneCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkPlane )` - Add a plane to the list.
- `vtkPlane = obj.GetNextItem ()` - Get the next plane in the list.
- `vtkPlane = obj.GetItem (int i)` - Get the ith plane in the list.

30.112 vtkPlanes

30.112.1 Usage

vtkPlanes computes the implicit function and function gradient for a set of planes. The planes must define a convex space.

The function value is the closest first order distance of a point to the convex region defined by the planes. The function gradient is the plane normal at the function value. Note that the normals must point outside of the convex region. Thus, a negative function value means that a point is inside the convex region.

There are several methods to define the set of planes. The most general is to supply an instance of vtkPoints and an instance of vtkDataArray. (The points define a point on the plane, and the normals corresponding plane normals.) Two other specialized ways are to 1) supply six planes defining the view frustrum of a camera, and 2) provide a bounding box.

To create an instance of class vtkPlanes, simply invoke its constructor as follows

```
obj = vtkPlanes
```

30.112.2 Methods

The class vtkPlanes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPlanes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPlanes = obj.NewInstance ()`
- `vtkPlanes = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double n[3])`
- `obj.SetPoints (vtkPoints )` - Specify a list of points defining points through which the planes pass.
- `vtkPoints = obj.GetPoints ()` - Specify a list of points defining points through which the planes pass.
- `obj.SetNormals (vtkDataArray normals)` - Specify a list of normal vectors for the planes. There is a one-to-one correspondence between plane points and plane normals.
- `vtkDataArray = obj.GetNormals ()` - Specify a list of normal vectors for the planes. There is a one-to-one correspondence between plane points and plane normals.
- `obj.SetFrustumPlanes (double planes[24])` - An alternative method to specify six planes defined by the camera view frustrum. See vtkCamera::GetFrustumPlanes() documentation.
- `obj.SetBounds (double bounds[6])` - An alternative method to specify six planes defined by a bounding box. The bounding box is a six-vector defined as (xmin,xmax,ymin,ymax,zmin,zmax). It defines six planes orthogonal to the x-y-z coordinate axes.
- `obj.SetBounds (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - An alternative method to specify six planes defined by a bounding box. The bounding box is a six-vector defined as (xmin,xmax,ymin,ymax,zmin,zmax). It defines six planes orthogonal to the x-y-z coordinate axes.
- `int = obj.GetNumberOfPlanes ()` - Return the number of planes in the set of planes.
- `vtkPlane = obj.GetPlane (int i)` - Create and return a pointer to a vtkPlane object at the ith position. Asking for a plane outside the allowable range returns NULL. This method always returns the same object. Use GetPlane(int i, vtkPlane *plane) instead
- `obj.GetPlane (int i, vtkPlane plane)` - Create and return a pointer to a vtkPlane object at the ith position. Asking for a plane outside the allowable range returns NULL. This method always returns the same object. Use GetPlane(int i, vtkPlane *plane) instead

30.113 vtkPoints

30.113.1 Usage

vtkPoints represents 3D points. The data model for vtkPoints is an array of vx-vy-vz triplets accessible by (point or cell) id.

To create an instance of class vtkPoints, simply invoke its constructor as follows

```
obj = vtkPoints
```

30.113.2 Methods

The class vtkPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPoints = obj.NewInstance ()`
- `vtkPoints = obj.SafeDownCast (vtkObject o)`
- `int = obj.Allocate (vtkIdType sz, vtkIdType ext)` - Allocate initial memory size.
- `obj.Initialize ()` - Return object to instantiated state.
- `obj.SetData (vtkDataArray )` - Set/Get the underlying data array. This function must be implemented in a concrete subclass to check for consistency. (The tuple size must match the type of data. For example, 3-tuple data array can be assigned to a vector, normal, or points object, but not a tensor object, which has a tuple dimension of 9. Scalars, on the other hand, can have tuple dimension from 1-4, depending on the type of scalar.)
- `vtkDataArray = obj.GetData ()` - Set/Get the underlying data array. This function must be implemented in a concrete subclass to check for consistency. (The tuple size must match the type of data. For example, 3-tuple data array can be assigned to a vector, normal, or points object, but not a tensor object, which has a tuple dimension of 9. Scalars, on the other hand, can have tuple dimension from 1-4, depending on the type of scalar.)
- `int = obj.GetDataType ()` - Return the underlying data type. An integer indicating data type is returned as specified in vtkSetGet.h.
- `obj.SetDataType (int dataType)` - Specify the underlying data type of the object.
- `obj.SetDataTypeToBit ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToChar ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToUnsignedChar ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToShort ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToUnsignedShort ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToInt ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToUnsignedInt ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToLong ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToUnsignedLong ()` - Specify the underlying data type of the object.

- `obj.SetDataTypeToFloat ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToDouble ()` - Specify the underlying data type of the object.
- `obj.Squeeze ()` - Reclaim any extra memory.
- `obj.Reset ()` - Make object look empty but do not delete memory.
- `obj.DeepCopy (vtkPoints ad)` - Different ways to copy data. Shallow copy does reference count (i.e., assigns pointers and updates reference count); deep copy runs through entire data array assigning values.
- `obj.ShallowCopy (vtkPoints ad)` - Different ways to copy data. Shallow copy does reference count (i.e., assigns pointers and updates reference count); deep copy runs through entire data array assigning values.
- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this attribute data. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.
- `vtkIdType = obj.GetNumberOfPoints ()` - Return number of points in array.
- `double = obj.GetPoint (vtkIdType id)` - Return a pointer to a double point x[3] for a specific id. WARNING: Just don't use this error-prone method, the returned pointer and its values are only valid as long as another method invocation is not performed. Prefer GetPoint() with the return value in argument.
- `obj.GetPoint (vtkIdType id, double x[3])` - Copy point components into user provided array v[3] for specified id.
- `obj.SetPoint (vtkIdType id, float x[3])` - Insert point into object. No range checking performed (fast!). Make sure you use SetNumberOfPoints() to allocate memory prior to using SetPoint().
- `obj.SetPoint (vtkIdType id, double x[3])` - Insert point into object. No range checking performed (fast!). Make sure you use SetNumberOfPoints() to allocate memory prior to using SetPoint().
- `obj.SetPoint (vtkIdType id, double x, double y, double z)` - Insert point into object. No range checking performed (fast!). Make sure you use SetNumberOfPoints() to allocate memory prior to using SetPoint().
- `obj.InsertPoint (vtkIdType id, float x[3])` - Insert point into object. Range checking performed and memory allocated as necessary.
- `obj.InsertPoint (vtkIdType id, double x[3])` - Insert point into object. Range checking performed and memory allocated as necessary.
- `obj.InsertPoint (vtkIdType id, double x, double y, double z)` - Insert point into object. Range checking performed and memory allocated as necessary.
- `vtkIdType = obj.InsertNextPoint (float x[3])` - Insert point into next available slot. Returns id of slot.
- `vtkIdType = obj.InsertNextPoint (double x[3])` - Insert point into next available slot. Returns id of slot.
- `vtkIdType = obj.InsertNextPoint (double x, double y, double z)` - Insert point into next available slot. Returns id of slot.

- `obj.SetNumberOfPoints (vtkIdType number)` - Specify the number of points for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetPoint() method for fast insertion.
- `obj.GetPoints (vtkIdList ptId, vtkPoints fp)` - Given a list of pt ids, return an array of points.
- `obj.ComputeBounds ()` - Determine (xmin,xmax, ymin,ymax, zmin,zmax) bounds of points.
- `double = obj.GetBounds ()` - Return the bounds of the points.
- `obj.GetBounds (double bounds[6])` - Return the bounds of the points.

30.114 vtkPoints2D

30.114.1 Usage

vtkPoints2D represents 2D points. The data model for vtkPoints2D is an array of vx-vy doublets accessible by (point or cell) id.

To create an instance of class vtkPoints2D, simply invoke its constructor as follows

```
obj = vtkPoints2D
```

30.114.2 Methods

The class vtkPoints2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPoints2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPoints2D = obj.NewInstance ()`
- `vtkPoints2D = obj.SafeDownCast (vtkObject o)`
- `int = obj.Allocate (vtkIdType sz, vtkIdType ext)` - Allocate initial memory size.
- `obj.Initialize ()` - Return object to instantiated state.
- `obj.SetData (vtkDataArray )` - Set/Get the underlying data array. This function must be implemented in a concrete subclass to check for consistency. (The tuple size must match the type of data. For example, 3-tuple data array can be assigned to a vector, normal, or points object, but not a tensor object, which has a tuple dimension of 9. Scalars, on the other hand, can have tuple dimension from 1-4, depending on the type of scalar.)
- `vtkDataArray = obj.GetData ()` - Return the underlying data type. An integer indicating data type is returned as specified in vtkSetGet.h.
- `int = obj.GetDataType ()` - Return the underlying data type. An integer indicating data type is returned as specified in vtkSetGet.h.
- `obj.SetDataType (int dataType)` - Specify the underlying data type of the object.
- `obj.SetDataTypeToBit ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToChar ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToUnsignedChar ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToShort ()` - Specify the underlying data type of the object.

- `obj.SetDataTypeToUnsignedShort ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToInt ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToUnsignedInt ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToLong ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToUnsignedLong ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToFloat ()` - Specify the underlying data type of the object.
- `obj.SetDataTypeToDouble ()` - Return a void pointer. For image pipeline interface and other special pointer manipulation.
- `obj.Squeeze ()` - Reclaim any extra memory.
- `obj.Reset ()` - Make object look empty but do not delete memory.
- `obj.DeepCopy (vtkPoints2D ad)` - Different ways to copy data. Shallow copy does reference count (i.e., assigns pointers and updates reference count); deep copy runs through entire data array assigning values.
- `obj.ShallowCopy (vtkPoints2D ad)` - Different ways to copy data. Shallow copy does reference count (i.e., assigns pointers and updates reference count); deep copy runs through entire data array assigning values.
- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this attribute data. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.
- `vtkIdType = obj.GetNumberOfPoints ()` - Return a pointer to a double point x[2] for a specific id. WARNING: Just don't use this error-prone method, the returned pointer and its values are only valid as long as another method invocation is not performed. Prefer GetPoint() with the return value in argument.
- `obj.GetPoint (vtkIdType id, double x[2])` - Insert point into object. No range checking performed (fast!). Make sure you use SetNumberOfPoints() to allocate memory prior to using SetPoint().
- `obj.SetPoint (vtkIdType id, float x[2])` - Insert point into object. No range checking performed (fast!). Make sure you use SetNumberOfPoints() to allocate memory prior to using SetPoint().
- `obj.SetPoint (vtkIdType id, double x[2])` - Insert point into object. No range checking performed (fast!). Make sure you use SetNumberOfPoints() to allocate memory prior to using SetPoint().
- `obj.SetPoint (vtkIdType id, double x, double y)` - Insert point into object. No range checking performed (fast!). Make sure you use SetNumberOfPoints() to allocate memory prior to using SetPoint().
- `obj.InsertPoint (vtkIdType id, float x[2])` - Insert point into object. Range checking performed and memory allocated as necessary.
- `obj.InsertPoint (vtkIdType id, double x[2])` - Insert point into object. Range checking performed and memory allocated as necessary.
- `obj.InsertPoint (vtkIdType id, double x, double y)` - Insert point into object. Range checking performed and memory allocated as necessary.
- `vtkIdType = obj.InsertNextPoint (float x[2])` - Insert point into next available slot. Returns id of slot.

- `vtkIdType = obj.InsertNextPoint (double x[2])` - Insert point into next available slot. Returns id of slot.
- `vtkIdType = obj.InsertNextPoint (double x, double y)` - Insert point into next available slot. Returns id of slot.
- `obj.SetNumberOfPoints (vtkIdType number)` - Specify the number of points for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetPoint() method for fast insertion.
- `obj.GetPoints (vtkIdList ptId, vtkPoints2D fp)` - Given a list of pt ids, return an array of points.
- `obj.ComputeBounds ()` - Determine (xmin,xmax, ymin,ymax) bounds of points.
- `obj.GetBounds (double bounds[4])` - Return the bounds of the points.

30.115 vtkPolynomialSolversUnivariate

30.115.1 Usage

vtkPolynomialSolversUnivariate provides solvers for univariate polynomial equations with real coefficients. The Tartaglia-Cardan and Ferrari solvers work on polynomials of fixed degree 3 and 4, respectively. The Lin-Bairstow and Sturm solvers work on polynomials of arbitrary degree. The Sturm solver is the most robust solver but only reports roots within an interval and does not report multiplicities. The Lin-Bairstow solver reports multiplicities.

For difficult polynomials, you may wish to use FilterRoots to eliminate some of the roots reported by the Sturm solver. FilterRoots evaluates the derivatives near each root to eliminate cases where a local minimum or maximum is close to zero.

.SECTION Thanks Thanks to Philippe Pebay, Korben Rusek, David Thompson, and Maurice Rojas for implementing these solvers.

To create an instance of class vtkPolynomialSolversUnivariate, simply invoke its constructor as follows

```
obj = vtkPolynomialSolversUnivariate
```

30.115.2 Methods

The class vtkPolynomialSolversUnivariate has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolynomialSolversUnivariate class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolynomialSolversUnivariate = obj.NewInstance ()`
- `vtkPolynomialSolversUnivariate = obj.SafeDownCast (vtkObject o)`

30.116 vtkPriorityQueue

30.116.1 Usage

vtkPriorityQueue is a general object for creating and manipulating lists of object ids (e.g., point or cell ids). Object ids are sorted according to a user-specified priority, where entries at the top of the queue have the smallest values.

This implementation provides a feature beyond the usual ability to insert and retrieve (or pop) values from the queue. It is also possible to pop any item in the queue given its id number. This allows you to delete entries in the queue which can useful for reinserting an item into the queue.

To create an instance of class vtkPriorityQueue, simply invoke its constructor as follows

```
obj = vtkPriorityQueue
```

30.116.2 Methods

The class vtkPriorityQueue has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPriorityQueue class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPriorityQueue = obj.NewInstance ()`
- `vtkPriorityQueue = obj.SafeDownCast (vtkObject o)`
- `obj.Allocate (vtkIdType sz, vtkIdType ext)` - Allocate initial space for priority queue.
- `obj.Insert (double priority, vtkIdType id)` - Insert id with priority specified. The id is generally an index like a point id or cell id.
- `vtkIdType = obj.Pop (vtkIdType location)` - Same as above but simplified for easier wrapping into interpreted languages.
- `vtkIdType = obj.Peek (vtkIdType location)` - Peek into the queue without actually removing anything. Returns the id.
- `double = obj.DeleteId (vtkIdType id)` - Delete entry in queue with specified id. Returns priority value associated with that id; or VTK_DOUBLE_MAX if not in queue.
- `double = obj.GetPriority (vtkIdType id)` - Get the priority of an entry in the queue with specified id. Returns priority value of that id or VTK_DOUBLE_MAX if not in queue.
- `vtkIdType = obj.GetNumberOfItems ()` - Return the number of items in this queue.
- `obj.Reset ()` - Empty the queue but without releasing memory. This avoids the overhead of memory allocation/deletion.

30.117 vtkProp

30.117.1 Usage

vtkProp is an abstract superclass for any objects that can exist in a rendered scene (either 2D or 3D). Instances of vtkProp may respond to various render methods (e.g., RenderOpaqueGeometry()). vtkProp also defines the API for picking, LOD manipulation, and common instance variables that control visibility, picking, and dragging.

To create an instance of class vtkProp, simply invoke its constructor as follows

```
obj = vtkProp
```

30.117.2 Methods

The class vtkProp has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProp class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProp = obj.NewInstance ()`
- `vtkProp = obj.SafeDownCast (vtkObject o)`
- `obj.GetActors (vtkPropCollection )` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.
- `obj.GetActors2D (vtkPropCollection )` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.
- `obj.GetVolumes (vtkPropCollection )` - Set/Get visibility of this vtkProp. Initial value is true.
- `obj.SetVisibility (int )` - Set/Get visibility of this vtkProp. Initial value is true.
- `int = obj.GetVisibility ()` - Set/Get visibility of this vtkProp. Initial value is true.
- `obj.VisibilityOn ()` - Set/Get visibility of this vtkProp. Initial value is true.
- `obj.VisibilityOff ()` - Set/Get visibility of this vtkProp. Initial value is true.
- `obj.SetPickable (int )` - Set/Get the pickable instance variable. This determines if the vtkProp can be picked (typically using the mouse). Also see dragable. Initial value is true.
- `int = obj.GetPickable ()` - Set/Get the pickable instance variable. This determines if the vtkProp can be picked (typically using the mouse). Also see dragable. Initial value is true.
- `obj.PickableOn ()` - Set/Get the pickable instance variable. This determines if the vtkProp can be picked (typically using the mouse). Also see dragable. Initial value is true.
- `obj.PickableOff ()` - Set/Get the pickable instance variable. This determines if the vtkProp can be picked (typically using the mouse). Also see dragable. Initial value is true.
- `obj.Pick ()` - Method fires PickEvent if the prop is picked.
- `obj.SetDragable (int )` - Set/Get the value of the dragable instance variable. This determines if an Prop, once picked, can be dragged (translated) through space. This is typically done through an interactive mouse interface. This does not affect methods such as SetPosition, which will continue to work. It is just intended to prevent some vtkProp'ss from being dragged from within a user interface. Initial value is true.
- `int = obj.GetDragable ()` - Set/Get the value of the dragable instance variable. This determines if an Prop, once picked, can be dragged (translated) through space. This is typically done through an interactive mouse interface. This does not affect methods such as SetPosition, which will continue to work. It is just intended to prevent some vtkProp'ss from being dragged from within a user interface. Initial value is true.
- `obj.DragableOn ()` - Set/Get the value of the dragable instance variable. This determines if an Prop, once picked, can be dragged (translated) through space. This is typically done through an interactive mouse interface. This does not affect methods such as SetPosition, which will continue to work. It is just intended to prevent some vtkProp'ss from being dragged from within a user interface. Initial value is true.

- `obj.DragableOff ()` - Set/Get the value of the dragable instance variable. This determines if an Prop, once picked, can be dragged (translated) through space. This is typically done through an interactive mouse interface. This does not affect methods such as SetPosition, which will continue to work. It is just intended to prevent some vtkProp'ss from being dragged from within a user interface. Initial value is true.

- `long = obj.GetRedrawMTime ()` - In case the Visibility flag is true, tell if the bounds of this prop should be taken into account or ignored during the computation of other bounding boxes, like in vtkRenderer::ResetCamera(). Initial value is true.

- `obj.SetUseBounds (bool )` - In case the Visibility flag is true, tell if the bounds of this prop should be taken into account or ignored during the computation of other bounding boxes, like in vtkRenderer::ResetCamera(). Initial value is true.

- `bool = obj.GetUseBounds ()` - In case the Visibility flag is true, tell if the bounds of this prop should be taken into account or ignored during the computation of other bounding boxes, like in vtkRenderer::ResetCamera(). Initial value is true.

- `obj.UseBoundsOn ()` - In case the Visibility flag is true, tell if the bounds of this prop should be taken into account or ignored during the computation of other bounding boxes, like in vtkRenderer::ResetCamera(). Initial value is true.

- `obj.UseBoundsOff ()` - In case the Visibility flag is true, tell if the bounds of this prop should be taken into account or ignored during the computation of other bounding boxes, like in vtkRenderer::ResetCamera(). Initial value is true.

- `double = obj.GetBounds ()` - Shallow copy of this vtkProp.

- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this vtkProp.

- `obj.InitPathTraversal ()` - vtkProp and its subclasses can be picked by subclasses of vtkAbstractPicker (e.g., vtkPropPicker). The following methods interface with the picking classes and return "pick paths". A pick path is a hierarchical, ordered list of props that form an assembly. Most often, when a vtkProp is picked, its path consists of a single node (i.e., the prop). However, classes like vtkAssembly and vtkPropAssembly can return more than one path, each path being several layers deep. (See vtkAssemblyPath for more information.) To use these methods - first invoke InitPathTraversal() followed by repeated calls to GetNextPath(). GetNextPath() returns a NULL pointer when the list is exhausted.

- `vtkAssemblyPath = obj.GetNextPath ()` - vtkProp and its subclasses can be picked by subclasses of vtkAbstractPicker (e.g., vtkPropPicker). The following methods interface with the picking classes and return "pick paths". A pick path is a hierarchical, ordered list of props that form an assembly. Most often, when a vtkProp is picked, its path consists of a single node (i.e., the prop). However, classes like vtkAssembly and vtkPropAssembly can return more than one path, each path being several layers deep. (See vtkAssemblyPath for more information.) To use these methods - first invoke InitPathTraversal() followed by repeated calls to GetNextPath(). GetNextPath() returns a NULL pointer when the list is exhausted.

- `int = obj.GetNumberOfPaths ()` - These methods are used by subclasses to place a matrix (if any) in the prop prior to rendering. Generally used only for picking. See vtkProp3D for more information.

- `obj.PokeMatrix (vtkMatrix4x4 )` - These methods are used by subclasses to place a matrix (if any) in the prop prior to rendering. Generally used only for picking. See vtkProp3D for more information.

- `vtkMatrix4x4 = obj.GetMatrix ()` - Set/Get property keys. Property keys can be digest by some rendering passes. For instance, the user may mark a prop as a shadow caster for a shadow mapping render pass. Keys are documented in render pass classes. Initial value is NULL.

- `vtkInformation = obj.GetPropertyKeys ()` - Set/Get property keys. Property keys can be digest by some rendering passes. For instance, the user may mark a prop as a shadow caster for a shadow mapping render pass. Keys are documented in render pass classes. Initial value is NULL.

- `obj.SetPropertyKeys (vtkInformation keys)` - Set/Get property keys. Property keys can be digest by some rendering passes. For instance, the user may mark a prop as a shadow caster for a shadow mapping render pass. Keys are documented in render pass classes. Initial value is NULL.

- `bool = obj.HasKeys (vtkInformation requiredKeys)` - Tells if the prop has all the required keys.

30.118 vtkPropCollection

30.118.1 Usage

vtkPropCollection represents and provides methods to manipulate a list of Props (i.e., vtkProp and subclasses). The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkPropCollection, simply invoke its constructor as follows

```
obj = vtkPropCollection
```

30.118.2 Methods

The class vtkPropCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPropCollection class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkPropCollection = obj.NewInstance ()`

- `vtkPropCollection = obj.SafeDownCast (vtkObject o)`

- `obj.AddItem (vtkProp a)` - Add an Prop to the list.

- `vtkProp = obj.GetNextProp ()` - Get the next Prop in the list.

- `vtkProp = obj.GetLastProp ()` - Get the last Prop in the list.

- `int = obj.GetNumberOfPaths ()` - Get the number of paths contained in this list. (Recall that a vtkProp can consist of multiple parts.) Used in picking and other activities to get the parts of composite entities like vtkAssembly or vtkPropAssembly.

30.119 vtkProperty2D

30.119.1 Usage

vtkProperty2D contains properties used to render two dimensional images and annotations.

To create an instance of class vtkProperty2D, simply invoke its constructor as follows

```
obj = vtkProperty2D
```

30.119.2 Methods

The class vtkProperty2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProperty2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProperty2D = obj.NewInstance ()`
- `vtkProperty2D = obj.SafeDownCast (vtkObject o)`
- `obj.DeepCopy (vtkProperty2D p)` - Assign one property to another.
- `obj.SetColor (double , double , double )` - Set/Get the RGB color of this property.
- `obj.SetColor (double  a[3])` - Set/Get the RGB color of this property.
- `double = obj. GetColor ()` - Set/Get the RGB color of this property.
- `double = obj.GetOpacity ()` - Set/Get the Opacity of this property.
- `obj.SetOpacity (double )` - Set/Get the Opacity of this property.
- `obj.SetPointSize (float )` - Set/Get the diameter of a Point. The size is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetPointSizeMinValue ()` - Set/Get the diameter of a Point. The size is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetPointSizeMaxValue ()` - Set/Get the diameter of a Point. The size is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetPointSize ()` - Set/Get the diameter of a Point. The size is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `obj.SetLineWidth (float )` - Set/Get the width of a Line. The width is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetLineWidthMinValue ()` - Set/Get the width of a Line. The width is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetLineWidthMaxValue ()` - Set/Get the width of a Line. The width is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetLineWidth ()` - Set/Get the width of a Line. The width is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `obj.SetLineStipplePattern (int )` - Set/Get the stippling pattern of a Line, as a 16-bit binary pattern (1 = pixel on, 0 = pixel off). This is only implemented for OpenGL. The default is 0xFFFF.
- `int = obj.GetLineStipplePattern ()` - Set/Get the stippling pattern of a Line, as a 16-bit binary pattern (1 = pixel on, 0 = pixel off). This is only implemented for OpenGL. The default is 0xFFFF.
- `obj.SetLineStippleRepeatFactor (int )` - Set/Get the stippling repeat factor of a Line, which specifies how many times each bit in the pattern is to be repeated. This is only implemented for OpenGL. The default is 1.

- `int = obj.GetLineStippleRepeatFactorMinValue ()` - Set/Get the stippling repeat factor of a Line, which specifies how many times each bit in the pattern is to be repeated. This is only implemented for OpenGL. The default is 1.
- `int = obj.GetLineStippleRepeatFactorMaxValue ()` - Set/Get the stippling repeat factor of a Line, which specifies how many times each bit in the pattern is to be repeated. This is only implemented for OpenGL. The default is 1.
- `int = obj.GetLineStippleRepeatFactor ()` - Set/Get the stippling repeat factor of a Line, which specifies how many times each bit in the pattern is to be repeated. This is only implemented for OpenGL. The default is 1.
- `obj.SetDisplayLocation (int )` - The DisplayLocation is either background or foreground. If it is background, then this 2D actor will be drawn behind all 3D props or foreground 2D actors. If it is background, then this 2D actor will be drawn in front of all 3D props and background 2D actors. Within 2D actors of the same DisplayLocation type, order is determined by the order in which the 2D actors were added to the viewport.
- `int = obj.GetDisplayLocationMinValue ()` - The DisplayLocation is either background or foreground. If it is background, then this 2D actor will be drawn behind all 3D props or foreground 2D actors. If it is background, then this 2D actor will be drawn in front of all 3D props and background 2D actors. Within 2D actors of the same DisplayLocation type, order is determined by the order in which the 2D actors were added to the viewport.
- `int = obj.GetDisplayLocationMaxValue ()` - The DisplayLocation is either background or foreground. If it is background, then this 2D actor will be drawn behind all 3D props or foreground 2D actors. If it is background, then this 2D actor will be drawn in front of all 3D props and background 2D actors. Within 2D actors of the same DisplayLocation type, order is determined by the order in which the 2D actors were added to the viewport.
- `int = obj.GetDisplayLocation ()` - The DisplayLocation is either background or foreground. If it is background, then this 2D actor will be drawn behind all 3D props or foreground 2D actors. If it is background, then this 2D actor will be drawn in front of all 3D props and background 2D actors. Within 2D actors of the same DisplayLocation type, order is determined by the order in which the 2D actors were added to the viewport.
- `obj.SetDisplayLocationToBackground ()` - The DisplayLocation is either background or foreground. If it is background, then this 2D actor will be drawn behind all 3D props or foreground 2D actors. If it is background, then this 2D actor will be drawn in front of all 3D props and background 2D actors. Within 2D actors of the same DisplayLocation type, order is determined by the order in which the 2D actors were added to the viewport.
- `obj.SetDisplayLocationToForeground ()` - The DisplayLocation is either background or foreground. If it is background, then this 2D actor will be drawn behind all 3D props or foreground 2D actors. If it is background, then this 2D actor will be drawn in front of all 3D props and background 2D actors. Within 2D actors of the same DisplayLocation type, order is determined by the order in which the 2D actors were added to the viewport.

30.120 vtkQuadratureSchemeDefinition

30.120.1 Usage

An Elemental data type that holds a definition of a numerical quadrature scheme. The definition contains the requisite information to interpolate to the so called quadrature points of the specific scheme. namely:

```
1)
A matrix of shape function weights(shape functions evaluated
```

```
at parametric coordinates of the quadrature points).

2)
The number of quadrature points and cell nodes. These parameters
size the matrix, and allow for convinent evaluation by users
of the definition.
```

To create an instance of class vtkQuadratureSchemeDefinition, simply invoke its constructor as follows

```
obj = vtkQuadratureSchemeDefinition
```

30.120.2 Methods

The class vtkQuadratureSchemeDefinition has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadratureSchemeDefinition class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadratureSchemeDefinition = obj.NewInstance ()`
- `vtkQuadratureSchemeDefinition = obj.SafeDownCast (vtkObject o)`
- `int = obj.DeepCopy (vtkQuadratureSchemeDefinition other)` - Deep copy.
- `int = obj.SaveState (vtkXMLDataElement e)` - Put the object into an XML representation. The element passed in is assumed to be empty.
- `int = obj.RestoreState (vtkXMLDataElement e)` - Restore the object from an XML representation.
- `obj.Clear ()` - Release all allocated resources and set the object to an unitialized state.
- `obj.Initialize (int cellType, int numberOfNodes, int numberOfQuadraturePoints, double shapeFunctior` - Initialize the object allocating resources as needed.
- `obj.Initialize (int cellType, int numberOfNodes, int numberOfQuadraturePoints, double shapeFunctior` - Initialize the object allocating resources as needed.
- `int = obj.GetCellType () const` - Access to an alternative key.
- `int = obj.GetQuadratureKey () const` - Get the number of nodes associated with the interpolation.
- `int = obj.GetNumberOfNodes () const` - Get the number of quadrature points associated with the scheme.
- `int = obj.GetNumberOfQuadraturePoints () const` - Get the array of shape function weights. Shape function weights are the shape functions evaluated at the quadrature points. There are "NumberOfNodes" weights for each quadrature point.

30.121 vtkQuadric

30.121.1 Usage

vtkQuadric evaluates the quadric function F(x,y,z) = a0*x$\hat{2}$ + a1*y$\hat{2}$ + a2*z$\hat{2}$ + a3*x*y + a4*y*z + a5*x*z + a6*x + a7*y + a8*z + a9. vtkQuadric is a concrete implementation of vtkImplicitFunction.

To create an instance of class vtkQuadric, simply invoke its constructor as follows

```
obj = vtkQuadric
```

30.121.2 Methods

The class vtkQuadric has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadric class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadric = obj.NewInstance ()`
- `vtkQuadric = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double g[3])`
- `obj.SetCoefficients (double a[10])`
- `obj.SetCoefficients (double a0, double a1, double a2, double a3, double a4, double a5, double a6, d`
- `double = obj. GetCoefficients ()`

30.122 vtkRandomSequence

30.122.1 Usage

vtkRandomSequence defines the interface of any sequence of random numbers.

At this level of abstraction, there is no assumption about the distribution of the numbers or about the quality of the sequence of numbers to be statistically independent. There is no assumption about the range of values.

To the question about why a random "sequence" class instead of a random "generator" class or to a random "number" class?, see the OOSC book: "Object-Oriented Software Construction", 2nd Edition, by Bertrand Meyer. chapter 23, "Principles of class design", "Pseudo-random number generators: a design exercise", page 754–755.

To create an instance of class vtkRandomSequence, simply invoke its constructor as follows

```
obj = vtkRandomSequence
```

30.122.2 Methods

The class vtkRandomSequence has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRandomSequence class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRandomSequence = obj.NewInstance ()`
- `vtkRandomSequence = obj.SafeDownCast (vtkObject o)`
- `double = obj.GetValue ()` - Current value
- `obj.Next ()` - Move to the next number in the random sequence.

30.123 vtkReferenceCount

30.123.1 Usage

vtkReferenceCount functionality has now been moved into vtkObject

To create an instance of class vtkReferenceCount, simply invoke its constructor as follows

```
obj = vtkReferenceCount
```

30.123.2 Methods

The class vtkReferenceCount has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkReferenceCount class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkReferenceCount = obj.NewInstance ()`
- `vtkReferenceCount = obj.SafeDownCast (vtkObject o)`

30.124 vtkRungeKutta2

30.124.1 Usage

This is a concrete sub-class of vtkInitialValueProblemSolver. It uses a 2nd order Runge-Kutta method to obtain the values of a set of functions at the next time step.

To create an instance of class vtkRungeKutta2, simply invoke its constructor as follows

```
obj = vtkRungeKutta2
```

30.124.2 Methods

The class vtkRungeKutta2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRungeKutta2 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRungeKutta2 = obj.NewInstance ()`
- `vtkRungeKutta2 = obj.SafeDownCast (vtkObject o)`

30.125 vtkRungeKutta4

30.125.1 Usage

This is a concrete sub-class of vtkInitialValueProblemSolver. It uses a 4th order Runge-Kutta method to obtain the values of a set of functions at the next time step.

To create an instance of class vtkRungeKutta4, simply invoke its constructor as follows

```
obj = vtkRungeKutta4
```

30.125.2 Methods

The class vtkRungeKutta4 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRungeKutta4 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRungeKutta4 = obj.NewInstance ()`
- `vtkRungeKutta4 = obj.SafeDownCast (vtkObject o)`

30.126 vtkRungeKutta45

30.126.1 Usage

This is a concrete sub-class of vtkInitialValueProblemSolver. It uses a 5th order Runge-Kutta method with stepsize control to obtain the values of a set of functions at the next time step. The stepsize is adjusted by calculating an estimated error using an embedded 4th order Runge-Kutta formula: Press, W. H. et al., 1992, Numerical Recipes in Fortran, Second Edition, Cambridge University Press Cash, J.R. and Karp, A.H. 1990, ACM Transactions on Mathematical Software, vol 16, pp 201-222

To create an instance of class vtkRungeKutta45, simply invoke its constructor as follows

```
obj = vtkRungeKutta45
```

30.126.2 Methods

The class vtkRungeKutta45 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRungeKutta45 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRungeKutta45 = obj.NewInstance ()`
- `vtkRungeKutta45 = obj.SafeDownCast (vtkObject o)`

30.127 vtkScalarsToColors

30.127.1 Usage

vtkScalarsToColors is a general purpose superclass for objects that convert scalars to colors. This include vtkLookupTable classes and color transfer functions.

The scalars to color mapping can be augmented with an additional uniform alpha blend. This is used, for example, to blend a vtkActor's opacity with the lookup table values.

To create an instance of class vtkScalarsToColors, simply invoke its constructor as follows

```
obj = vtkScalarsToColors
```

30.127.2 Methods

The class vtkScalarsToColors has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkScalarsToColors class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkScalarsToColors = obj.NewInstance ()`
- `vtkScalarsToColors = obj.SafeDownCast (vtkObject o)`
- `int = obj.IsOpaque ()` - Return true if all of the values defining the mapping have an opacity equal to 1. Default implementation return true.
- `obj.Build ()` - Perform any processing required (if any) before processing scalars.
- `double = obj.GetRange ()` - Sets/Gets the range of scalars which will be mapped.
- `obj.SetRange (double min, double max)` - Sets/Gets the range of scalars which will be mapped.
- `obj.SetRange (double rng[2])` - Map one value through the lookup table and return a color defined as a RGBA unsigned char tuple (4 bytes).
- `obj.GetColor (double v, double rgb[3])` - Map one value through the lookup table and return the color as an RGB array of doubles between 0 and 1.
- `double = obj.GetColor (double v)` - Map one value through the lookup table and return the alpha value (the opacity) as a double between 0 and 1.
- `double = obj.GetOpacity (double )` - Map one value through the lookup table and return the luminance 0.3*red + 0.59*green + 0.11*blue as a double between 0 and 1. Returns the luminance value for the specified scalar value.
- `double = obj.GetLuminance (double x)` - Specify an additional opacity (alpha) value to blend with. Values != 1 modify the resulting color consistent with the requested form of the output. This is typically used by an actor in order to blend its opacity.
- `obj.SetAlpha (double alpha)` - Specify an additional opacity (alpha) value to blend with. Values != 1 modify the resulting color consistent with the requested form of the output. This is typically used by an actor in order to blend its opacity.
- `double = obj.GetAlpha ()` - Specify an additional opacity (alpha) value to blend with. Values != 1 modify the resulting color consistent with the requested form of the output. This is typically used by an actor in order to blend its opacity.
- `vtkUnsignedCharArray = obj.MapScalars (vtkDataArray scalars, int colorMode, int component)` - An internal method maps a data array into a 4-component, unsigned char RGBA array. The color mode determines the behavior of mapping. If VTK_COLOR_MODE_DEFAULT is set, then unsigned char data arrays are treated as colors (and converted to RGBA if necessary); otherwise, the data is mapped through this instance of ScalarsToColors. The offset is used for data arrays with more than one component; it indicates which component to use to do the blending. When the component argument is -1, then the this object uses its own selected technique to change a vector into a scalar to map.
- `obj.SetVectorMode (int )` - Change mode that maps vectors by magnitude vs. component.
- `int = obj.GetVectorMode ()` - Change mode that maps vectors by magnitude vs. component.

- `obj.SetVectorModeToMagnitude ()` - Change mode that maps vectors by magnitude vs. component.
- `obj.SetVectorModeToComponent ()` - Change mode that maps vectors by magnitude vs. component.
- `obj.SetVectorComponent (int )` - If the mapper does not select which component of a vector to map to colors, you can specify it here.
- `int = obj.GetVectorComponent ()` - If the mapper does not select which component of a vector to map to colors, you can specify it here.
- `obj.MapScalarsThroughTable (vtkDataArray scalars, string output, int outputFormat)` - Map a set of scalars through the lookup table in a single operation. The output format can be set to VTK_RGBA (4 components), VTK_RGB (3 components), VTK_LUMINANCE (1 component, greyscale), or VTK_LUMINANCE_ALPHA (2 components) If not supplied, the output format defaults to RGBA.
- `obj.MapScalarsThroughTable (vtkDataArray scalars, string output)` - An internal method typically not used in applications.
- `vtkUnsignedCharArray = obj.ConvertUnsignedCharToRGBA (vtkUnsignedCharArray colors, int numComp, int` - An internal method used to convert a color array to RGBA. The method instantiates a vtkUnsignedCharArray and returns it. The user is responsible for managing the memory.
- `int = obj.UsingLogScale ()` - This should return 1 is the subclass is using log scale for mapping scalars to colors. Default implementation returns 0.

30.128 vtkServerSocket

30.128.1 Usage

To create an instance of class vtkServerSocket, simply invoke its constructor as follows

```
obj = vtkServerSocket
```

30.128.2 Methods

The class vtkServerSocket has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkServerSocket class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkServerSocket = obj.NewInstance ()`
- `vtkServerSocket = obj.SafeDownCast (vtkObject o)`
- `int = obj.CreateServer (int port)` - Creates a server socket at a given port and binds to it. Returns -1 on error. 0 on success.
- `vtkClientSocket = obj.WaitForConnection (long msec)` - Waits for a connection. When a connection is received a new vtkClientSocket object is created and returned. Returns NULL on timeout.
- `int = obj.GetServerPort ()` - Returns the port on which the server is running.

30.129 vtkShortArray

30.129.1 Usage

vtkShortArray is an array of values of type short. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkShortArray, simply invoke its constructor as follows

```
obj = vtkShortArray
```

30.129.2 Methods

The class vtkShortArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkShortArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkShortArray = obj.NewInstance ()`
- `vtkShortArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, short tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, short tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, short tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (short tuple)` - Get the data at a particular index.
- `short = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, short value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, short f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (short f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `obj.SetArray (short array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (short array, vtkIdType size, int save, int deleteMethod)`

30.130 vtkSignedCharArray

30.130.1 Usage

vtkSignedCharArray is an array of values of type signed char. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkSignedCharArray, simply invoke its constructor as follows

```
obj = vtkSignedCharArray
```

30.130.2 Methods

The class vtkSignedCharArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSignedCharArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSignedCharArray = obj.NewInstance ()`
- `vtkSignedCharArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `signed = obj.char GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, signed char value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, signed char f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (signed char f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `signed = obj.string WritePointer (vtkIdType id, vtkIdType number)` - Get the address of a particular data index. Performs no checks to verify that the memory has been allocated etc.
- `signed = obj.string GetPointer (vtkIdType id)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.

30.131 vtkSocket

30.131.1 Usage

This abstract class encapsulates a BSD socket. It provides an API for basic socket operations.

To create an instance of class vtkSocket, simply invoke its constructor as follows

```
obj = vtkSocket
```

30.131.2 Methods

The class vtkSocket has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSocket class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSocket = obj.NewInstance ()`
- `vtkSocket = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetConnected ()` - Close the socket.
- `obj.CloseSocket ()` - These methods send data over the socket. Returns 1 on success, 0 on error and raises vtkCommand::ErrorEvent.

30.132 vtkSocketCollection

30.132.1 Usage

Apart from being vtkCollection subclass for sockets, this class provides means to wait for activity on all the sockets in the collection simultaneously.

To create an instance of class vtkSocketCollection, simply invoke its constructor as follows

```
obj = vtkSocketCollection
```

30.132.2 Methods

The class vtkSocketCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSocketCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSocketCollection = obj.NewInstance ()`
- `vtkSocketCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkSocket soc)`
- `int = obj.SelectSockets (long msec)` - Select all Connected sockets in the collection. If msec is specified, it timesout after msec milliseconds on inactivity. Returns 0 on timeout, -1 on error; 1 is a socket was selected. The selected socket can be retrieved by GetLastSelectedSocket().
- `vtkSocket = obj.GetLastSelectedSocket ()` - Overridden to unset SelectedSocket.
- `obj.ReplaceItem (int i, vtkObject )` - Overridden to unset SelectedSocket.
- `obj.RemoveItem (int i)` - Overridden to unset SelectedSocket.
- `obj.RemoveItem (vtkObject )` - Overridden to unset SelectedSocket.
- `obj.RemoveAllItems ()` - Overridden to unset SelectedSocket.

30.133 vtkSphericalTransform

30.133.1 Usage

vtkSphericalTransform will convert (r,phi,theta) coordinates to (x,y,z) coordinates and back again. The angles are given in radians. By default, it converts spherical coordinates to rectangular, but GetInverse() returns a transform that will do the opposite. The equation that is used is x = r*sin(phi)*cos(theta), y = r*sin(phi)*sin(theta), z = r*cos(phi).

To create an instance of class vtkSphericalTransform, simply invoke its constructor as follows

```
obj = vtkSphericalTransform
```

30.133.2 Methods

The class vtkSphericalTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSphericalTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSphericalTransform = obj.NewInstance ()`
- `vtkSphericalTransform = obj.SafeDownCast (vtkObject o)`
- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.

30.134 vtkStringArray

30.134.1 Usage

Points and cells may sometimes have associated data that are stored as strings, e.g. many information visualization projects. This class provides a reasonably clean way to store and access those.

To create an instance of class vtkStringArray, simply invoke its constructor as follows

```
obj = vtkStringArray
```

30.134.2 Methods

The class vtkStringArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStringArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStringArray = obj.NewInstance ()`
- `vtkStringArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()`
- `int = obj.IsNumeric ()` - Release storage and reset array to initial state.
- `obj.Initialize ()` - Release storage and reset array to initial state.

- `int = obj.GetDataTypeSize ()` - Return the size of the data type. WARNING: This may not mean what you expect with strings. It will return sizeof(vtkstd::string) and not take into account the data included in any particular string.
- `obj.Squeeze ()` - Resize the array while conserving the data.
- `int = obj.Resize (vtkIdType numTuples)` - Resize the array while conserving the data.
- `obj.SetTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Set the tuple at the ith location using the jth tuple in the source array. This method assumes that the two arrays have the same type and structure. Note that range checking and memory allocation is not performed; use in conjunction with SetNumberOfTuples() to allocate space.
- `obj.InsertTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at ith location in this array. Note that memory allocation is performed as necessary to hold the data.
- `vtkIdType = obj.InsertNextTuple (vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at the end in this array. Note that memory allocation is performed as necessary to hold the data. Returns the location at which the data was inserted.
- `obj.InterpolateTuple (vtkIdType i, vtkIdList ptIndices, vtkAbstractArray source, double weights)` - Set the ith tuple in this array as the interpolated tuple value, given the ptIndices in the source array and associated interpolation weights. This method assumes that the two arrays are of the same type and strcuture.
- `obj.InterpolateTuple (vtkIdType i, vtkIdType id1, vtkAbstractArray source1, vtkIdType id2, vtkAbstr`
- `obj.GetTuples (vtkIdList ptIds, vtkAbstractArray output)` - Given a list of indices, return an array of values. You must insure that the output array has been previously allocated with enough space to hold the data and that the types match sufficiently to allow conversion (if necessary).
- `obj.GetTuples (vtkIdType p1, vtkIdType p2, vtkAbstractArray output)` - Get the values for the range of indices specified (i.e., p1-¿p2 inclusive). You must insure that the output array has been previously allocated with enough space to hold the data and that the type of the output array is compatible with the type of this array.
- `int = obj.Allocate (vtkIdType sz, vtkIdType ext)` - Allocate memory for this array. Delete old storage only if necessary. Note that ext is no longer used.
- `vtkStdString = obj.\&GetValue (vtkIdType id)` - Get the data at a particular index.
- `obj.SetValue (vtkIdType id, string value)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetNumberOfTuples (vtkIdType number)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `vtkIdType = obj.GetNumberOfValues ()`
- `int = obj.GetNumberOfElementComponents ()`
- `int = obj.GetElementComponentSize ()` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, string val)` - Insert data at a specified position in the array.

- `vtkIdType = obj.InsertNextValue (string f)` - Insert data at the end of the array. Return its location in the array.
- `obj.DeepCopy (vtkAbstractArray aa)` - Deep copy of another string array. Will complain and change nothing if the array passed in is not a vtkStringArray.
- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this data array. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.

  This function takes into account the size of the contents of the strings as well as the string containers themselves.
- `vtkArrayIterator = obj.NewIterator ()` - Returns a vtkArrayIteratorTemplate¡vtkStdString¿.
- `vtkIdType = obj.GetDataSize ()` - Returns the size of the data in DataTypeSize units. Thus, the number of bytes for the data can be computed by GetDataSize() * GetDataTypeSize(). The size computation includes the string termination character for each string.
- `vtkIdType = obj.LookupValue (string value)`
- `obj.LookupValue (string value, vtkIdList ids)`
- `obj.DataChanged ()` - Tell the array explicitly that the data has changed. This is only necessary to call when you modify the array contents without using the array's API (i.e. you retrieve a pointer to the data and modify the array contents). You need to call this so that the fast lookup will know to rebuild itself. Otherwise, the lookup functions will give incorrect results.
- `obj.DataElementChanged (vtkIdType id)` - Tell the array explicitly that a single data element has changed. Like DataChanged(), then is only necessary when you modify the array contents without using the array's API.
- `obj.ClearLookup ()` - Delete the associated fast lookup data structure on this array, if it exists. The lookup will be rebuilt on the next call to a lookup function.

30.135 vtkStructuredData

30.135.1 Usage

vtkStructuredData is an abstract class that specifies an interface for topologically regular data. Regular data is data that can be accessed in rectangular fashion using an i-j-k index. A finite difference grid, a volume, or a pixmap are all considered regular.

To create an instance of class vtkStructuredData, simply invoke its constructor as follows

```
obj = vtkStructuredData
```

30.135.2 Methods

The class vtkStructuredData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredData = obj.NewInstance ()`
- `vtkStructuredData = obj.SafeDownCast (vtkObject o)`

30.136 vtkStructuredVisibilityConstraint

30.136.1 Usage

vtkStructuredVisibilityConstraint is a general class to manage a list of points/cell marked as invalid or invisible. Currently, it does this by maintaining an unsigned char array associated with points/cells. To conserve memory, this array is allocated only when it is needed (when Blank() is called the first time). Make sure to call Initialize() with the right dimensions before calling any methods that set/get visibility.

To create an instance of class vtkStructuredVisibilityConstraint, simply invoke its constructor as follows

```
obj = vtkStructuredVisibilityConstraint
```

30.136.2 Methods

The class vtkStructuredVisibilityConstraint has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredVisibilityConstraint class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredVisibilityConstraint = obj.NewInstance ()`
- `vtkStructuredVisibilityConstraint = obj.SafeDownCast (vtkObject o)`
- `char = obj.IsVisible (vtkIdType id)` - Returns 1 is the point/cell is visible, 0 otherwise.
- `obj.Blank (vtkIdType id)` - Sets the visibility flag of the given point/cell off. The first time blank is called, a new visibility array is created if it doesn't exist.
- `obj.UnBlank (vtkIdType id)` - Sets the visibility flag of the given point/cell on.
- `int = obj. GetDimensions ()` - Get the dimensions used to initialize the object.
- `obj.Initialize (int dims[3])` - Set the dimensions and set the Initialized flag to 1. Once an object is initialized, it's dimensions can not be changed anymore.
- `obj.SetVisibilityById (vtkUnsignedCharArray vis)` - Set/Get the array used to store the visibility flags.
- `vtkUnsignedCharArray = obj.GetVisibilityById ()` - Set/Get the array used to store the visibility flags.
- `obj.ShallowCopy (vtkStructuredVisibilityConstraint src)` - Copies the dimensions, the visibility array pointer and the initialized flag.
- `obj.DeepCopy (vtkStructuredVisibilityConstraint src)` - Copies the dimensions, the visibility array and the initialized flag.
- `char = obj.IsConstrained ()`

30.137 vtkTableExtentTranslator

30.137.1 Usage

vtkTableExtentTranslator provides a vtkExtentTranslator that is programmed with a specific extent corresponding to each piece number. Readers can provide this to an application to allow the pipeline to execute using the same piece breakdown that is provided in the input file.

To create an instance of class vtkTableExtentTranslator, simply invoke its constructor as follows

```
obj = vtkTableExtentTranslator
```

30.137.2 Methods

The class vtkTableExtentTranslator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableExtentTranslator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableExtentTranslator = obj.NewInstance ()`
- `vtkTableExtentTranslator = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfPieces (int pieces)`
- `obj.SetNumberOfPiecesInTable (int pieces)` - Set the real number of pieces in the extent table.
- `int = obj.GetNumberOfPiecesInTable ()` - Set the real number of pieces in the extent table.
- `int = obj.PieceToExtent ()` - Called to translate the current piece into an extent. This is not thread safe.
- `int = obj.PieceToExtentByPoints ()` - Not supported by this subclass of vtkExtentTranslator.
- `int = obj.PieceToExtentThreadSafe (int piece, int numPieces, int ghostLevel, int wholeExtent, int r` - Not supported by this subclass of vtkExtentTranslator.
- `obj.SetExtentForPiece (int piece, int extent)` - Set the extent to be used for a piece. This sets the extent table entry for the piece.
- `obj.GetExtentForPiece (int piece, int extent)` - Get the extent table entry for the given piece. This is only for code that is setting up the table. Extent translation should always be done through the PieceToExtent method.
- `obj.SetMaximumGhostLevel (int )` - Set the maximum ghost level that can be requested. This can be used by a reader to make sure an extent request does not go outside the boundaries of the piece's file.
- `int = obj.GetMaximumGhostLevel ()` - Set the maximum ghost level that can be requested. This can be used by a reader to make sure an extent request does not go outside the boundaries of the piece's file.
- `obj.SetPieceAvailable (int piece, int available)` - Get/Set whether the given piece is available. Requesting a piece that is not available will produce errors in the pipeline.
- `int = obj.GetPieceAvailable (int piece)` - Get/Set whether the given piece is available. Requesting a piece that is not available will produce errors in the pipeline.

30.138 vtkTensor

30.138.1 Usage

vtkTensor is a floating point representation of an nxn tensor. vtkTensor provides methods for assignment and reference of tensor components. It does it in such a way as to minimize data copying.

To create an instance of class vtkTensor, simply invoke its constructor as follows

```
obj = vtkTensor
```

30.138.2 Methods

The class vtkTensor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTensor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTensor = obj.NewInstance ()`
- `vtkTensor = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Initialize tensor components to 0.0.
- `double = obj.GetComponent (int i, int j)` - Get the tensor component (i,j).
- `obj.SetComponent (int i, int j, double v)` - Set the value of the tensor component (i,j).
- `obj.AddComponent (int i, int j, double v)` - Add to the value of the tensor component at location (i,j).
- `obj.DeepCopy (vtkTensor t)` - Deep copy of one tensor to another tensor.

30.139 vtkThreadMessager

30.139.1 Usage

vtkMultithreader is a class that provides support for messaging between threads multithreaded using pthreads or Windows messaging.

To create an instance of class vtkThreadMessager, simply invoke its constructor as follows

```
obj = vtkThreadMessager
```

30.139.2 Methods

The class vtkThreadMessager has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkThreadMessager class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkThreadMessager = obj.NewInstance ()`
- `vtkThreadMessager = obj.SafeDownCast (vtkObject o)`
- `obj.WaitForMessage ()` - Wait (block, non-busy) until another thread sends a message.
- `obj.SendWakeMessage ()` - Send a message to all threads who are waiting via WaitForMessage().
- `obj.EnableWaitForReceiver ()` - pthreads only. If the wait is enabled, the thread who is to call WaitForMessage() will block until a receiver thread is ready to receive.
- `obj.DisableWaitForReceiver ()` - pthreads only. If the wait is enabled, the thread who is to call WaitForMessage() will block until a receiver thread is ready to receive.
- `obj.WaitForReceiver ()` - pthreads only. If wait is enable, this will block until one thread is ready to receive a message.
- `obj.SendMessage ()` - @deprecated Replaced by vtkThreadMessager::SendWakeMessage() as of VTK 5.0.

30.140 vtkTimePointUtility

30.140.1 Usage

vtkTimePointUtility is provides methods to perform common time operations.

To create an instance of class vtkTimePointUtility, simply invoke its constructor as follows

```
obj = vtkTimePointUtility
```

30.140.2 Methods

The class vtkTimePointUtility has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTimePointUtility class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTimePointUtility = obj.NewInstance ()`
- `vtkTimePointUtility = obj.SafeDownCast (vtkObject o)`

30.141 vtkTimerLog

30.141.1 Usage

vtkTimerLog contains walltime and cputime measurements associated with a given event. These results can be later analyzed when "dumping out" the table.

In addition, vtkTimerLog allows the user to simply get the current time, and to start/stop a simple timer separate from the timing table logging.

To create an instance of class vtkTimerLog, simply invoke its constructor as follows

```
obj = vtkTimerLog
```

30.141.2 Methods

The class vtkTimerLog has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTimerLog class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTimerLog = obj.NewInstance ()`
- `vtkTimerLog = obj.SafeDownCast (vtkObject o)`
- `obj.StartTimer ()` - Set the StartTime to the current time. Used with GetElapsedTime().
- `obj.StopTimer ()` - Sets EndTime to the current time. Used with GetElapsedTime().
- `double = obj.GetElapsedTime ()` - Returns the difference between StartTime and EndTime as a doubleing point value indicating the elapsed time in seconds.

30.142 vtkTransform

30.142.1 Usage

A vtkTransform can be used to describe the full range of linear (also known as affine) coordinate transformations in three dimensions, which are internally represented as a 4x4 homogeneous transformation matrix. When you create a new vtkTransform, it is always initialized to the identity transformation. ¡P¿The SetInput() method allows you to set another transform, instead of the identity transform, to be the base transformation. There is a pipeline mechanism to ensure that when the input is modified, the current transformation will be updated accordingly. This pipeline mechanism is also supported by the Concatenate() method. ¡P¿Most of the methods for manipulating this transformation, e.g. Translate, Rotate, and Concatenate, can operate in either PreMultiply (the default) or PostMultiply mode. In PreMultiply mode, the translation, concatenation, etc. will occur before any transformations which are represented by the current matrix. In PostMultiply mode, the additional transformation will occur after any transformations represented by the current matrix. ¡P¿This class performs all of its operations in a right handed coordinate system with right handed rotations. Some other graphics libraries use left handed coordinate systems and rotations.

To create an instance of class vtkTransform, simply invoke its constructor as follows

```
obj = vtkTransform
```

30.142.2 Methods

The class vtkTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransform = obj.NewInstance ()`
- `vtkTransform = obj.SafeDownCast (vtkObject o)`
- `obj.Identity ()` - Set the transformation to the identity transformation. If the transform has an Input, then the transformation will be reset so that it is the same as the Input.
- `obj.Inverse ()` - Invert the transformation. This will also set a flag so that the transformation will use the inverse of its Input, if an Input has been set.
- `obj.Translate (double x, double y, double z)` - Create a translation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Translate (double x[3])` - Create a translation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Translate (float x[3])` - Create a translation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.RotateWXYZ (double angle, double x, double y, double z)` - Create a rotation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is in degrees, and (x,y,z) specifies the axis that the rotation will be performed around.
- `obj.RotateWXYZ (double angle, double axis[3])` - Create a rotation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is in degrees, and (x,y,z) specifies the axis that the rotation will be performed around.

- `obj.RotateWXYZ (double angle, float axis[3])` - Create a rotation matrix and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is in degrees, and (x,y,z) specifies the axis that the rotation will be performed around.
- `obj.RotateX (double angle)` - Create a rotation matrix about the X, Y, or Z axis and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is expressed in degrees.
- `obj.RotateY (double angle)` - Create a rotation matrix about the X, Y, or Z axis and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is expressed in degrees.
- `obj.RotateZ (double angle)` - Create a rotation matrix about the X, Y, or Z axis and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics. The angle is expressed in degrees.
- `obj.Scale (double x, double y, double z)` - Create a scale matrix (i.e. set the diagonal elements to x, y, z) and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Scale (double s[3])` - Create a scale matrix (i.e. set the diagonal elements to x, y, z) and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Scale (float s[3])` - Create a scale matrix (i.e. set the diagonal elements to x, y, z) and concatenate it with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.SetMatrix (vtkMatrix4x4 matrix)` - Set the current matrix directly. This actually calls Identity(), followed by Concatenate(matrix).
- `obj.SetMatrix (double elements[16])` - Set the current matrix directly. This actually calls Identity(), followed by Concatenate(matrix).
- `obj.Concatenate (vtkMatrix4x4 matrix)` - Concatenates the matrix with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Concatenate (double elements[16])` - Concatenates the matrix with the current transformation according to PreMultiply or PostMultiply semantics.
- `obj.Concatenate (vtkLinearTransform transform)` - Concatenate the specified transform with the current transformation according to PreMultiply or PostMultiply semantics. The concatenation is pipelined, meaning that if any of the transformations are changed, even after Concatenate() is called, those changes will be reflected when you call TransformPoint().
- `obj.PreMultiply ()` - Sets the internal state of the transform to PreMultiply. All subsequent operations will occur before those already represented in the current transformation. In homogeneous matrix notation, M = M*A where M is the current transformation matrix and A is the applied matrix. The default is PreMultiply.
- `obj.PostMultiply ()` - Sets the internal state of the transform to PostMultiply. All subsequent operations will occur after those already represented in the current transformation. In homogeneous matrix notation, M = A*M where M is the current transformation matrix and A is the applied matrix. The default is PreMultiply.
- `int = obj.GetNumberOfConcatenatedTransforms ()` - Get the total number of transformations that are linked into this one via Concatenate() operations or via SetInput().
- `vtkLinearTransform = obj.GetConcatenatedTransform (int i)` - Get the x, y, z orientation angles from the transformation matrix as an array of three floating point values.

- `obj.GetOrientation (double orient[3])` - Get the x, y, z orientation angles from the transformation matrix as an array of three floating point values.
- `obj.GetOrientation (float orient[3])` - Get the x, y, z orientation angles from the transformation matrix as an array of three floating point values.
- `double = obj.GetOrientation ()` - Get the x, y, z orientation angles from the transformation matrix as an array of three floating point values.
- `obj.GetOrientationWXYZ (double wxyz[4])` - Return the wxyz angle+axis representing the current orientation. The angle is in degrees and the axis is a unit vector.
- `obj.GetOrientationWXYZ (float wxyz[4])` - Return the wxyz angle+axis representing the current orientation. The angle is in degrees and the axis is a unit vector.
- `double = obj.GetOrientationWXYZ ()` - Return the wxyz angle+axis representing the current orientation. The angle is in degrees and the axis is a unit vector.
- `obj.GetPosition (double pos[3])` - Return the position from the current transformation matrix as an array of three floating point numbers. This is simply returning the translation component of the 4x4 matrix.
- `obj.GetPosition (float pos[3])` - Return the position from the current transformation matrix as an array of three floating point numbers. This is simply returning the translation component of the 4x4 matrix.
- `double = obj.GetPosition ()` - Return the position from the current transformation matrix as an array of three floating point numbers. This is simply returning the translation component of the 4x4 matrix.
- `obj.GetScale (double scale[3])` - Return the scale factors of the current transformation matrix as an array of three float numbers. These scale factors are not necessarily about the x, y, and z axes unless unless the scale transformation was applied before any rotations.
- `obj.GetScale (float scale[3])` - Return the scale factors of the current transformation matrix as an array of three float numbers. These scale factors are not necessarily about the x, y, and z axes unless unless the scale transformation was applied before any rotations.
- `double = obj.GetScale ()` - Return the scale factors of the current transformation matrix as an array of three float numbers. These scale factors are not necessarily about the x, y, and z axes unless unless the scale transformation was applied before any rotations.
- `obj.GetInverse (vtkMatrix4x4 inverse)` - Return a matrix which is the inverse of the current transformation matrix.
- `obj.GetTranspose (vtkMatrix4x4 transpose)` - Return a matrix which is the transpose of the current transformation matrix. This is equivalent to the inverse if and only if the transformation is a pure rotation with no translation or scale.
- `obj.SetInput (vtkLinearTransform input)` - Set the input for this transformation. This will be used as the base transformation if it is set. This method allows you to build a transform pipeline: if the input is modified, then this transformation will automatically update accordingly. Note that the InverseFlag, controlled via Inverse(), determines whether this transformation will use the Input or the inverse of the Input.
- `vtkLinearTransform = obj.GetInput ()` - Set the input for this transformation. This will be used as the base transformation if it is set. This method allows you to build a transform pipeline: if the input is modified, then this transformation will automatically update accordingly. Note that the InverseFlag, controlled via Inverse(), determines whether this transformation will use the Input or the inverse of the Input.

- `int = obj.GetInverseFlag ()` - Get the inverse flag of the transformation. This controls whether it is the Input or the inverse of the Input that is used as the base transformation. The InverseFlag is flipped every time Inverse() is called. The InverseFlag is off when a transform is first created.
- `obj.Push ()` - Pushes the current transformation onto the transformation stack.
- `obj.Pop ()` - Deletes the transformation on the top of the stack and sets the top to the next transformation on the stack.
- `int = obj.CircuitCheck (vtkAbstractTransform transform)` - Check for self-reference. Will return true if concatenating with the specified transform, setting it to be our inverse, or setting it to be our input will create a circular reference. CircuitCheck is automatically called by SetInput(), SetInverse(), and Concatenate(vtkXTransform *). Avoid using this function, it is experimental.
- `vtkAbstractTransform = obj.GetInverse ()` - Make a new transform of the same type.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make a new transform of the same type.
- `long = obj.GetMTime ()` - Override GetMTime to account for input and concatenation.
- `obj.MultiplyPoint (float in[4], float out[4])` - Use this method only if you wish to compute the transformation in homogeneous (x,y,z,w) coordinates, otherwise use TransformPoint(). This method calls this-¿GetMatrix()-¿MultiplyPoint().
- `obj.MultiplyPoint (double in[4], double out[4])` - Use this method only if you wish to compute the transformation in homogeneous (x,y,z,w) coordinates, otherwise use TransformPoint(). This method calls this-¿GetMatrix()-¿MultiplyPoint().

30.143 vtkTransform2D

30.143.1 Usage

A vtkTransform2D can be used to describe the full range of linear (also known as affine) coordinate transformations in two dimensions, which are internally represented as a 3x3 homogeneous transformation matrix. When you create a new vtkTransform2D, it is always initialized to the identity transformation.

This class performs all of its operations in a right handed coordinate system with right handed rotations. Some other graphics libraries use left handed coordinate systems and rotations.

To create an instance of class vtkTransform2D, simply invoke its constructor as follows

```
obj = vtkTransform2D
```

30.143.2 Methods

The class vtkTransform2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransform2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransform2D = obj.NewInstance ()`
- `vtkTransform2D = obj.SafeDownCast (vtkObject o)`
- `obj.Identity ()` - Set the transformation to the identity transformation.
- `obj.Inverse ()` - Invert the transformation.

- `obj.Translate (double x, double y)` - Create a translation matrix and concatenate it with the current transformation.
- `obj.Translate (double x[2])` - Create a translation matrix and concatenate it with the current transformation.
- `obj.Translate (float x[2])` - Create a rotation matrix and concatenate it with the current transformation. The angle is in degrees.
- `obj.Rotate (double angle)` - Create a rotation matrix and concatenate it with the current transformation. The angle is in degrees.
- `obj.Scale (double x, double y)` - Create a scale matrix (i.e. set the diagonal elements to x, y) and concatenate it with the current transformation.
- `obj.Scale (double s[2])` - Create a scale matrix (i.e. set the diagonal elements to x, y) and concatenate it with the current transformation.
- `obj.Scale (float s[2])` - Set the current matrix directly.
- `obj.SetMatrix (vtkMatrix3x3 matrix)` - Set the current matrix directly.
- `obj.SetMatrix (double elements[9])` - Set the current matrix directly.
- `vtkMatrix3x3 = obj.GetMatrix ()` - Get the underlying 3x3 matrix.
- `obj.GetMatrix (vtkMatrix3x3 matrix)` - Get the underlying 3x3 matrix.
- `obj.GetPosition (double pos[2])` - Return the position from the current transformation matrix as an array of two floating point numbers. This is simply returning the translation component of the 3x3 matrix.
- `obj.GetPosition (float pos[2])` - Return a matrix which is the inverse of the current transformation matrix.
- `obj.GetInverse (vtkMatrix3x3 inverse)` - Return a matrix which is the inverse of the current transformation matrix.
- `obj.GetTranspose (vtkMatrix3x3 transpose)` - Return a matrix which is the transpose of the current transformation matrix. This is equivalent to the inverse if and only if the transformation is a pure rotation with no translation or scale.
- `long = obj.GetMTime ()` - Override GetMTime to account for input and concatenation.
- `obj.TransformPoints (float inPts, float outPts, int n)` - Apply the transformation to a series of points, and append the results to outPts. Where n is the number of points, and the float pointers are of length 2*n.
- `obj.TransformPoints (double inPts, double outPts, int n)` - Apply the transformation to a series of points, and append the results to outPts. Where n is the number of points, and the float pointers are of length 2*n.
- `obj.TransformPoints (vtkPoints2D inPts, vtkPoints2D outPts)` - Apply the transformation to a series of points, and append the results to outPts.
- `obj.InverseTransformPoints (float inPts, float outPts, int n)` - Apply the transformation to a series of points, and append the results to outPts. Where n is the number of points, and the float pointers are of length 2*n.
- `obj.InverseTransformPoints (double inPts, double outPts, int n)` - Apply the transformation to a series of points, and append the results to outPts. Where n is the number of points, and the float pointers are of length 2*n.

- `obj.InverseTransformPoints (vtkPoints2D inPts, vtkPoints2D outPts)` - Apply the transformation to a series of points, and append the results to outPts.
- `obj.MultiplyPoint (float in[3], float out[3])` - Use this method only if you wish to compute the transformation in homogeneous (x,y,w) coordinates, otherwise use TransformPoint(). This method calls this-¿GetMatrix()-¿MultiplyPoint().
- `obj.MultiplyPoint (double in[3], double out[3])` - Use this method only if you wish to compute the transformation in homogeneous (x,y,w) coordinates, otherwise use TransformPoint(). This method calls this-¿GetMatrix()-¿MultiplyPoint().

30.144 vtkTransformCollection

30.144.1 Usage

vtkTransformCollection is an object that creates and manipulates lists of objects of type vtkTransform.

To create an instance of class vtkTransformCollection, simply invoke its constructor as follows

```
obj = vtkTransformCollection
```

30.144.2 Methods

The class vtkTransformCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransformCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransformCollection = obj.NewInstance ()`
- `vtkTransformCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkTransform )` - Add a Transform to the list.
- `vtkTransform = obj.GetNextItem ()` - Get the next Transform in the list. Return NULL when the end of the list is reached.

30.145 vtkTypeFloat32Array

30.145.1 Usage

vtkTypeFloat32Array is an array of values of type vtkTypeFloat32. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeFloat32Array, simply invoke its constructor as follows

```
obj = vtkTypeFloat32Array
```

30.145.2 Methods

The class vtkTypeFloat32Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeFloat32Array class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkTypeFloat32Array = obj.NewInstance ()`
- `vtkTypeFloat32Array = obj.SafeDownCast (vtkObject o)`

30.146 vtkTypeFloat64Array

30.146.1 Usage

vtkTypeFloat64Array is an array of values of type vtkTypeFloat64. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeFloat64Array, simply invoke its constructor as follows

```
obj = vtkTypeFloat64Array
```

30.146.2 Methods

The class vtkTypeFloat64Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeFloat64Array class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTypeFloat64Array = obj.NewInstance ()`
- `vtkTypeFloat64Array = obj.SafeDownCast (vtkObject o)`

30.147 vtkTypeInt16Array

30.147.1 Usage

vtkTypeInt16Array is an array of values of type vtkTypeInt16. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeInt16Array, simply invoke its constructor as follows

```
obj = vtkTypeInt16Array
```

30.147.2 Methods

The class vtkTypeInt16Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeInt16Array class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTypeInt16Array = obj.NewInstance ()`
- `vtkTypeInt16Array = obj.SafeDownCast (vtkObject o)`

30.148 vtkTypeInt32Array

30.148.1 Usage

vtkTypeInt32Array is an array of values of type vtkTypeInt32. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeInt32Array, simply invoke its constructor as follows

```
obj = vtkTypeInt32Array
```

30.148.2 Methods

The class vtkTypeInt32Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeInt32Array class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTypeInt32Array = obj.NewInstance ()`
- `vtkTypeInt32Array = obj.SafeDownCast (vtkObject o)`

30.149 vtkTypeInt64Array

30.149.1 Usage

vtkTypeInt64Array is an array of values of type vtkTypeInt64. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeInt64Array, simply invoke its constructor as follows

```
obj = vtkTypeInt64Array
```

30.149.2 Methods

The class vtkTypeInt64Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeInt64Array class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTypeInt64Array = obj.NewInstance ()`
- `vtkTypeInt64Array = obj.SafeDownCast (vtkObject o)`

30.150 vtkTypeInt8Array

30.150.1 Usage

vtkTypeInt8Array is an array of values of type vtkTypeInt8. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeInt8Array, simply invoke its constructor as follows

```
obj = vtkTypeInt8Array
```

30.150.2 Methods

The class vtkTypeInt8Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeInt8Array class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTypeInt8Array = obj.NewInstance ()`
- `vtkTypeInt8Array = obj.SafeDownCast (vtkObject o)`

30.151 vtkTypeUInt16Array

30.151.1 Usage

vtkTypeUInt16Array is an array of values of type vtkTypeUInt16. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeUInt16Array, simply invoke its constructor as follows

```
obj = vtkTypeUInt16Array
```

30.151.2 Methods

The class vtkTypeUInt16Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeUInt16Array class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTypeUInt16Array = obj.NewInstance ()`
- `vtkTypeUInt16Array = obj.SafeDownCast (vtkObject o)`

30.152 vtkTypeUInt32Array

30.152.1 Usage

vtkTypeUInt32Array is an array of values of type vtkTypeUInt32. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeUInt32Array, simply invoke its constructor as follows

```
obj = vtkTypeUInt32Array
```

30.152.2 Methods

The class vtkTypeUInt32Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeUInt32Array class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkTypeUInt32Array = obj.NewInstance ()`
- `vtkTypeUInt32Array = obj.SafeDownCast (vtkObject o)`

30.153 vtkTypeUInt64Array

30.153.1 Usage

vtkTypeUInt64Array is an array of values of type vtkTypeUInt64. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeUInt64Array, simply invoke its constructor as follows

```
obj = vtkTypeUInt64Array
```

30.153.2 Methods

The class vtkTypeUInt64Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeUInt64Array class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTypeUInt64Array = obj.NewInstance ()`
- `vtkTypeUInt64Array = obj.SafeDownCast (vtkObject o)`

30.154 vtkTypeUInt8Array

30.154.1 Usage

vtkTypeUInt8Array is an array of values of type vtkTypeUInt8. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkTypeUInt8Array, simply invoke its constructor as follows

```
obj = vtkTypeUInt8Array
```

30.154.2 Methods

The class vtkTypeUInt8Array has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTypeUInt8Array class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTypeUInt8Array = obj.NewInstance ()`
- `vtkTypeUInt8Array = obj.SafeDownCast (vtkObject o)`

30.155 vtkUnicodeStringArray

30.155.1 Usage

.SECTION Thanks Developed by Timothy M. Shead (tshead@sandia.gov) at Sandia National Laboratories.
To create an instance of class vtkUnicodeStringArray, simply invoke its constructor as follows

```
obj = vtkUnicodeStringArray
```

30.155.2 Methods

The class vtkUnicodeStringArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnicodeStringArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnicodeStringArray = obj.NewInstance ()`
- `vtkUnicodeStringArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.Allocate (vtkIdType sz, vtkIdType ext)`
- `obj.Initialize ()`
- `int = obj.GetDataType ()`
- `int = obj.GetDataTypeSize ()`
- `int = obj.GetElementComponentSize ()`
- `obj.SetNumberOfTuples (vtkIdType number)`
- `obj.SetTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)`
- `obj.InsertTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)`
- `vtkIdType = obj.InsertNextTuple (vtkIdType j, vtkAbstractArray source)`
- `obj.DeepCopy (vtkAbstractArray da)`
- `obj.InterpolateTuple (vtkIdType i, vtkIdList ptIndices, vtkAbstractArray source, double weights)`
- `obj.InterpolateTuple (vtkIdType i, vtkIdType id1, vtkAbstractArray source1, vtkIdType id2, vtkAbstr`
- `obj.Squeeze ()`
- `int = obj.Resize (vtkIdType numTuples)`
- `long = obj.GetActualMemorySize ()`
- `int = obj.IsNumeric ()`
- `vtkArrayIterator = obj.NewIterator ()`
- `obj.DataChanged ()`
- `obj.ClearLookup ()`
- `obj.InsertNextUTF8Value (string )`
- `obj.SetUTF8Value (vtkIdType i, string )`
- `string = obj.GetUTF8Value (vtkIdType i)`

30.156 vtkUnsignedCharArray

30.156.1 Usage

vtkUnsignedCharArray is an array of values of type unsigned char. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkUnsignedCharArray, simply invoke its constructor as follows

```
obj = vtkUnsignedCharArray
```

30.156.2 Methods

The class vtkUnsignedCharArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnsignedCharArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnsignedCharArray = obj.NewInstance ()`
- `vtkUnsignedCharArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, string tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, string tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, string tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (string tuple)` - Get the data at a particular index.
- `char = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, char value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, char f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (char f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `obj.SetArray (string array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (string array, vtkIdType size, int save, int deleteMethod)`

30.157 vtkUnsignedIntArray

30.157.1 Usage

vtkUnsignedIntArray is an array of values of type unsigned int. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkUnsignedIntArray, simply invoke its constructor as follows

```
obj = vtkUnsignedIntArray
```

30.157.2 Methods

The class vtkUnsignedIntArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnsignedIntArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnsignedIntArray = obj.NewInstance ()`
- `vtkUnsignedIntArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, int tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, int tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, int tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (int tuple)` - Get the data at a particular index.
- `int = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, int value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, int f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (int f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `obj.SetArray (int array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (int array, vtkIdType size, int save, int deleteMethod)`

30.158 vtkUnsignedLongArray

30.158.1 Usage

vtkUnsignedLongArray is an array of values of type unsigned long. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkUnsignedLongArray, simply invoke its constructor as follows

```
obj = vtkUnsignedLongArray
```

30.158.2 Methods

The class vtkUnsignedLongArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnsignedLongArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnsignedLongArray = obj.NewInstance ()`
- `vtkUnsignedLongArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, long tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, long tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, long tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (long tuple)` - Get the data at a particular index.
- `long = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, long value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, long f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (long f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `obj.SetArray (long array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (long array, vtkIdType size, int save, int deleteMethod)`

30.159 vtkUnsignedLongLongArray

30.159.1 Usage

vtkUnsignedLongLongArray is an array of values of type unsigned long long. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkUnsignedLongLongArray, simply invoke its constructor as follows

```
obj = vtkUnsignedLongLongArray
```

30.159.2 Methods

The class vtkUnsignedLongLongArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnsignedLongLongArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnsignedLongLongArray = obj.NewInstance ()`
- `vtkUnsignedLongLongArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `long = obj.long GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, long long value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, long long f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (long long f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `long = obj.long WritePointer (vtkIdType id, vtkIdType number)` - Get the address of a particular data index. Performs no checks to verify that the memory has been allocated etc.
- `long = obj.long GetPointer (vtkIdType id)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.

30.160 vtkUnsignedShortArray

30.160.1 Usage

vtkUnsignedShortArray is an array of values of type unsigned short. It provides methods for insertion and retrieval of values and will automatically resize itself to hold new data.

To create an instance of class vtkUnsignedShortArray, simply invoke its constructor as follows

```
obj = vtkUnsignedShortArray
```

30.160.2 Methods

The class vtkUnsignedShortArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnsignedShortArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnsignedShortArray = obj.NewInstance ()`
- `vtkUnsignedShortArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataType ()` - Copy the tuple value into a user-provided array.
- `obj.GetTupleValue (vtkIdType i, short tuple)` - Set the tuple value at the ith location in the array.
- `obj.SetTupleValue (vtkIdType i, short tuple)` - Insert (memory allocation performed) the tuple into the ith location in the array.
- `obj.InsertTupleValue (vtkIdType i, short tuple)` - Insert (memory allocation performed) the tuple onto the end of the array.
- `vtkIdType = obj.InsertNextTupleValue (short tuple)` - Get the data at a particular index.
- `short = obj.GetValue (vtkIdType id)` - Set the data at a particular index. Does not do range checking. Make sure you use the method SetNumberOfValues() before inserting data.
- `obj.SetValue (vtkIdType id, short value)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `obj.SetNumberOfValues (vtkIdType number)` - Insert data at a specified position in the array.
- `obj.InsertValue (vtkIdType id, short f)` - Insert data at the end of the array. Return its location in the array.
- `vtkIdType = obj.InsertNextValue (short f)` - Get the address of a particular data index. Make sure data is allocated for the number of items requested. Set MaxId according to the number of data values requested.
- `obj.SetArray (short array, vtkIdType size, int save)` - This method lets the user specify data to be held by the array. The array argument is a pointer to the data. size is the size of the array supplied by the user. Set save to 1 to keep the class from deleting the array when it cleans up or reallocates memory. The class uses the actual array provided; it does not copy the data from the suppled array.
- `obj.SetArray (short array, vtkIdType size, int save, int deleteMethod)`

30.161 vtkVariantArray

30.161.1 Usage

.SECTION Thanks Thanks to Patricia Crossno, Ken Moreland, Andrew Wilson and Brian Wylie from Sandia National Laboratories for their help in developing this class.

To create an instance of class vtkVariantArray, simply invoke its constructor as follows

```
obj = vtkVariantArray
```

30.161.2 Methods

The class vtkVariantArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVariantArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVariantArray = obj.NewInstance ()`
- `vtkVariantArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.Allocate (vtkIdType sz, vtkIdType ext)` - Allocate memory for this array. Delete old storage only if necessary. Note that ext is no longer used.
- `obj.Initialize ()` - Release storage and reset array to initial state.
- `int = obj.GetDataType ()` - Return the underlying data type. An integer indicating data type is returned as specified in vtkSetGet.h.
- `int = obj.GetDataTypeSize ()` - Return the size of the underlying data type. For a bit, 1 is returned. For string 0 is returned. Arrays with variable length components return 0.
- `int = obj.GetElementComponentSize ()` - Return the size, in bytes, of the lowest-level element of an array. For vtkDataArray and subclasses this is the size of the data type. For vtkStringArray, this is sizeof(vtkStdString::value_type), which winds up being sizeof(char).
- `obj.SetNumberOfTuples (vtkIdType number)` - Set the number of tuples (a component group) in the array. Note that this may allocate space depending on the number of components.
- `obj.SetTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Set the tuple at the ith location using the jth tuple in the source array. This method assumes that the two arrays have the same type and structure. Note that range checking and memory allocation is not performed; use in conjunction with SetNumberOfTuples() to allocate space.
- `obj.InsertTuple (vtkIdType i, vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at ith location in this array. Note that memory allocation is performed as necessary to hold the data.
- `vtkIdType = obj.InsertNextTuple (vtkIdType j, vtkAbstractArray source)` - Insert the jth tuple in the source array, at the end in this array. Note that memory allocation is performed as necessary to hold the data. Returns the location at which the data was inserted.
- `obj.DeepCopy (vtkAbstractArray da)` - Deep copy of data. Implementation left to subclasses, which should support as many type conversions as possible given the data type.
- `obj.InterpolateTuple (vtkIdType i, vtkIdList ptIndices, vtkAbstractArray source, double weights)` - Set the ith tuple in this array as the interpolated tuple value, given the ptIndices in the source array and associated interpolation weights. This method assumes that the two arrays are of the same type and strcuture.
- `obj.InterpolateTuple (vtkIdType i, vtkIdType id1, vtkAbstractArray source1, vtkIdType id2, vtkAbstr`
- `obj.Squeeze ()` - Resize object to just fit data requirement. Reclaims extra memory.
- `int = obj.Resize (vtkIdType numTuples)` - Resize the array while conserving the data. Returns 1 if resizing succeeded and 0 otherwise.

- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this data array. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.
- `int = obj.IsNumeric ()` - Since each item can be of a different type, we say that a variant array is not numeric.
- `vtkArrayIterator = obj.NewIterator ()` - Subclasses must override this method and provide the right kind of templated vtkArrayIteratorTemplate.
- `obj.SetNumberOfValues (vtkIdType number)` - Specify the number of values for this object to hold. Does an allocation as well as setting the MaxId ivar. Used in conjunction with SetValue() method for fast insertion.
- `vtkIdType = obj.GetNumberOfValues ()` - Tell the array explicitly that the data has changed. This is only necessary to call when you modify the array contents without using the array's API (i.e. you retrieve a pointer to the data and modify the array contents). You need to call this so that the fast lookup will know to rebuild itself. Otherwise, the lookup functions will give incorrect results.
- `obj.DataChanged ()` - Tell the array explicitly that the data has changed. This is only necessary to call when you modify the array contents without using the array's API (i.e. you retrieve a pointer to the data and modify the array contents). You need to call this so that the fast lookup will know to rebuild itself. Otherwise, the lookup functions will give incorrect results.
- `obj.DataElementChanged (vtkIdType id)` - Tell the array explicitly that a single data element has changed. Like DataChanged(), then is only necessary when you modify the array contents without using the array's API.
- `obj.ClearLookup ()` - Delete the associated fast lookup data structure on this array, if it exists. The lookup will be rebuilt on the next call to a lookup function.
- `~vtkVariantArray = obj.()` - This destructor is public to work around a bug in version 1.36.0 of the Boost.Serialization library.

30.162 vtkVersion

30.162.1 Usage

Holds methods for defining/determining the current vtk version (major, minor, build).

To create an instance of class vtkVersion, simply invoke its constructor as follows

```
obj = vtkVersion
```

30.162.2 Methods

The class vtkVersion has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVersion class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVersion = obj.NewInstance ()`
- `vtkVersion = obj.SafeDownCast (vtkObject o)`

30.163 vtkVoidArray

30.163.1 Usage

vtkVoidArray is an array of pointers to void. It provides methods for insertion and retrieval of these pointers values, and will automatically resize itself to hold new data.

To create an instance of class vtkVoidArray, simply invoke its constructor as follows

```
obj = vtkVoidArray
```

30.163.2 Methods

The class vtkVoidArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVoidArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVoidArray = obj.NewInstance ()`
- `vtkVoidArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.Allocate (vtkIdType sz, vtkIdType ext)` - Allocate memory for this array. Delete old storage only if necessary. Note that the parameter ext is no longer used.
- `obj.Initialize ()` - Release storage and reset array to initial state.
- `int = obj.GetDataType ()` - Return the size of the data contained in the array.
- `int = obj.GetDataTypeSize ()` - Set the number of void* pointers held in the array.
- `obj.SetNumberOfPointers (vtkIdType number)` - Get the number of void* pointers held in the array.
- `vtkIdType = obj.GetNumberOfPointers ()` - Get the void* pointer at the ith location.
- `obj.Reset ()` - Resize the array to just fit the inserted memory. Reclaims extra memory.
- `obj.Squeeze ()` - Get the address of a particular data index. Performs no checks to verify that the memory has been allocated etc.
- `obj.DeepCopy (vtkVoidArray va)` - Deep copy of another void array.

30.164 vtkWarpTransform

30.164.1 Usage

vtkWarpTransform provides a generic interface for nonlinear warp transformations.

To create an instance of class vtkWarpTransform, simply invoke its constructor as follows

```
obj = vtkWarpTransform
```

30.164.2 Methods

The class vtkWarpTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWarpTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWarpTransform = obj.NewInstance ()`
- `vtkWarpTransform = obj.SafeDownCast (vtkObject o)`
- `obj.Inverse ()` - Invert the transformation. Warp transformations are usually inverted using an iterative technique such as Newton's method. The inverse transform is usually around five or six times as computationally expensive as the forward transform.
- `int = obj.GetInverseFlag ()` - Get the inverse flag of the transformation. This flag is set to zero when the transformation is first created, and is flipped each time Inverse() is called.
- `obj.SetInverseTolerance (double )` - Set the tolerance for inverse transformation. The default is 0.001.
- `double = obj.GetInverseTolerance ()` - Set the tolerance for inverse transformation. The default is 0.001.
- `obj.SetInverseIterations (int )` - Set the maximum number of iterations for the inverse transformation. The default is 500, but usually only 2 to 5 iterations are used. The inversion method is fairly robust, and it should converge for nearly all smooth transformations that do not fold back on themselves.
- `int = obj.GetInverseIterations ()` - Set the maximum number of iterations for the inverse transformation. The default is 500, but usually only 2 to 5 iterations are used. The inversion method is fairly robust, and it should converge for nearly all smooth transformations that do not fold back on themselves.
- `obj.InternalTransformPoint (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.TemplateTransformPoint (float in[3], float out[3])` - Do not use these methods. They exists only as a work-around for internal templated functions (I really didn't want to make the Forward/Inverse methods public, is there a decent work around for this sort of thing?)
- `obj.TemplateTransformPoint (double in[3], double out[3])` - Do not use these methods. They exists only as a work-around for internal templated functions (I really didn't want to make the Forward/Inverse methods public, is there a decent work around for this sort of thing?)
- `obj.TemplateTransformInverse (float in[3], float out[3])` - Do not use these methods. They exists only as a work-around for internal templated functions (I really didn't want to make the Forward/Inverse methods public, is there a decent work around for this sort of thing?)
- `obj.TemplateTransformInverse (double in[3], double out[3])` - Do not use these methods. They exists only as a work-around for internal templated functions (I really didn't want to make the Forward/Inverse methods public, is there a decent work around for this sort of thing?)

30.165 vtkWindow

30.165.1 Usage

vtkWindow is an abstract object to specify the behavior of a rendering window. It contains vtkViewports.

To create an instance of class vtkWindow, simply invoke its constructor as follows

```
obj = vtkWindow
```

30.165.2 Methods

The class vtkWindow has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWindow class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWindow = obj.NewInstance ()`
- `vtkWindow = obj.SafeDownCast (vtkObject o)`
- `obj.SetWindowInfo (string )` - These are window system independent methods that are used to help interface vtkWindow to native windowing systems.
- `obj.SetParentInfo (string )` - These are window system independent methods that are used to help interface vtkWindow to native windowing systems.
- `int = obj.GetPosition ()` - Set/Get the position in screen coordinates of the rendering window.
- `obj.SetPosition (int , int )` - Set/Get the position in screen coordinates of the rendering window.
- `obj.SetPosition (int a[2])` - Set/Get the position in screen coordinates of the rendering window.
- `int = obj.GetSize ()` - Set/Get the size of the window in screen coordinates in pixels.
- `obj.SetSize (int , int )` - Set/Get the size of the window in screen coordinates in pixels.
- `obj.SetSize (int a[2])` - Set/Get the size of the window in screen coordinates in pixels.
- `obj.SetMapped (int )` - Keep track of whether the rendering window has been mapped to screen.
- `int = obj.GetMapped ()` - Keep track of whether the rendering window has been mapped to screen.
- `obj.MappedOn ()` - Keep track of whether the rendering window has been mapped to screen.
- `obj.MappedOff ()` - Keep track of whether the rendering window has been mapped to screen.
- `obj.SetErase (int )` - Turn on/off erasing the screen between images. This allows multiple exposure sequences if turned on. You will need to turn double buffering off or make use of the SwapBuffers methods to prevent you from swapping buffers between exposures.
- `int = obj.GetErase ()` - Turn on/off erasing the screen between images. This allows multiple exposure sequences if turned on. You will need to turn double buffering off or make use of the SwapBuffers methods to prevent you from swapping buffers between exposures.
- `obj.EraseOn ()` - Turn on/off erasing the screen between images. This allows multiple exposure sequences if turned on. You will need to turn double buffering off or make use of the SwapBuffers methods to prevent you from swapping buffers between exposures.

- `obj.EraseOff ()` - Turn on/off erasing the screen between images. This allows multiple exposure sequences if turned on. You will need to turn double buffering off or make use of the SwapBuffers methods to prevent you from swapping buffers between exposures.
- `obj.SetDoubleBuffer (int )` - Keep track of whether double buffering is on or off
- `int = obj.GetDoubleBuffer ()` - Keep track of whether double buffering is on or off
- `obj.DoubleBufferOn ()` - Keep track of whether double buffering is on or off
- `obj.DoubleBufferOff ()` - Keep track of whether double buffering is on or off
- `string = obj.GetWindowName ()` - Get name of rendering window
- `obj.SetWindowName (string )` - Get name of rendering window
- `obj.Render ()` - Ask each viewport owned by this Window to render its image and synchronize this process.
- `int = obj.GetPixelData (int x, int y, int x2, int y2, int front, vtkUnsignedCharArray data)` - Get the pixel data of an image, transmitted as RGBRGBRGB. The front argument indicates if the front buffer should be used or the back buffer. It is the caller's responsibility to delete the resulting array. It is very important to realize that the memory in this array is organized from the bottom of the window to the top. The origin of the screen is in the lower left corner. The y axis increases as you go up the screen. So the storage of pixels is from left to right and from bottom to top. (x,y) is any corner of the rectangle. (x2,y2) is its opposite corner on the diagonal.
- `int = obj.GetDPI ()` - Return a best estimate to the dots per inch of the display device being rendered (or printed).
- `obj.SetDPI (int )` - Return a best estimate to the dots per inch of the display device being rendered (or printed).
- `int = obj.GetDPIMinValue ()` - Return a best estimate to the dots per inch of the display device being rendered (or printed).
- `int = obj.GetDPIMaxValue ()` - Return a best estimate to the dots per inch of the display device being rendered (or printed).
- `obj.SetOffScreenRendering (int )` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `int = obj.GetOffScreenRendering ()` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `obj.OffScreenRenderingOn ()` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `obj.OffScreenRenderingOff ()` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `obj.MakeCurrent ()` - Make the window current. May be overridden in subclasses to do for example a glXMakeCurrent or a wglMakeCurrent.

- `obj.SetTileScale (int , int )` - These methods are used by vtkWindowToImageFilter to tell a VTK window to simulate a larger window by tiling. For 3D geometry these methods have no impact. It is just in handling annotation that this information must be available to the mappers and the coordinate calculations.

- `obj.SetTileScale (int a[2])` - These methods are used by vtkWindowToImageFilter to tell a VTK window to simulate a larger window by tiling. For 3D geometry these methods have no impact. It is just in handling annotation that this information must be available to the mappers and the coordinate calculations.

- `int = obj. GetTileScale ()` - These methods are used by vtkWindowToImageFilter to tell a VTK window to simulate a larger window by tiling. For 3D geometry these methods have no impact. It is just in handling annotation that this information must be available to the mappers and the coordinate calculations.

- `obj.SetTileScale (int s)` - These methods are used by vtkWindowToImageFilter to tell a VTK window to simulate a larger window by tiling. For 3D geometry these methods have no impact. It is just in handling annotation that this information must be available to the mappers and the coordinate calculations.

- `obj.SetTileViewport (double , double , double , double )` - These methods are used by vtkWindowToImageFilter to tell a VTK window to simulate a larger window by tiling. For 3D geometry these methods have no impact. It is just in handling annotation that this information must be available to the mappers and the coordinate calculations.

- `obj.SetTileViewport (double a[4])` - These methods are used by vtkWindowToImageFilter to tell a VTK window to simulate a larger window by tiling. For 3D geometry these methods have no impact. It is just in handling annotation that this information must be available to the mappers and the coordinate calculations.

- `double = obj. GetTileViewport ()` - These methods are used by vtkWindowToImageFilter to tell a VTK window to simulate a larger window by tiling. For 3D geometry these methods have no impact. It is just in handling annotation that this information must be available to the mappers and the coordinate calculations.

30.166 vtkWindowLevelLookupTable

30.166.1 Usage

vtkWindowLevelLookupTable is an object that is used by mapper objects to map scalar values into rgba (red-green-blue-alpha transparency) color specification, or rgba into scalar values. The color table can be created by direct insertion of color values, or by specifying a window and level. Window / Level is used in medical imaging to specify a linear greyscale ramp. The Level is the center of the ramp. The Window is the width of the ramp.

To create an instance of class vtkWindowLevelLookupTable, simply invoke its constructor as follows

```
obj = vtkWindowLevelLookupTable
```

30.166.2 Methods

The class vtkWindowLevelLookupTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWindowLevelLookupTable class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkWindowLevelLookupTable = obj.NewInstance ()`
- `vtkWindowLevelLookupTable = obj.SafeDownCast (vtkObject o)`
- `obj.Build ()` - Generate lookup table as a linear ramp between MinimumTableValue and MaximumTableValue.
- `obj.SetWindow (double window)` - Set the window for the lookup table. The window is the difference between TableRange[0] and TableRange[1].
- `double = obj.GetWindow ()` - Set the window for the lookup table. The window is the difference between TableRange[0] and TableRange[1].
- `obj.SetLevel (double level)` - Set the Level for the lookup table. The level is the average of TableRange[0] and TableRange[1].
- `double = obj.GetLevel ()` - Set the Level for the lookup table. The level is the average of TableRange[0] and TableRange[1].
- `obj.SetInverseVideo (int iv)` - Set inverse video on or off. You can achieve the same effect by switching the MinimumTableValue and the MaximumTableValue.
- `int = obj.GetInverseVideo ()` - Set inverse video on or off. You can achieve the same effect by switching the MinimumTableValue and the MaximumTableValue.
- `obj.InverseVideoOn ()` - Set inverse video on or off. You can achieve the same effect by switching the MinimumTableValue and the MaximumTableValue.
- `obj.InverseVideoOff ()` - Set inverse video on or off. You can achieve the same effect by switching the MinimumTableValue and the MaximumTableValue.
- `obj.SetMinimumTableValue (double , double , double , double )` - Set the minimum table value. All lookup table entries below the start of the ramp will be set to this color. After you change this value, you must re-build the lookup table.
- `obj.SetMinimumTableValue (double a[4])` - Set the minimum table value. All lookup table entries below the start of the ramp will be set to this color. After you change this value, you must re-build the lookup table.
- `double = obj. GetMinimumTableValue ()` - Set the minimum table value. All lookup table entries below the start of the ramp will be set to this color. After you change this value, you must re-build the lookup table.
- `obj.SetMaximumTableValue (double , double , double , double )` - Set the maximum table value. All lookup table entries above the end of the ramp will be set to this color. After you change this value, you must re-build the lookup table.
- `obj.SetMaximumTableValue (double a[4])` - Set the maximum table value. All lookup table entries above the end of the ramp will be set to this color. After you change this value, you must re-build the lookup table.
- `double = obj. GetMaximumTableValue ()` - Set the maximum table value. All lookup table entries above the end of the ramp will be set to this color. After you change this value, you must re-build the lookup table.
- `obj.SetMinimumColor (int r, int g, int b, int a)` - @deprecated For backwards compatibility: specify the color using integers in the range [0,255].

- `obj.SetMinimumColor (char rgba[4])` - @deprecated For backwards compatibility: specify the color using integers in the range [0,255].
- `obj.GetMinimumColor (char rgba[4])` - @deprecated For backwards compatibility: specify the color using integers in the range [0,255].
- `obj.SetMaximumColor (int r, int g, int b, int a)` - @deprecated For backwards compatibility: specify the color using integers in the range [0,255].
- `obj.SetMaximumColor (char rgba[4])` - @deprecated For backwards compatibility: specify the color using integers in the range [0,255].
- `obj.GetMaximumColor (char rgba[4])` - @deprecated For backwards compatibility: specify the color using integers in the range [0,255].

30.167 vtkXMLDataElement

30.167.1 Usage

vtkXMLDataElement is used by vtkXMLDataParser to represent an XML element. It provides methods to access the element's attributes and nested elements in a convenient manner. This allows easy traversal of an input XML file by vtkXMLReader and its subclasses.

To create an instance of class vtkXMLDataElement, simply invoke its constructor as follows

```
obj = vtkXMLDataElement
```

30.167.2 Methods

The class vtkXMLDataElement has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLDataElement class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLDataElement = obj.NewInstance ()`
- `vtkXMLDataElement = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetName ()` - Set/Get the name of the element. This is its XML tag.
- `obj.SetName (string \_arg)` - Set/Get the name of the element. This is its XML tag.
- `string = obj.GetId ()` - Set/Get the value of the id attribute of the element, if any.
- `obj.SetId (string )` - Set/Get the value of the id attribute of the element, if any.
- `string = obj.GetAttribute (string name)` - Get the attribute with the given name. If it doesn't exist, returns 0.
- `obj.SetAttribute (string name, string value)` - Set the attribute with the given name and value. If it doesn't exist, adds it.
- `obj.SetCharacterData (string c, int length)` - Set/Get the character data between XML start/end tags.
- `string = obj.GetCharacterData ()` - Set/Get the character data between XML start/end tags.

- `obj.SetIntAttribute (string name, int value)` - Set the attribute with the given name. We can not use the same GetScalarAttribute() construct since the compiler will not be able to resolve between SetAttribute(..., int) and SetAttribute(..., unsigned long).
- `obj.SetFloatAttribute (string name, float value)` - Set the attribute with the given name. We can not use the same GetScalarAttribute() construct since the compiler will not be able to resolve between SetAttribute(..., int) and SetAttribute(..., unsigned long).
- `obj.SetDoubleAttribute (string name, double value)` - Set the attribute with the given name. We can not use the same GetScalarAttribute() construct since the compiler will not be able to resolve between SetAttribute(..., int) and SetAttribute(..., unsigned long).
- `obj.SetUnsignedLongAttribute (string name, long value)` - Set the attribute with the given name. We can not use the same GetScalarAttribute() construct since the compiler will not be able to resolve between SetAttribute(..., int) and SetAttribute(..., unsigned long).
- `int = obj.GetVectorAttribute (string name, int length, int value)` - Get the attribute with the given name and converted to a scalar value. Returns length of vector read.
- `int = obj.GetVectorAttribute (string name, int length, float value)` - Get the attribute with the given name and converted to a scalar value. Returns length of vector read.
- `int = obj.GetVectorAttribute (string name, int length, double value)` - Get the attribute with the given name and converted to a scalar value. Returns length of vector read.
- `int = obj.GetVectorAttribute (string name, int length, long value)` - Get the attribute with the given name and converted to a scalar value. Returns length of vector read.
- `obj.SetVectorAttribute (string name, int length, int value)` - Set the attribute with the given name.
- `obj.SetVectorAttribute (string name, int length, float value)` - Set the attribute with the given name.
- `obj.SetVectorAttribute (string name, int length, double value)` - Set the attribute with the given name.
- `obj.SetVectorAttribute (string name, int length, long value)` - Set the attribute with the given name.
- `int = obj.GetNumberOfAttributes ()` - Get the number of attributes.
- `string = obj.GetAttributeName (int idx)` - Get the n-th attribute name. Returns 0 if there is no such attribute.
- `string = obj.GetAttributeValue (int idx)` - Get the n-th attribute value. Returns 0 if there is no such attribute.
- `obj.RemoveAttribute (string name)` - Remove one or all attributes.
- `obj.RemoveAllAttributes ()` - Remove one or all attributes.
- `vtkXMLDataElement = obj.GetParent ()` - Set/Get the parent of this element.
- `obj.SetParent (vtkXMLDataElement parent)` - Set/Get the parent of this element.
- `vtkXMLDataElement = obj.GetRoot ()` - Get root of the XML tree this element is part of.
- `int = obj.GetNumberOfNestedElements ()` - Get the number of elements nested in this one.
- `vtkXMLDataElement = obj.GetNestedElement (int index)` - Get the element nested in this one at the given index.

- `obj.AddNestedElement (vtkXMLDataElement element)` - Add nested element
- `obj.RemoveNestedElement (vtkXMLDataElement )` - Remove nested element.
- `obj.RemoveAllNestedElements ()` - Remove all nested elements.
- `vtkXMLDataElement = obj.FindNestedElement (string id)` - Find the first nested element with the given id, given name, or given name and id. WARNING: the search is only performed on the children, not the grand-children.
- `vtkXMLDataElement = obj.FindNestedElementWithName (string name)` - Find the first nested element with the given id, given name, or given name and id. WARNING: the search is only performed on the children, not the grand-children.
- `vtkXMLDataElement = obj.FindNestedElementWithNameAndId (string name, string id)` - Find the first nested element with the given id, given name, or given name and id. WARNING: the search is only performed on the children, not the grand-children.
- `vtkXMLDataElement = obj.FindNestedElementWithNameAndAttribute (string name, string att\_name, strin` - Find the first nested element with the given id, given name, or given name and id. WARNING: the search is only performed on the children, not the grand-children.
- `vtkXMLDataElement = obj.LookupElementWithName (string name)` - Find the first nested element with given name. WARNING: the search is performed on the whole XML tree.
- `vtkXMLDataElement = obj.LookupElement (string id)` - Lookup the element with the given id, starting at this scope.
- `long = obj.GetXMLByteIndex ()` - Set/Get the offset from the beginning of the XML document to this element.
- `obj.SetXMLByteIndex (long )` - Set/Get the offset from the beginning of the XML document to this element.
- `int = obj.IsEqualTo (vtkXMLDataElement elem)` - Check if the instance has the same name, attributes, character data and nested elements contents than the given element (this method is applied recursively on the nested elements, and they must be stored in the same order). Warning: Id, Parent, XMLByteIndex are ignored.
- `obj.DeepCopy (vtkXMLDataElement elem)` - Copy this element from another of the same type (elem), recursively. Old attributes and nested elements are removed, new ones are created given the contents of 'elem'. Warning: Parent is ignored.
- `obj.SetAttributeEncoding (int )` - Get/Set the internal character encoding of the attributes. Default type is VTK_ENCODING_UTF_8. Note that a vtkXMLDataParser has its own AttributesEncoding ivar. If this ivar is set to something other than VTK_ENCODING_NONE, it will be used to set the attribute encoding of each vtkXMLDataElement created by this vtkXMLDataParser.
- `int = obj.GetAttributeEncodingMinValue ()` - Get/Set the internal character encoding of the attributes. Default type is VTK_ENCODING_UTF_8. Note that a vtkXMLDataParser has its own AttributesEncoding ivar. If this ivar is set to something other than VTK_ENCODING_NONE, it will be used to set the attribute encoding of each vtkXMLDataElement created by this vtkXMLDataParser.
- `int = obj.GetAttributeEncodingMaxValue ()` - Get/Set the internal character encoding of the attributes. Default type is VTK_ENCODING_UTF_8. Note that a vtkXMLDataParser has its own AttributesEncoding ivar. If this ivar is set to something other than VTK_ENCODING_NONE, it will be used to set the attribute encoding of each vtkXMLDataElement created by this vtkXMLDataParser.

- `int = obj.GetAttributeEncoding ()` - Get/Set the internal character encoding of the attributes. Default type is VTK_ENCODING_UTF_8. Note that a vtkXMLDataParser has its own AttributesEncoding ivar. If this ivar is set to something other than VTK_ENCODING_NONE, it will be used to set the attribute encoding of each vtkXMLDataElement created by this vtkXMLDataParser.

- `obj.PrintXML (string fname)` - Prints element tree as XML.

- `int = obj.GetCharacterDataWidth ()` - Get/Set the width (in number of fields) that character data (that between open and closing tags ie. ¡X¿ ... ¡/X¿) is printed. If the width is less than one the tag's character data is printed all on one line. If it is greater than one the character data is streamed insterting line feeds every width number of fields. See PrintXML.

- `obj.SetCharacterDataWidth (int )` - Get/Set the width (in number of fields) that character data (that between open and closing tags ie. ¡X¿ ... ¡/X¿) is printed. If the width is less than one the tag's character data is printed all on one line. If it is greater than one the character data is streamed insterting line feeds every width number of fields. See PrintXML.

30.168 vtkXMLFileOutputWindow

30.168.1 Usage

Writes debug/warning/error output to an XML file. Uses prefined XML tags for each text display method. The text is processed to replace XML markup characters.

DisplayText - ¡Text¿
DisplayErrorText - ¡Error¿
DisplayWarningText - ¡Warning¿
DisplayGenericWarningText - ¡GenericWarning¿
DisplayDebugText - ¡Debug¿

The method DisplayTag outputs the text unprocessed. To use this class, instantiate it and then call SetInstance(this).

To create an instance of class vtkXMLFileOutputWindow, simply invoke its constructor as follows

```
obj = vtkXMLFileOutputWindow
```

30.168.2 Methods

The class vtkXMLFileOutputWindow has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLFileOutputWindow class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkXMLFileOutputWindow = obj.NewInstance ()`

- `vtkXMLFileOutputWindow = obj.SafeDownCast (vtkObject o)`

- `obj.DisplayText (string )` - Put the text into the log file. The text is processed to replace &, ¡, ¿ with &, &lt, and &gt. Each display method outputs a different XML tag.

- `obj.DisplayErrorText (string )` - Put the text into the log file. The text is processed to replace &, ¡, ¿ with &, &lt, and &gt. Each display method outputs a different XML tag.

- `obj.DisplayWarningText (string )` - Put the text into the log file. The text is processed to replace &, ¡, ¿ with &, &lt, and &gt. Each display method outputs a different XML tag.

- `obj.DisplayGenericWarningText (string )` - Put the text into the log file. The text is processed to replace &, ¡, ¿ with &amp, &lt, and &gt. Each display method outputs a different XML tag.
- `obj.DisplayDebugText (string )` - Put the text into the log file. The text is processed to replace &, ¡, ¿ with &amp, &lt, and &gt. Each display method outputs a different XML tag.
- `obj.DisplayTag (string )` - Put the text into the log file without processing it.

Chapter 31

Visualization Toolkit Filtering Classes

31.1 vtkAbstractCellLocator

31.1.1 Usage

vtkAbstractCellLocator is a spatial search object to quickly locate cells in 3D. vtkAbstractCellLocator supplies a basic interface which concrete subclasses should implement.

.SECTION Warning When deriving a class from vtkAbstractCellLocator, one should include the 'hidden' member functions by the following construct in the derived class

```
//BTX
 using vtkAbstractCellLocator::IntersectWithLine;
 using vtkAbstractCellLocator::FindClosestPoint;
 using vtkAbstractCellLocator::FindClosestPointWithinRadius;
//ETX
```

To create an instance of class vtkAbstractCellLocator, simply invoke its constructor as follows

```
obj = vtkAbstractCellLocator
```

31.1.2 Methods

The class vtkAbstractCellLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractCellLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractCellLocator = obj.NewInstance ()`
- `vtkAbstractCellLocator = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfCellsPerNode (int )` - Specify the preferred/maximum number of cells in each node/bucket. Default 32. Locators generally operate by subdividing space into smaller regions until the number of cells in each region (or node) reaches the desired level.
- `int = obj.GetNumberOfCellsPerNodeMinValue ()` - Specify the preferred/maximum number of cells in each node/bucket. Default 32. Locators generally operate by subdividing space into smaller regions until the number of cells in each region (or node) reaches the desired level.

- `int = obj.GetNumberOfCellsPerNodeMaxValue ()` - Specify the preferred/maximum number of cells in each node/bucket. Default 32. Locators generally operate by subdividing space into smaller regions until the number of cells in each region (or node) reaches the desired level.
- `int = obj.GetNumberOfCellsPerNode ()` - Specify the preferred/maximum number of cells in each node/bucket. Default 32. Locators generally operate by subdividing space into smaller regions until the number of cells in each region (or node) reaches the desired level.
- `obj.SetCacheCellBounds (int )` - Boolean controls whether the bounds of each cell are computed only once and then saved. Should be 10 to 20calling any of the Intersect/Find routines and the extra memory won't cause disk caching (24 extra bytes per cell are required to save the bounds).
- `int = obj.GetCacheCellBounds ()` - Boolean controls whether the bounds of each cell are computed only once and then saved. Should be 10 to 20calling any of the Intersect/Find routines and the extra memory won't cause disk caching (24 extra bytes per cell are required to save the bounds).
- `obj.CacheCellBoundsOn ()` - Boolean controls whether the bounds of each cell are computed only once and then saved. Should be 10 to 20calling any of the Intersect/Find routines and the extra memory won't cause disk caching (24 extra bytes per cell are required to save the bounds).
- `obj.CacheCellBoundsOff ()` - Boolean controls whether the bounds of each cell are computed only once and then saved. Should be 10 to 20calling any of the Intersect/Find routines and the extra memory won't cause disk caching (24 extra bytes per cell are required to save the bounds).
- `obj.SetRetainCellLists (int )` - Boolean controls whether to maintain list of cells in each node. not applicable to all implementations, but if the locator is being used as a geometry simplification technique, there is no need to keep them.
- `int = obj.GetRetainCellLists ()` - Boolean controls whether to maintain list of cells in each node. not applicable to all implementations, but if the locator is being used as a geometry simplification technique, there is no need to keep them.
- `obj.RetainCellListsOn ()` - Boolean controls whether to maintain list of cells in each node. not applicable to all implementations, but if the locator is being used as a geometry simplification technique, there is no need to keep them.
- `obj.RetainCellListsOff ()` - Boolean controls whether to maintain list of cells in each node. not applicable to all implementations, but if the locator is being used as a geometry simplification technique, there is no need to keep them.
- `obj.SetLazyEvaluation (int )` - Most Locators build their search structures during BuildLocator but some may delay construction until it is actually needed. If LazyEvaluation is supported, this turns on/off the feature. if not supported, it is ignored.
- `int = obj.GetLazyEvaluation ()` - Most Locators build their search structures during BuildLocator but some may delay construction until it is actually needed. If LazyEvaluation is supported, this turns on/off the feature. if not supported, it is ignored.
- `obj.LazyEvaluationOn ()` - Most Locators build their search structures during BuildLocator but some may delay construction until it is actually needed. If LazyEvaluation is supported, this turns on/off the feature. if not supported, it is ignored.
- `obj.LazyEvaluationOff ()` - Most Locators build their search structures during BuildLocator but some may delay construction until it is actually needed. If LazyEvaluation is supported, this turns on/off the feature. if not supported, it is ignored.
- `obj.SetUseExistingSearchStructure (int )` - Some locators support querying a new dataset without rebuilding the search structure (typically this may occur when a dataset changes due to a time update, but is actually the same topology) Turning on this flag enables some locators to skip the rebuilding phase

- `int = obj.GetUseExistingSearchStructure ()` - Some locators support querying a new dataset without rebuilding the search structure (typically this may occur when a dataset changes due to a time update, but is actually the same topology) Turning on this flag enables some locators to skip the rebuilding phase
- `obj.UseExistingSearchStructureOn ()` - Some locators support querying a new dataset without rebuilding the search structure (typically this may occur when a dataset changes due to a time update, but is actually the same topology) Turning on this flag enables some locators to skip the rebuilding phase
- `obj.UseExistingSearchStructureOff ()` - Some locators support querying a new dataset without rebuilding the search structure (typically this may occur when a dataset changes due to a time update, but is actually the same topology) Turning on this flag enables some locators to skip the rebuilding phase
- `int = obj.IntersectWithLine (double p1[3], double p2[3], vtkPoints points, vtkIdList cellIds)` - Take the passed line segment and intersect it with the data set. This method assumes that the data set is a vtkPolyData that describes a closed surface, and the intersection points that are returned in 'points' alternate between entrance points and exit points. The return value of the function is 0 if no intersections were found, -1 if point 'a0' lies inside the closed surface, or +1 if point 'a0' lies outside the closed surface. Either 'points' or 'cellIds' can be set to NULL if you don't want to receive that information. This method is currently only implemented in vtkOBBTree
- `obj.FindCellsWithinBounds (double bbox, vtkIdList cells)` - Return a list of unique cell ids inside of a given bounding box. The user must provide the vtkIdList to populate. This method returns data only after the locator has been built.
- `obj.FindCellsAlongLine (double p1[3], double p2[3], double tolerance, vtkIdList cells)` - Given a finite line defined by the two points (p1,p2), return the list of unique cell ids in the buckets containing the line. It is possible that an empty cell list is returned. The user must provide the vtkIdList to populate. This method returns data only after the locator has been built.
- `vtkIdType = obj.FindCell (double x[3])` - Returns the Id of the cell containing the point, returns -1 if no cell found. This interface uses a tolerance of zero
- `vtkIdType = obj.FindCell (double x[3], double tol2, vtkGenericCell GenCell, double pcoords[3], doub` - Find the cell containing a given point. returns -1 if no cell found the cell parameters are copied into the supplied variables, a cell must be provided to store the information.
- `bool = obj.InsideCellBounds (double x[3], vtkIdType cell\_ID)` - Quickly test if a point is inside the bounds of a particular cell. Some locators cache cell bounds and this function can make use of fast access to the data.

31.2 vtkAbstractInterpolatedVelocityField

31.2.1 Usage

vtkAbstractInterpolatedVelocityField acts as a continuous velocity field by performing cell interpolation on the underlying vtkDataSet. This is an abstract sub-class of vtkFunctionSet, NumberOfIndependentVariables = 4 (x,y,z,t) and NumberOfFunctions = 3 (u,v,w). With a brute-force scheme, every time an evaluation is performed, the target cell containing point (x,y,z) needs to be found by calling FindCell(), via either vtkDataSet or vtkAbstractCelllocator's sub-classes (vtkCellLocator & vtkModifiedBSPTree). As it incurs a large cost, one (for vtkCellLocatorInterpolatedVelocityField via vtkAbstractCellLocator) or two (for vtkInterpolatedVelocityField via vtkDataSet that involves vtkPointLocator in addressing vtkPointSet) levels of cell caching may be exploited to increase the performance.

For vtkInterpolatedVelocityField, level #0 begins with intra-cell caching. Specifically if the previous cell is valid and the next point is still in it (i.e., vtkCell::EvaluatePosition() returns 1, coupled with

newly created parametric coordinates & weights), the function values can be interpolated and only vtkCell::EvaluatePosition() is invoked. If this fails, then level #1 follows by inter-cell search for the target cell that contains the next point. By an inter-cell search, the previous cell provides an important clue or serves as an immediate neighbor to aid in locating the target cell via vtkPointSet:: FindCell(). If this still fails, a global cell location / search is invoked via vtkPointSet::FindCell(). Here regardless of either inter-cell or global search, vtkPointLocator is in fact employed (for datasets of type vtkPointSet only, note vtkImageData and vtkRectilinearGrid are able to provide rapid and robust cell location due to the simple mesh topology) as a crucial tool underlying the cell locator. However, the use of vtkPointLocator makes vtkInterpolatedVelocityField non-robust in cell location for vtkPointSet.

For vtkCellLocatorInterpolatedVelocityField, the only caching (level #0) works by intra-cell trial. In case of failure, a global search for the target cell is invoked via vtkAbstractCellLocator::FindCell() and the actual work is done by either vtkCellLocator or vtkModifiedBSPTree (for datasets of type vtkPointSet only, while vtkImageData and vtkRectilinearGrid themselves are able to provide fast robust cell location). Without the involvement of vtkPointLocator, robust cell location is achieved for vtkPointSet.

To create an instance of class vtkAbstractInterpolatedVelocityField, simply invoke its constructor as follows

```
obj = vtkAbstractInterpolatedVelocityField
```

31.2.2 Methods

The class vtkAbstractInterpolatedVelocityField has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractInterpolatedVelocityField class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractInterpolatedVelocityField = obj.NewInstance ()`
- `vtkAbstractInterpolatedVelocityField = obj.SafeDownCast (vtkObject o)`
- `obj.SetCaching (bool )` - Set/Get the caching flag. If this flag is turned ON, there are two levels of caching for derived concrete class vtkInterpolatedVelocityField and one level of caching for derived concrete class vtkCellLocatorInterpolatedVelocityField. Otherwise a global cell location is always invoked for evaluating the function values at any point.
- `bool = obj.GetCaching ()` - Set/Get the caching flag. If this flag is turned ON, there are two levels of caching for derived concrete class vtkInterpolatedVelocityField and one level of caching for derived concrete class vtkCellLocatorInterpolatedVelocityField. Otherwise a global cell location is always invoked for evaluating the function values at any point.
- `int = obj.GetCacheHit ()` - Get the caching statistics. CacheHit refers to the number of level #0 cache hits while CacheMiss is the number of level #0 cache misses.
- `int = obj.GetCacheMiss ()` - Get the caching statistics. CacheHit refers to the number of level #0 cache hits while CacheMiss is the number of level #0 cache misses.
- `int = obj.GetLastDataSetIndex ()` - Get the most recently visited dataset and it id. The dataset is used for a guess regarding where the next point will be, without searching through all datasets. When setting the last dataset, care is needed as no reference counting or checks are performed. This feature is intended for custom interpolators only that cache datasets independently.
- `vtkDataSet = obj.GetLastDataSet ()` - Get the most recently visited dataset and it id. The dataset is used for a guess regarding where the next point will be, without searching through all datasets. When setting the last dataset, care is needed as no reference counting or checks are performed. This feature is intended for custom interpolators only that cache datasets independently.

- `vtkIdType = obj.GetLastCellId ()` - Get/Set the id of the cell cached from last evaluation.
- `obj.SetLastCellId (vtkIdType c)` - Set the id of the most recently visited cell of a dataset.
- `obj.SetLastCellId (vtkIdType c, int dataindex)` - Set the id of the most recently visited cell of a dataset.
- `string = obj.GetVectorsSelection ()` - Get/Set the name of a spcified vector array. By default it is NULL, with the active vector array for use.
- `obj.SelectVectors (string fieldName)` - Set/Get the flag indicating vector post-normalization (following vector interpolation). Vector post-normalization is required to avoid the 'curve-overshooting' problem (caused by high velocity magnitude) that occurs when Cell-Length is used as the step size unit (particularly the Minimum step size unit). Furthermore, it is required by RK45 to achieve, as expected, high numerical accuracy (or high smoothness of flow lines) through adaptive step sizing. Note this operation is performed (when NormalizeVector TRUE) right after vector interpolation such that the differing amount of contribution of each node (of a cell) to the resulting direction of the interpolated vector, due to the possibly significantly-differing velocity magnitude values at the nodes (which is the case with large cells), can be reflected as is. Also note that this flag needs to be turned to FALSE after vtkInitialValueProblemSolver:: ComputeNextStep() as subsequent operations, e.g., vorticity computation, may need non-normalized vectors.
- `obj.SetNormalizeVector (bool )` - Set/Get the flag indicating vector post-normalization (following vector interpolation). Vector post-normalization is required to avoid the 'curve-overshooting' problem (caused by high velocity magnitude) that occurs when Cell-Length is used as the step size unit (particularly the Minimum step size unit). Furthermore, it is required by RK45 to achieve, as expected, high numerical accuracy (or high smoothness of flow lines) through adaptive step sizing. Note this operation is performed (when NormalizeVector TRUE) right after vector interpolation such that the differing amount of contribution of each node (of a cell) to the resulting direction of the interpolated vector, due to the possibly significantly-differing velocity magnitude values at the nodes (which is the case with large cells), can be reflected as is. Also note that this flag needs to be turned to FALSE after vtkInitialValueProblemSolver:: ComputeNextStep() as subsequent operations, e.g., vorticity computation, may need non-normalized vectors.
- `bool = obj.GetNormalizeVector ()` - Set/Get the flag indicating vector post-normalization (following vector interpolation). Vector post-normalization is required to avoid the 'curve-overshooting' problem (caused by high velocity magnitude) that occurs when Cell-Length is used as the step size unit (particularly the Minimum step size unit). Furthermore, it is required by RK45 to achieve, as expected, high numerical accuracy (or high smoothness of flow lines) through adaptive step sizing. Note this operation is performed (when NormalizeVector TRUE) right after vector interpolation such that the differing amount of contribution of each node (of a cell) to the resulting direction of the interpolated vector, due to the possibly significantly-differing velocity magnitude values at the nodes (which is the case with large cells), can be reflected as is. Also note that this flag needs to be turned to FALSE after vtkInitialValueProblemSolver:: ComputeNextStep() as subsequent operations, e.g., vorticity computation, may need non-normalized vectors.
- `obj.CopyParameters (vtkAbstractInterpolatedVelocityField from)` - Add a dataset for implicit velocity function evaluation. If more than one dataset is added, the evaluation point is searched in all until a match is found. THIS FUNCTION DOES NOT CHANGE THE REFERENCE COUNT OF dataset FOR THREAD SAFETY REASONS.
- `obj.AddDataSet (vtkDataSet dataset)` - Add a dataset for implicit velocity function evaluation. If more than one dataset is added, the evaluation point is searched in all until a match is found. THIS FUNCTION DOES NOT CHANGE THE REFERENCE COUNT OF dataset FOR THREAD SAFETY REASONS.
- `int = obj.FunctionValues (double x, double f)` - Evaluate the velocity field f at point (x, y, z).

- `obj.ClearLastCellId ()` - Get the interpolation weights cached from last evaluation. Return 1 if the cached cell is valid and 0 otherwise.
- `int = obj.GetLastWeights (double w)` - Get the interpolation weights cached from last evaluation. Return 1 if the cached cell is valid and 0 otherwise.
- `int = obj.GetLastLocalCoordinates (double pcoords[3])` - Get the interpolation weights cached from last evaluation. Return 1 if the cached cell is valid and 0 otherwise.

31.3 vtkAbstractMapper

31.3.1 Usage

vtkAbstractMapper is an abstract class to specify interface between data and graphics primitives or software rendering techniques. Subclasses of vtkAbstractMapper can be used for rendering 2D data, geometry, or volumetric data.

To create an instance of class vtkAbstractMapper, simply invoke its constructor as follows

```
obj = vtkAbstractMapper
```

31.3.2 Methods

The class vtkAbstractMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractMapper = obj.NewInstance ()`
- `vtkAbstractMapper = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Override Modifiedtime as we have added Clipping planes
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this mapper. The parameter window could be used to determine which graphic resources to release.
- `double = obj.GetTimeToDraw ()` - Get the time required to draw the geometry last time it was rendered
- `obj.AddClippingPlane (vtkPlane plane)` - Specify clipping planes to be applied when the data is mapped (at most 6 clipping planes can be specified).
- `obj.RemoveClippingPlane (vtkPlane plane)` - Specify clipping planes to be applied when the data is mapped (at most 6 clipping planes can be specified).
- `obj.RemoveAllClippingPlanes ()` - Specify clipping planes to be applied when the data is mapped (at most 6 clipping planes can be specified).
- `obj.SetClippingPlanes (vtkPlaneCollection )` - Get/Set the vtkPlaneCollection which specifies the clipping planes.
- `vtkPlaneCollection = obj.GetClippingPlanes ()` - Get/Set the vtkPlaneCollection which specifies the clipping planes.
- `obj.SetClippingPlanes (vtkPlanes planes)` - An alternative way to set clipping planes: use up to six planes found in the supplied instance of the implicit function vtkPlanes.
- `obj.ShallowCopy (vtkAbstractMapper m)` - Make a shallow copy of this mapper.

31.4 vtkAbstractPointLocator

31.4.1 Usage

vtkAbstractPointLocator is an abstract spatial search object to quickly locate points in 3D. vtkAbstractPointLocator works by dividing a specified region of space into "rectangular" buckets, and then keeping a list of points that lie in each bucket. Typical operation involves giving a position in 3D and finding the closest point. The points are provided from the specified dataset input.

To create an instance of class vtkAbstractPointLocator, simply invoke its constructor as follows

```
obj = vtkAbstractPointLocator
```

31.4.2 Methods

The class vtkAbstractPointLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractPointLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractPointLocator = obj.NewInstance ()`
- `vtkAbstractPointLocator = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.FindClosestPoint (double x[3])` - Given a position x, return the id of the point closest to it. Alternative method requires separate x-y-z values. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `vtkIdType = obj.FindClosestPoint (double x, double y, double z)` - Given a position x, return the id of the point closest to it. Alternative method requires separate x-y-z values. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.FindClosestNPoints (int N, double x[3], vtkIdList result)` - Find the closest N points to a position. This returns the closest N points to a position. A faster method could be created that returned N close points to a position, but necessarily the exact N closest. The returned points are sorted from closest to farthest. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.FindClosestNPoints (int N, double x, double y, double z, vtkIdList result)` - Find the closest N points to a position. This returns the closest N points to a position. A faster method could be created that returned N close points to a position, but necessarily the exact N closest. The returned points are sorted from closest to farthest. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.FindPointsWithinRadius (double R, double x[3], vtkIdList result)` - Find all points within a specified radius R of position x. The result is not sorted in any specific manner. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.FindPointsWithinRadius (double R, double x, double y, double z, vtkIdList result)` - Find all points within a specified radius R of position x. The result is not sorted in any specific manner. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.GetBounds (double )` - Provide an accessor to the bounds.
- `obj.FreeSearchStructure ()` - See vtkLocator interface documentation. These methods are not thread safe.

- `obj.BuildLocator ()` - See vtkLocator interface documentation. These methods are not thread safe.
- `obj.GenerateRepresentation (int level, vtkPolyData pd)` - See vtkLocator interface documentation. These methods are not thread safe.

31.5 vtkActor2D

31.5.1 Usage

vtkActor2D is similar to vtkActor, but it is made to be used with two dimensional images and annotation. vtkActor2D has a position but does not use a transformation matrix like vtkActor (see the superclass vtkProp for information on positioning vtkActor2D). vtkActor2D has a reference to a vtkMapper2D object which does the rendering.

To create an instance of class vtkActor2D, simply invoke its constructor as follows

```
obj = vtkActor2D
```

31.5.2 Methods

The class vtkActor2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkActor2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkActor2D = obj.NewInstance ()`
- `vtkActor2D = obj.SafeDownCast (vtkObject o)`
- `int = obj.RenderOverlay (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.SetMapper (vtkMapper2D mapper)` - Set/Get the vtkMapper2D which defines the data to be drawn.
- `vtkMapper2D = obj.GetMapper ()` - Set/Get the vtkMapper2D which defines the data to be drawn.
- `obj.SetLayerNumber (int )` - Set/Get the layer number in the overlay planes into which to render.
- `int = obj.GetLayerNumber ()` - Set/Get the layer number in the overlay planes into which to render.
- `vtkProperty2D = obj.GetProperty ()` - Returns this actor's vtkProperty2D. Creates a property if one doesn't already exist.
- `obj.SetProperty (vtkProperty2D )` - Set this vtkProp's vtkProperty2D.
- `vtkCoordinate = obj.GetPositionCoordinate ()` - Get the PositionCoordinate instance of vtkCoordinate. This is used for for complicated or relative positioning. The position variable controls the lower left corner of the Actor2D

- `obj.SetPosition (double, double)` - Get the PositionCoordinate instance of vtkCoordinate. This is used for for complicated or relative positioning. The position variable controls the lower left corner of the Actor2D
- `obj.SetPosition (double a[2])` - Get the PositionCoordinate instance of vtkCoordinate. This is used for for complicated or relative positioning. The position variable controls the lower left corner of the Actor2D
- `double = obj.GetPosition ()` - Get the PositionCoordinate instance of vtkCoordinate. This is used for for complicated or relative positioning. The position variable controls the lower left corner of the Actor2D
- `obj.SetDisplayPosition (int , int )` - Set the Prop2D's position in display coordinates.
- `vtkCoordinate = obj.GetPosition2Coordinate ()` - Access the Position2 instance variable. This variable controls the upper right corner of the Actor2D. It is by default relative to Position and in normalized viewport coordinates. Some 2D actor subclasses ignore the position2 variable
- `obj.SetPosition2 (double, double)` - Access the Position2 instance variable. This variable controls the upper right corner of the Actor2D. It is by default relative to Position and in normalized viewport coordinates. Some 2D actor subclasses ignore the position2 variable
- `obj.SetPosition2 (double a[2])` - Access the Position2 instance variable. This variable controls the upper right corner of the Actor2D. It is by default relative to Position and in normalized viewport coordinates. Some 2D actor subclasses ignore the position2 variable
- `double = obj.GetPosition2 ()` - Access the Position2 instance variable. This variable controls the upper right corner of the Actor2D. It is by default relative to Position and in normalized viewport coordinates. Some 2D actor subclasses ignore the position2 variable
- `obj.SetWidth (double w)` - Set/Get the height and width of the Actor2D. The value is expressed as a fraction of the viewport. This really is just another way of setting the Position2 instance variable.
- `double = obj.GetWidth ()` - Set/Get the height and width of the Actor2D. The value is expressed as a fraction of the viewport. This really is just another way of setting the Position2 instance variable.
- `obj.SetHeight (double h)` - Set/Get the height and width of the Actor2D. The value is expressed as a fraction of the viewport. This really is just another way of setting the Position2 instance variable.
- `double = obj.GetHeight ()` - Set/Get the height and width of the Actor2D. The value is expressed as a fraction of the viewport. This really is just another way of setting the Position2 instance variable.
- `long = obj.GetMTime ()` - Return this objects MTime.
- `obj.GetActors2D (vtkPropCollection pc)` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this vtkActor2D. Overloads the virtual vtkProp method.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `vtkCoordinate = obj.GetActualPositionCoordinate (void )` - Return the actual vtkCoordinate reference that the mapper should use to position the actor. This is used internally by the mappers and should be overridden in specialized subclasses and otherwise ignored.
- `vtkCoordinate = obj.GetActualPosition2Coordinate (void )`

31.6 vtkActor2DCollection

31.6.1 Usage

vtkActor2DCollection is a subclass of vtkCollection. vtkActor2DCollection maintains a collection of vtkActor2D objects that is sorted by layer number, with lower layer numbers at the start of the list. This allows the vtkActor2D objects to be rendered in the correct order.

To create an instance of class vtkActor2DCollection, simply invoke its constructor as follows

```
obj = vtkActor2DCollection
```

31.6.2 Methods

The class vtkActor2DCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkActor2DCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkActor2DCollection = obj.NewInstance ()`
- `vtkActor2DCollection = obj.SafeDownCast (vtkObject o)`
- `obj.Sort ()` - Sorts the vtkActor2DCollection by layer number. Smaller layer numbers are first. Layer numbers can be any integer value.
- `obj.AddItem (vtkActor2D a)` - Add an actor to the list. The new actor is inserted in the list according to it's layer number.
- `int = obj.IsItemPresent (vtkActor2D a)` - Standard Collection methods
- `vtkActor2D = obj.GetNextActor2D ()` - Standard Collection methods
- `vtkActor2D = obj.GetLastActor2D ()` - Standard Collection methods
- `vtkActor2D = obj.GetNextItem ()` - Access routines that are provided for compatibility with previous version of VTK. Please use the GetNextActor2D(), GetLastActor2D() variants where possible.
- `vtkActor2D = obj.GetLastItem ()` - Access routines that are provided for compatibility with previous version of VTK. Please use the GetNextActor2D(), GetLastActor2D() variants where possible.
- `obj.RenderOverlay (vtkViewport viewport)` - Sort and then render the collection of 2D actors.

31.7 vtkAdjacentVertexIterator

31.7.1 Usage

vtkAdjacentVertexIterator iterates through all vertices adjacent to a vertex, i.e. the vertices which may be reached by traversing an out edge of the source vertex. Use graph-¿GetAdjacentVertices(v, it) to initialize the iterator.

To create an instance of class vtkAdjacentVertexIterator, simply invoke its constructor as follows

```
obj = vtkAdjacentVertexIterator
```

31.7.2 Methods

The class vtkAdjacentVertexIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAdjacentVertexIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAdjacentVertexIterator = obj.NewInstance ()`
- `vtkAdjacentVertexIterator = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (vtkGraph g, vtkIdType v)` - Initialize the iterator with a graph and vertex.
- `vtkGraph = obj.GetGraph ()` - Get the graph and vertex associated with this iterator.
- `vtkIdType = obj.GetVertex ()` - Get the graph and vertex associated with this iterator.
- `vtkIdType = obj.Next ()` - Whether this iterator has more edges.
- `bool = obj.HasNext ()`

31.8 vtkAlgorithm

31.8.1 Usage

vtkAlgorithm is the superclass for all sources, filters, and sinks in VTK. It defines a generalized interface for executing data processing algorithms. Pipeline connections are associated with input and output ports that are independent of the type of data passing through the connections.

Instances may be used independently or within pipelines with a variety of architectures and update mechanisms. Pipelines are controlled by instances of vtkExecutive. Every vtkAlgorithm instance has an associated vtkExecutive when it is used in a pipeline. The executive is responsible for data flow.

To create an instance of class vtkAlgorithm, simply invoke its constructor as follows

```
obj = vtkAlgorithm
```

31.8.2 Methods

The class vtkAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAlgorithm = obj.NewInstance ()`
- `vtkAlgorithm = obj.SafeDownCast (vtkObject o)`
- `int = obj.HasExecutive ()` - Check whether this algorithm has an assigned executive. This will NOT create a default executive.
- `vtkExecutive = obj.GetExecutive ()` - Get this algorithm's executive. If it has none, a default executive will be created.

- `obj.SetExecutive (vtkExecutive executive)` - Set this algorithm's executive. This algorithm is removed from any executive to which it has previously been assigned and then assigned to the given executive.
- `int = obj.ModifyRequest (vtkInformation request, int when)` - This method gives the algorithm a chance to modify the contents of a request before or after (specified in the when argument) it is forwarded. The default implementation is empty. Returns 1 on success, 0 on failure. When can be either vtkExecutive::BeforeForward or vtkExecutive::AfterForward.
- `vtkInformation = obj.GetInputPortInformation (int port)` - Get the information object associated with an input port. There is one input port per kind of input to the algorithm. Each input port tells executives what kind of data and downstream requests this algorithm can handle for that input.
- `vtkInformation = obj.GetOutputPortInformation (int port)` - Get the information object associated with an output port. There is one output port per output from the algorithm. Each output port tells executives what kind of upstream requests this algorithm can handle for that output.
- `vtkInformation = obj.GetInformation ()` - Set/Get the information object associated with this algorithm.
- `obj.SetInformation (vtkInformation )` - Set/Get the information object associated with this algorithm.
- `int = obj.GetNumberOfInputPorts ()` - Get the number of input ports used by the algorithm.
- `int = obj.GetNumberOfOutputPorts ()` - Get the number of output ports provided by the algorithm.
- `obj.Register (vtkObjectBase o)` - Participate in garbage collection.
- `obj.UnRegister (vtkObjectBase o)` - Participate in garbage collection.
- `obj.SetAbortExecute (int )` - Set/Get the AbortExecute flag for the process object. Process objects may handle premature termination of execution in different ways.
- `int = obj.GetAbortExecute ()` - Set/Get the AbortExecute flag for the process object. Process objects may handle premature termination of execution in different ways.
- `obj.AbortExecuteOn ()` - Set/Get the AbortExecute flag for the process object. Process objects may handle premature termination of execution in different ways.
- `obj.AbortExecuteOff ()` - Set/Get the AbortExecute flag for the process object. Process objects may handle premature termination of execution in different ways.
- `obj.SetProgress (double )` - Set/Get the execution progress of a process object.
- `double = obj.GetProgressMinValue ()` - Set/Get the execution progress of a process object.
- `double = obj.GetProgressMaxValue ()` - Set/Get the execution progress of a process object.
- `double = obj.GetProgress ()` - Set/Get the execution progress of a process object.
- `obj.UpdateProgress (double amount)` - Update the progress of the process object. If a ProgressMethod exists, executes it. Then set the Progress ivar to amount. The parameter amount should range between (0,1).
- `obj.SetProgressText (string ptext)` - Set the current text message associated with the progress state. This may be used by a calling process/GUI. Note: Because SetProgressText() is called from inside RequestData() it does not modify the algorithm object. Algorithms are not allowed to modify themselves from inside RequestData().

- `string = obj.GetProgressText ()` - Set the current text message associated with the progress state. This may be used by a calling process/GUI. Note: Because SetProgressText() is called from inside RequestData() it does not modify the algorithm object. Algorithms are not allowed to modify themselves from inside RequestData().

- `long = obj.GetErrorCode ()` - The error code contains a possible error that occured while reading or writing the file.

- `obj.SetInputArrayToProcess (int idx, int port, int connection, int fieldAssociation, string name)` - Set the input data arrays that this algorithm will process. Specifically the idx array that this algorithm will process (starting from 0) is the array on port, connection with the specified association and name or attribute type (such as SCALARS). The fieldAssociation refers to which field in the data object the array is stored. See vtkDataObject::FieldAssociations for detail.

- `obj.SetInputArrayToProcess (int idx, int port, int connection, int fieldAssociation, int fieldAttri` - Set the input data arrays that this algorithm will process. Specifically the idx array that this algorithm will process (starting from 0) is the array on port, connection with the specified association and name or attribute type (such as SCALARS). The fieldAssociation refers to which field in the data object the array is stored. See vtkDataObject::FieldAssociations for detail.

- `obj.SetInputArrayToProcess (int idx, vtkInformation info)` - Set the input data arrays that this algorithm will process. Specifically the idx array that this algorithm will process (starting from 0) is the array on port, connection with the specified association and name or attribute type (such as SCALARS). The fieldAssociation refers to which field in the data object the array is stored. See vtkDataObject::FieldAssociations for detail.

- `obj.SetInputArrayToProcess (int idx, int port, int connection, string fieldAssociation, string attr` - String based versions of SetInputArrayToProcess(). Because fieldAssociation and fieldAttributeType are enums, they cannot be easily accessed from scripting language. These methods provides an easy and safe way of passing association and attribute type information. Field association is one of the following: @verbatim vtkDataObject::FIELD_ASSOCIATION_POINTS vtkDataObject::FIELD_ASSOCIATION_CELLS vtkDataObject::FIELD_ASSOCIATION_NONE vtkDataObject::FIELD_ASSOCIATION_POINTS_THEN_CELLS @endverbatim Attribute type is one of the following: @verbatim vtkDataSetAttributes::SCALARS vtkDataSetAttributes::VECTORS vtkDataSetAttributes::NORMALS vtkDataSetAttributes::TCOORDS vtkDataSetAttributes::TENSORS @endverbatim If the last argument is not an attribute type, it is assumed to be an array name.

- `vtkInformation = obj.GetInputArrayInformation (int idx)` - Get the info object for the specified input array to this algorithm

- `obj.RemoveAllInputs ()` - Remove all the input data.

- `vtkDataObject = obj.GetOutputDataObject (int port)` - Get the data object that will contain the algorithm output for the given port.

- `vtkDataObject = obj.GetInputDataObject (int port, int connection)` - Get the data object that will contain the algorithm input for the given port and given connection.

- `obj.SetInputConnection (int port, vtkAlgorithmOutput input)` - Set the connection for the given input port index. Each input port of a filter has a specific purpose. A port may have zero or more connections and the required number is specified by each filter. Setting the connection with this method removes all other connections from the port. To add more than one connection use AddInputConnection().

  The input for the connection is the output port of another filter, which is obtained with GetOutputPort(). Typical usage is

  filter2-¿SetInputConnection(0, filter1-¿GetOutputPort(0)).

- `obj.SetInputConnection (vtkAlgorithmOutput input)` - Set the connection for the given input port index. Each input port of a filter has a specific purpose. A port may have zero or more connections and the required number is specified by each filter. Setting the connection with this method removes all other connections from the port. To add more than one connection use AddInputConnection().

  The input for the connection is the output port of another filter, which is obtained with GetOutputPort(). Typical usage is

  filter2-¿SetInputConnection(0, filter1-¿GetOutputPort(0)).

- `obj.AddInputConnection (int port, vtkAlgorithmOutput input)` - Add a connection to the given input port index. See SetInputConnection() for details on input connections. This method is the complement to RemoveInputConnection() in that it adds only the connection specified without affecting other connections. Typical usage is

  filter2-¿AddInputConnection(0, filter1-¿GetOutputPort(0)).

- `obj.AddInputConnection (vtkAlgorithmOutput input)` - Add a connection to the given input port index. See SetInputConnection() for details on input connections. This method is the complement to RemoveInputConnection() in that it adds only the connection specified without affecting other connections. Typical usage is

  filter2-¿AddInputConnection(0, filter1-¿GetOutputPort(0)).

- `obj.RemoveInputConnection (int port, vtkAlgorithmOutput input)` - Remove a connection from the given input port index. See SetInputConnection() for details on input connection. This method is the complement to AddInputConnection() in that it removes only the connection specified without affecting other connections. Typical usage is

  filter2-¿RemoveInputConnection(0, filter1-¿GetOutputPort(0)).

- `vtkAlgorithmOutput = obj.GetOutputPort (int index)` - Get a proxy object corresponding to the given output port of this algorithm. The proxy object can be passed to another algorithm's SetInputConnection(), AddInputConnection(), and RemoveInputConnection() methods to modify pipeline connectivity.

- `vtkAlgorithmOutput = obj.GetOutputPort ()` - Get the number of inputs currently connected to a port.

- `int = obj.GetNumberOfInputConnections (int port)` - Get the number of inputs currently connected to a port.

- `int = obj.GetTotalNumberOfInputConnections ()` - Get the total number of inputs for this algorithm

- `vtkAlgorithmOutput = obj.GetInputConnection (int port, int index)` - Get the algorithm output port connected to an input port.

- `obj.Update ()` - Bring this algorithm's outputs up-to-date.

- `obj.UpdateInformation ()` - Backward compatibility method to invoke UpdateInformation on executive.

- `obj.UpdateWholeExtent ()` - Bring this algorithm's outputs up-to-date.

- `obj.SetReleaseDataFlag (int )` - Turn release data flag on or off for all output ports.

- `int = obj.GetReleaseDataFlag ()` - Turn release data flag on or off for all output ports.

- `obj.ReleaseDataFlagOn ()` - Turn release data flag on or off for all output ports.

- `obj.ReleaseDataFlagOff ()` - Turn release data flag on or off for all output ports.

- `int = obj.UpdateExtentIsEmpty (vtkDataObject output)` - This detects when the UpdateExtent will generate no data This condition is satisfied when the UpdateExtent has zero volume (0,-1,...) or the UpdateNumberOfPieces is 0. The source uses this call to determine whether to call Execute.
- `int = obj.UpdateExtentIsEmpty (vtkInformation pinfo, int extentType)` - This detects when the UpdateExtent will generate no data This condition is satisfied when the UpdateExtent has zero volume (0,-1,...) or the UpdateNumberOfPieces is 0. The source uses this call to determine whether to call Execute.
- `double = obj.ComputePriority ()` - Returns the priority of the piece described by the current update extent. The priority is a number between 0.0 and 1.0 with 0 meaning skippable (REQUEST_DATA not needed) and 1.0 meaning important.

31.9 vtkAlgorithmOutput

31.9.1 Usage

vtkAlgorithmOutput is a proxy object returned by the GetOutputPort method of vtkAlgorithm. It may be passed to the SetInputConnection, AddInputConnection, or RemoveInputConnection methods of another vtkAlgorithm to establish a connection between an output and input port. The connection is not stored in the proxy object: it is simply a convenience for creating or removing connections.

To create an instance of class vtkAlgorithmOutput, simply invoke its constructor as follows

```
obj = vtkAlgorithmOutput
```

31.9.2 Methods

The class vtkAlgorithmOutput has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAlgorithmOutput class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAlgorithmOutput = obj.NewInstance ()`
- `vtkAlgorithmOutput = obj.SafeDownCast (vtkObject o)`
- `obj.SetIndex (int index)`
- `int = obj.GetIndex ()`
- `vtkAlgorithm = obj.GetProducer ()`
- `obj.SetProducer (vtkAlgorithm producer)`

31.10 vtkAnnotation

31.10.1 Usage

vtkAnnotation is a collection of annotation properties along with an associated selection indicating the portion of data the annotation refers to.

.SECTION Thanks Timothy M. Shead (tshead@sandia.gov) at Sandia National Laboratories contributed code to this class.

To create an instance of class vtkAnnotation, simply invoke its constructor as follows

```
obj = vtkAnnotation
```

31.10.2 Methods

The class vtkAnnotation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAnnotation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAnnotation = obj.NewInstance ()`
- `vtkAnnotation = obj.SafeDownCast (vtkObject o)`
- `vtkSelection = obj.GetSelection ()` - The selection to which this set of annotations will apply.
- `obj.SetSelection (vtkSelection selection)` - The selection to which this set of annotations will apply.
- `obj.Initialize ()` - Initialize the annotation to an empty state.
- `obj.ShallowCopy (vtkDataObject other)` - Make this annotation have the same properties and have the same selection of another annotation.
- `obj.DeepCopy (vtkDataObject other)` - Make this annotation have the same properties and have a copy of the selection of another annotation.
- `long = obj.GetMTime ()` - Get the modified time of this object.

31.11 vtkAnnotationLayers

31.11.1 Usage

vtkAnnotationLayers stores a vector of annotation layers. Each layer may contain any number of vtkAnnotation objects. The ordering of the layers introduces a prioritization of annotations. Annotations in higher layers may obscure annotations in lower layers.

To create an instance of class vtkAnnotationLayers, simply invoke its constructor as follows

```
obj = vtkAnnotationLayers
```

31.11.2 Methods

The class vtkAnnotationLayers has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAnnotationLayers class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAnnotationLayers = obj.NewInstance ()`
- `vtkAnnotationLayers = obj.SafeDownCast (vtkObject o)`
- `obj.SetCurrentAnnotation (vtkAnnotation ann)` - The current annotation associated with this annotation link.
- `vtkAnnotation = obj.GetCurrentAnnotation ()` - The current annotation associated with this annotation link.

- `obj.SetCurrentSelection (vtkSelection sel)` - The current selection associated with this annotation link. This is simply the selection contained in the current annotation.
- `vtkSelection = obj.GetCurrentSelection ()` - The current selection associated with this annotation link. This is simply the selection contained in the current annotation.
- `int = obj.GetNumberOfAnnotations ()` - The number of annotations in a specific layer.
- `vtkAnnotation = obj.GetAnnotation (int idx)` - Retrieve an annotation from a layer.
- `obj.AddAnnotation (vtkAnnotation ann)` - Add an annotation to a layer.
- `obj.RemoveAnnotation (vtkAnnotation ann)` - Remove an annotation from a layer.
- `obj.Initialize ()` - Initialize the data structure to an empty state.
- `obj.ShallowCopy (vtkDataObject other)` - Copy data from another data object into this one which references the same member annotations.
- `obj.DeepCopy (vtkDataObject other)` - Copy data from another data object into this one, performing a deep copy of member annotations.
- `long = obj.GetMTime ()` - The modified time for this object.

31.12 vtkAnnotationLayersAlgorithm

31.12.1 Usage

vtkAnnotationLayersAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline architecture. There are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this class constructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be vtkAnnotationLayers. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into separate functions such as ExecuteData and ExecuteInformation. For new algorithms you should implement RequestData(request, inputVec, outputVec) but for older filters there is a default implementation that calls the old ExecuteData(output) signature. For even older filters that don't implement ExecuteData the default implementation calls the even older Execute() signature.

To create an instance of class vtkAnnotationLayersAlgorithm, simply invoke its constructor as follows

```
obj = vtkAnnotationLayersAlgorithm
```

31.12.2 Methods

The class vtkAnnotationLayersAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAnnotationLayersAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAnnotationLayersAlgorithm = obj.NewInstance ()`
- `vtkAnnotationLayersAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkAnnotationLayers = obj.GetOutput ()` - Get the output data object for a port on this algorithm.

- `vtkAnnotationLayers = obj.GetOutput (int index)` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int index, vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.13 vtkArrayData

31.13.1 Usage

Because vtkArray cannot be stored as attributes of data objects (yet), a "carrier" object is needed to pass vtkArray through the pipeline. vtkArrayData acts as a container of zero-to-many vtkArray instances, which can be retrieved via a zero-based index. Note that a collection of arrays stored in vtkArrayData may-or-may-not have related types, dimensions, or extents.

To create an instance of class vtkArrayData, simply invoke its constructor as follows

```
obj = vtkArrayData
```

31.13.2 Methods

The class vtkArrayData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArrayData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArrayData = obj.NewInstance ()`
- `vtkArrayData = obj.SafeDownCast (vtkObject o)`
- `obj.AddArray (vtkArray )` - Adds a vtkArray to the collection
- `obj.ClearArrays ()` - Clears the contents of the collection
- `vtkIdType = obj.GetNumberOfArrays ()` - Returns the number of vtkArray instances in the collection
- `vtkArray = obj.GetArray (vtkIdType index)` - Returns the n-th vtkArray in the collection
- `vtkArray = obj.GetArrayByName (string name)` - Returns the array having called name from the collection
- `int = obj.GetDataObjectType ()`
- `obj.ShallowCopy (vtkDataObject other)`
- `obj.DeepCopy (vtkDataObject other)`

31.14 vtkArrayDataAlgorithm

31.14.1 Usage

vtkArrayDataAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline architecture. There are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this class constructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be vtkArrayData. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into separate functions such as ExecuteData and ExecuteInformation. For new algorithms you should implement RequestData(request, inputVec, outputVec) but for older filters there is a default implementation that calls the old ExecuteData(output) signature. For even older filters that don't implement ExecuteData the default implementation calls the even older Execute() signature.

To create an instance of class vtkArrayDataAlgorithm, simply invoke its constructor as follows

```
obj = vtkArrayDataAlgorithm
```

31.14.2 Methods

The class vtkArrayDataAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArrayDataAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArrayDataAlgorithm = obj.NewInstance ()`
- `vtkArrayDataAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkArrayData = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkArrayData = obj.GetOutput (int index)` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int index, vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.15 vtkAttributesErrorMetric

31.15.1 Usage

It is a concrete error metric, based on an attribute criterium: the variation of the active attribute/component value from a linear ramp

To create an instance of class vtkAttributesErrorMetric, simply invoke its constructor as follows

```
obj = vtkAttributesErrorMetric
```

31.15.2 Methods

The class vtkAttributesErrorMetric has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAttributesErrorMetric class.

- `string = obj.GetClassName ()` - Standard VTK type and error macros.
- `int = obj.IsA (string name)` - Standard VTK type and error macros.
- `vtkAttributesErrorMetric = obj.NewInstance ()` - Standard VTK type and error macros.
- `vtkAttributesErrorMetric = obj.SafeDownCast (vtkObject o)` - Standard VTK type and error macros.
- `double = obj.GetAbsoluteAttributeTolerance ()` - Absolute tolerance of the active scalar (attribute+component). Subdivision is required if the square distance between the real attribute at the mid point on the edge and the interpolated attribute is greater than AbsoluteAttributeTolerance. This is the attribute accuracy. 0.01 will give better result than 0.1.
- `obj.SetAbsoluteAttributeTolerance (double value)` - Set the absolute attribute accuracy to 'value'. See GetAbsoluteAttributeTolerance() for details. It is particularly useful when some concrete implementation of vtkGenericAttribute does not support GetRange() request, called internally in SetAttributeTolerance(). It may happen when the implementation support higher order attributes but cannot compute the range.
- `double = obj.GetAttributeTolerance ()` - Relative tolerance of the active scalar (attribute+component). Subdivision is required if the square distance between the real attribute at the mid point on the edge and the interpolated attribute is greater than AttributeTolerance. This is the attribute accuracy. 0.01 will give better result than 0.1.
- `obj.SetAttributeTolerance (double value)` - Set the relative attribute accuracy to 'value'. See GetAttributeTolerance() for details.
- `int = obj.RequiresEdgeSubdivision (double leftPoint, double midPoint, double rightPoint, double alp` - Does the edge need to be subdivided according to the distance between the value of the active attribute/component at the midpoint and the mean value between the endpoints? The edge is defined by its 'leftPoint' and its 'rightPoint'. 'leftPoint', 'midPoint' and 'rightPoint' have to be initialized before calling RequiresEdgeSubdivision(). Their format is global coordinates, parametric coordinates and point centered attributes: xyx rst abc de... 'alpha' is the normalized abscissa of the midpoint along the edge. (close to 0 means close to the left point, close to 1 means close to the right point)

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6
- `double = obj.GetError (double leftPoint, double midPoint, double rightPoint, double alpha)` - Return the error at the mid-point. The type of error depends on the state of the concrete error metric. For instance, it can return an absolute or relative error metric. See RequiresEdgeSubdivision() for a description of the arguments.

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6

31.16 vtkBiQuadraticQuad

31.16.1 Usage

vtkQuadraticQuad is a concrete implementation of vtkNonLinearCell to represent a two-dimensional, 9-node isoparametric parabolic quadrilateral element with a Centerpoint. The interpolation is the standard finite element, quadratic isoparametric shape function. The cell includes a mid-edge node for each of the four

edges of the cell and a center node at the surface. The ordering of the eight points defining the cell are point ids (0-3,4-8) where ids 0-3 define the four corner vertices of the quad; ids 4-7 define the midedge nodes (0,1), (1,2), (2,3), (3,0) and 8 define the face center node.

To create an instance of class vtkBiQuadraticQuad, simply invoke its constructor as follows

```
obj = vtkBiQuadraticQuad
```

31.16.2 Methods

The class vtkBiQuadraticQuad has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBiQuadraticQuad class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBiQuadraticQuad = obj.NewInstance ()`
- `vtkBiQuadraticQuad = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )`
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this biquadratic quad using scalar value provided. Like contouring, except that it cuts the twi quads to produce linear triangles.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the pyramid in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[9])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[18])`

31.17 vtkBiQuadraticQuadraticHexahedron

31.17.1 Usage

vtkBiQuadraticQuadraticHexahedron is a concrete implementation of vtkNonLinearCell to represent a three-dimensional, 24-node isoparametric biquadratic hexahedron. The interpolation is the standard finite element, biquadratic-quadratic isoparametric shape function. The cell includes mid-edge and center-face nodes. The ordering of the 24 points defining the cell is point ids (0-7,8-19, 20-23) where point ids 0-7 are the eight corner vertices of the cube; followed by twelve midedge nodes (8-19), nodes 20-23 are the center-face nodes. Note that these midedge nodes correspond lie on the edges defined by (0,1), (1,2), (2,3), (3,0), (4,5), (5,6), (6,7), (7,4), (0,4), (1,5), (2,6), (3,7). The center face nodes lieing in quad 22-(0,1,5,4), 21-(1,2,6,5), 23-(2,3,7,6) and 22-(3,0,4,7)

```
top
 7--14--6
 |      |
15      13
 |      |
 4--12--5

 middle
19--23--18
 |      |
20      21
 |      |
16--22--17

bottom
 3--10--2
 |      |
11      9
 |      |
 0-- 8--1
```

To create an instance of class vtkBiQuadraticQuadraticHexahedron, simply invoke its constructor as follows

```
obj = vtkBiQuadraticQuadraticHexahedron
```

31.17.2 Methods

The class vtkBiQuadraticQuadraticHexahedron has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBiQuadraticQuadraticHexahedron class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBiQuadraticQuadraticHexahedron = obj.NewInstance ()`
- `vtkBiQuadraticQuadraticHexahedron = obj.SafeDownCast (vtkObject o)`

- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this biquadratic hexahedron using scalar value provided. Like contouring, except that it cuts the hex to produce linear tetrahedron.
- `obj.InterpolateFunctions (double pcoords[3], double weights[24])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[72])` - Return the ids of the vertices defining edge/face ('edgeId'/'faceId'). Ids are related to the cell, not to the dataset.

31.18 vtkBiQuadraticQuadraticWedge

31.18.1 Usage

vtkBiQuadraticQuadraticWedge is a concrete implementation of vtkNonLinearCell to represent a three-dimensional, 18-node isoparametric biquadratic wedge. The interpolation is the standard finite element, biquadratic-quadratic isoparametric shape function plus the linear functions. The cell includes a mid-edge node. The ordering of the 18 points defining the cell is point ids (0-5,6-15, 16-18) where point ids 0-5 are the six corner vertices of the wedge; followed by nine midedge nodes (6-15) and 3 center-face nodes. Note that these midedge nodes correspond lie on the edges defined by (0,1), (1,2), (2,0), (3,4), (4,5), (5,3), (0,3), (1,4), (2,5), and the center-face nodes are lieing in quads 16-(0,1,4,3), 17-(1,2,5,4) and (2,0,3,5).

To create an instance of class vtkBiQuadraticQuadraticWedge, simply invoke its constructor as follows

```
obj = vtkBiQuadraticQuadraticWedge
```

31.18.2 Methods

The class vtkBiQuadraticQuadraticWedge has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBiQuadraticQuadraticWedge class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkBiQuadraticQuadraticWedge = obj.NewInstance ()`
- `vtkBiQuadraticQuadraticWedge = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quadratic Wedge using scalar value provided. Like contouring, except that it cuts the hex to produce linear tetrahedron.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the quadratic wedge in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[15])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[45])` - Return the ids of the vertices defining edge/face ('edgeId'/'faceId'). Ids are related to the cell, not to the dataset.

31.19 vtkBiQuadraticTriangle

31.19.1 Usage

vtkBiQuadraticTriangle is a concrete implementation of vtkNonLinearCell to represent a two-dimensional, 7-node, isoparametric parabolic triangle. The interpolation is the standard finite element, bi-quadratic isoparametric shape function. The cell includes three mid-edge nodes besides the three triangle vertices and a center node. The ordering of the three points defining the cell is point ids (0-2,3-6) where id #3 is the midedge node between points (0,1); id #4 is the midedge node between points (1,2); and id #5 is the midedge node between points (2,0). id #6 is the center node of the cell.

To create an instance of class vtkBiQuadraticTriangle, simply invoke its constructor as follows

```
obj = vtkBiQuadraticTriangle
```

31.19.2 Methods

The class vtkBiQuadraticTriangle has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBiQuadraticTriangle class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBiQuadraticTriangle = obj.NewInstance ()`
- `vtkBiQuadraticTriangle = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )`
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quadratic triangle using scalar value provided. Like contouring, except that it cuts the triangle to produce linear triangles.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the quadratic triangle in parametric coordinates.
- `double = obj.GetParametricDistance (double pcoords[3])` - Return the distance of the parametric coordinate provided to the cell. If inside the cell, a distance of zero is returned.
- `obj.InterpolateFunctions (double pcoords[3], double weights[7])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[14])`

31.20 vtkBSPCuts

31.20.1 Usage

This class converts between the vtkKdTree representation of a tree of vtkKdNodes (used by vtkDistributed-DataFilter) and a compact array representation that might be provided by a graph partitioning library like Zoltan. Such a representation could be used in message passing.

To create an instance of class vtkBSPCuts, simply invoke its constructor as follows

```
obj = vtkBSPCuts
```

31.20.2 Methods

The class vtkBSPCuts has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBSPCuts class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBSPCuts = obj.NewInstance ()`
- `vtkBSPCuts = obj.SafeDownCast (vtkObject o)`
- `obj.CreateCuts (double bounds, int ncuts, int dim, double coord, int lower, int upper, double lower`
- `obj.CreateCuts (vtkKdNode kd)`
- `vtkKdNode = obj.GetKdNodeTree ()`
- `int = obj.GetNumberOfCuts ()`
- `int = obj.GetArrays (int len, int dim, double coord, int lower, int upper, double lowerDataCoord, d`
- `int = obj.Equals (vtkBSPCuts other, double tolerance)` - Compare these cuts with those of the other tree. Returns true if the two trees are the same.
- `obj.PrintTree ()`
- `obj.PrintArrays ()`

31.21 vtkBSPIntersections

31.21.1 Usage

Given an axis aligned binary spatial partitioning described by a vtkBSPCuts object, perform intersection queries on various geometric entities with regions of the spatial partitioning.

To create an instance of class vtkBSPIntersections, simply invoke its constructor as follows

```
obj = vtkBSPIntersections
```

31.21.2 Methods

The class vtkBSPIntersections has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBSPIntersections class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBSPIntersections = obj.NewInstance ()`
- `vtkBSPIntersections = obj.SafeDownCast (vtkObject o)`
- `obj.SetCuts (vtkBSPCuts cuts)`
- `vtkBSPCuts = obj.GetCuts ()`
- `int = obj.GetBounds (double bounds)`
- `int = obj.GetNumberOfRegions ()`
- `int = obj.GetRegionBounds (int regionID, double bounds[6])`
- `int = obj.GetRegionDataBounds (int regionID, double bounds[6])`
- `int = obj.IntersectsBox (int regionId, double x)` - Determine whether a region of the spatial decomposition intersects an axis aligned box.
- `int = obj.IntersectsBox (int regionId, double xmin, double xmax, double ymin, double ymax, double z` - Determine whether a region of the spatial decomposition intersects an axis aligned box.
- `int = obj.IntersectsBox (int ids, int len, double x)` - Compute a list of the Ids of all regions that intersect the specified axis aligned box. Returns: the number of ids in the list.
- `int = obj.IntersectsBox (int ids, int len, double x0, double x1, double y0, double y1, double z0, d` - Compute a list of the Ids of all regions that intersect the specified axis aligned box. Returns: the number of ids in the list.
- `int = obj.IntersectsSphere2 (int regionId, double x, double y, double z, double rSquared)` - Determine whether a region of the spatial decomposition intersects a sphere, given the center of the sphere and the square of it's radius.
- `int = obj.IntersectsSphere2 (int ids, int len, double x, double y, double z, double rSquared)` - Compute a list of the Ids of all regions that intersect the specified sphere. The sphere is given by it's center and the square of it's radius. Returns: the number of ids in the list.
- `int = obj.IntersectsCell (int regionId, vtkCell cell, int cellRegion)` - Determine whether a region of the spatial decomposition intersects the given cell. If you already know the region that the cell centroid lies in, provide that as the last argument to make the computation quicker.
- `int = obj.IntersectsCell (int ids, int len, vtkCell cell, int cellRegion)` - Compute a list of the Ids of all regions that intersect the given cell. If you alrady know the region that the cell centroid lies in, provide that as the last argument to make the computation quicker. Returns the number of regions the cell intersects.
- `int = obj.GetComputeIntersectionsUsingDataBounds ()`
- `obj.SetComputeIntersectionsUsingDataBounds (int c)`
- `obj.ComputeIntersectionsUsingDataBoundsOn ()`
- `obj.ComputeIntersectionsUsingDataBoundsOff ()`

31.22 vtkCachedStreamingDemandDrivenPipeline

31.22.1 Usage

vtkCachedStreamingDemandDrivenPipeline

To create an instance of class vtkCachedStreamingDemandDrivenPipeline, simply invoke its constructor as follows

```
obj = vtkCachedStreamingDemandDrivenPipeline
```

31.22.2 Methods

The class vtkCachedStreamingDemandDrivenPipeline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCachedStreamingDemandDrivenPipeline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCachedStreamingDemandDrivenPipeline = obj.NewInstance ()`
- `vtkCachedStreamingDemandDrivenPipeline = obj.SafeDownCast (vtkObject o)`
- `int = obj.Update ()` - Bring the algorithm's outputs up-to-date.
- `int = obj.Update (int port)` - Bring the algorithm's outputs up-to-date.
- `obj.SetCacheSize (int size)` - This is the maximum number of images that can be retained in memory. it defaults to 10.
- `int = obj.GetCacheSize ()` - This is the maximum number of images that can be retained in memory. it defaults to 10.

31.23 vtkCardinalSpline

31.23.1 Usage

vtkCardinalSpline is a concrete implementation of vtkSpline using a Cardinal basis.

To create an instance of class vtkCardinalSpline, simply invoke its constructor as follows

```
obj = vtkCardinalSpline
```

31.23.2 Methods

The class vtkCardinalSpline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCardinalSpline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCardinalSpline = obj.NewInstance ()`
- `vtkCardinalSpline = obj.SafeDownCast (vtkObject o)`
- `obj.Compute ()`
- `double = obj.Evaluate (double t)` - Evaluate a 1D cardinal spline.
- `obj.DeepCopy (vtkSpline s)` - Deep copy of cardinal spline data.

31.24 vtkCastToConcrete

31.24.1 Usage

vtkCastToConcrete is a filter that works around type-checking limitations in the filter classes. Some filters generate abstract types on output, and cannot be connected to the input of filters requiring a concrete input type. For example, vtkElevationFilter generates vtkDataSet for output, and cannot be connected to vtkDecimate, because vtkDecimate requires vtkPolyData as input. This is true even though (in this example) the input to vtkElevationFilter is of type vtkPolyData, and you know the output of vtkElevationFilter is the same type as its input.

vtkCastToConcrete performs run-time checking to insure that output type is of the right type. An error message will result if you try to cast an input type improperly. Otherwise, the filter performs the appropriate cast and returns the data.

To create an instance of class vtkCastToConcrete, simply invoke its constructor as follows

```
obj = vtkCastToConcrete
```

31.24.2 Methods

The class vtkCastToConcrete has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCastToConcrete class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCastToConcrete = obj.NewInstance ()`
- `vtkCastToConcrete = obj.SafeDownCast (vtkObject o)`

31.25 vtkCell

31.25.1 Usage

vtkCell is an abstract class that specifies the interfaces for data cells. Data cells are simple topological elements like points, lines, polygons, and tetrahedra of which visualization datasets are composed. In some cases visualization datasets may explicitly represent cells (e.g., vtkPolyData, vtkUnstructuredGrid), and in some cases, the datasets are implicitly composed of cells (e.g., vtkStructuredPoints).

To create an instance of class vtkCell, simply invoke its constructor as follows

```
obj = vtkCell
```

31.25.2 Methods

The class vtkCell has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCell class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCell = obj.NewInstance ()`
- `vtkCell = obj.SafeDownCast (vtkObject o)`

- `obj.ShallowCopy (vtkCell c)` - Copy this cell by reference counting the internal data structures. This is safe if you want a "read-only" copy. If you modify the cell you might wish to use DeepCopy().
- `obj.DeepCopy (vtkCell c)` - Copy this cell by completely copying internal data structures. This is slower but safer than ShallowCopy().
- `int = obj.GetCellType ()` - Return the type of cell.
- `int = obj.GetCellDimension ()` - Return the topological dimensional of the cell (0,1,2, or 3).
- `int = obj.IsLinear ()` - Some cells require initialization prior to access. For example, they may have to triangulate themselves or set up internal data structures.
- `int = obj.RequiresInitialization ()` - Some cells require initialization prior to access. For example, they may have to triangulate themselves or set up internal data structures.
- `obj.Initialize ()` - Explicit cells require additional representational information beyond the usual cell type and connectivity list information. Most cells in VTK are implicit cells.
- `int = obj.IsExplicitCell ()` - Get the point coordinates for the cell.
- `vtkPoints = obj.GetPoints ()` - Return the number of points in the cell.
- `vtkIdType = obj.GetNumberOfPoints ()` - Return the number of edges in the cell.
- `int = obj.GetNumberOfEdges ()` - Return the number of edges in the cell.
- `int = obj.GetNumberOfFaces ()` - Return the number of faces in the cell.
- `vtkIdList = obj.GetPointIds ()` - For cell point i, return the actual point id.
- `vtkIdType = obj.GetPointId (int ptId)` - Return the edge cell from the edgeId of the cell.
- `vtkCell = obj.GetEdge (int edgeId)` - Return the edge cell from the edgeId of the cell.
- `vtkCell = obj.GetFace (int faceId)` - Return the face cell from the faceId of the cell.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - Given parametric coordinates of a point, return the closest cell boundary, and whether the point is inside or outside of the cell. The cell boundary is defined by a list of points (pts) that specify a face (3D cell), edge (2D cell), or vertex (1D cell). If the return value of the method is != 0, then the point is inside the cell.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - Generate contouring primitives. The scalar list cellScalars are scalar values at each cell point. The point locator is essentially a points list that merges points as they are inserted (i.e., prevents duplicates). Contouring primitives can be vertices, lines, or polygons. It is possible to interpolate point data along the edge by providing input and output point data - if outPd is NULL, then no interpolation is performed. Also, if the output cell data is non-NULL, the cell data from the contoured cell is passed to the generated contouring primitives. (Note: the CopyAllocate() method must be invoked on both the output cell and point data. The cellId refers to the cell from which the cell data is copied.)
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Cut (or clip) the cell based on the input cellScalars and the specified value. The output of the clip operation will be one or more cells of the same topological dimension as the original cell. The flag insideOut controls what part of the cell is considered inside - normally cell points whose scalar value is greater than "value" are considered inside. If insideOut is on, this is reversed. Also, if the output cell data is non-NULL, the cell data from the clipped cell is passed to the generated contouring primitives. (Note: the CopyAllocate() method must be invoked on both the output cell and point data. The cellId refers to the cell from which the cell data is copied.)

- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - Generate simplices of proper dimension. If cell is 3D, tetrahedron are generated; if 2D triangles; if 1D lines; if 0D points. The form of the output is a sequence of points, each n+1 points (where n is topological cell dimension) defining a simplex. The index is a parameter that controls which triangulation to use (if more than one is possible). If numerical degeneracy encountered, 0 is returned, otherwise 1 is returned. This method does not insert new points: all the points that define the simplices are the points that define the cell.

- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - Compute derivatives given cell subId and parametric coordinates. The values array is a series of data value(s) at the cell points. There is a one-to-one correspondence between cell point and data value(s). Dim is the number of data values per cell point. Derivs are derivatives in the x-y-z coordinate directions for each data value. Thus, if computing derivatives for a scalar function in a hexahedron, dim=1, 8 values are supplied, and 3 deriv values are returned (i.e., derivatives in x-y-z directions). On the other hand, if computing derivatives of velocity (vx,vy,vz) dim=3, 24 values are supplied ((vx,vy,vz)1, (vx,vy,vz)2, ...()8), and 9 deriv values are returned ((d(vx)/dx),(d(vx)/dy),(d(vx)/dz), (d(vy)/dx),(d(vy)/dy), (d(vy)/dz), (d(vz)/dx),(d(vz)/dy),(d(vz)/dz)).

- `obj.GetBounds (double bounds[6])` - Compute cell bounding box (xmin,xmax,ymin,ymax,zmin,zmax). Copy result into user provided array.

- `double = obj.GetBounds ()` - Compute cell bounding box (xmin,xmax,ymin,ymax,zmin,zmax). Return pointer to array of six double values.

- `double = obj.GetLength2 ()` - Compute Length squared of cell (i.e., bounding box diagonal squared).

- `int = obj.GetParametricCenter (double pcoords[3])` - Return center of the cell in parametric coordinates. Note that the parametric center is not always located at (0.5,0.5,0.5). The return value is the subId that the center is in (if a composite cell). If you want the center in x-y-z space, invoke the EvaluateLocation() method.

- `double = obj.GetParametricDistance (double pcoords[3])` - Return the distance of the parametric coordinate provided to the cell. If inside the cell, a distance of zero is returned. This is used during picking to get the correct cell picked. (The tolerance will occasionally allow cells to be picked who are not really intersected "inside" the cell.)

- `int = obj.IsPrimaryCell ()` - Return a contiguous array of parametric coordinates of the points defining this cell. In other words, (px,py,pz, px,py,pz, etc..) The coordinates are ordered consistent with the definition of the point ordering for the cell. This method returns a non-NULL pointer when the cell is a primary type (i.e., IsPrimaryCell() is true). Note that 3D parametric coordinates are returned no matter what the topological dimension of the cell.

- `obj.InterpolateFunctions (double pcoords[3], double weights[3])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives) No-ops at this level. Typically overridden in subclasses.

- `obj.InterpolateDerivs (double pcoords[3], double derivs[3])`

31.26 vtkCell3D

31.26.1 Usage

vtkCell3D is an abstract class that extends the interfaces for 3D data cells, and implements methods needed to satisfy the vtkCell API. The 3D cells include hexehedra, tetrahedra, wedge, pyramid, and voxel.

To create an instance of class vtkCell3D, simply invoke its constructor as follows

```
obj = vtkCell3D
```

31.26.2 Methods

The class vtkCell3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCell3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCell3D = obj.NewInstance ()`
- `vtkCell3D = obj.SafeDownCast (vtkObject o)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Cut (or clip) the cell based on the input cellScalars and the specified value. The output of the clip operation will be one or more cells of the same topological dimension as the original cell. The flag insideOut controls what part of the cell is considered inside - normally cell points whose scalar value is greater than "value" are considered inside. If insideOut is on, this is reversed. Also, if the output cell data is non-NULL, the cell data from the clipped cell is passed to the generated contouring primitives. (Note: the CopyAllocate() method must be invoked on both the output cell and point data. The cellId refers to the cell from which the cell data is copied.) (Satisfies vtkCell API.)
- `int = obj.GetCellDimension ()` - Set the tolerance for merging clip intersection points that are near the vertices of cells. This tolerance is used to prevent the generation of degenerate tetrahedra during clipping.
- `obj.SetMergeTolerance (double )` - Set the tolerance for merging clip intersection points that are near the vertices of cells. This tolerance is used to prevent the generation of degenerate tetrahedra during clipping.
- `double = obj.GetMergeToleranceMinValue ()` - Set the tolerance for merging clip intersection points that are near the vertices of cells. This tolerance is used to prevent the generation of degenerate tetrahedra during clipping.
- `double = obj.GetMergeToleranceMaxValue ()` - Set the tolerance for merging clip intersection points that are near the vertices of cells. This tolerance is used to prevent the generation of degenerate tetrahedra during clipping.
- `double = obj.GetMergeTolerance ()` - Set the tolerance for merging clip intersection points that are near the vertices of cells. This tolerance is used to prevent the generation of degenerate tetrahedra during clipping.

31.27 vtkCellArray

31.27.1 Usage

vtkCellArray is a supporting object that explicitly represents cell connectivity. The cell array structure is a raw integer list of the form: (n,id1,id2,...,idn, n,id1,id2,...,idn, ...) where n is the number of points in the cell, and id is a zero-offset index into an associated point list.

Advantages of this data structure are its compactness, simplicity, and easy interface to external data. However, it is totally inadequate for random access. This functionality (when necessary) is accomplished by using the vtkCellTypes and vtkCellLinks objects to extend the definition of the data structure.

To create an instance of class vtkCellArray, simply invoke its constructor as follows

```
obj = vtkCellArray
```

31.27.2 Methods

The class vtkCellArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCellArray = obj.NewInstance ()`
- `vtkCellArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.Allocate (vtkIdType sz, int ext)` - Free any memory and reset to an empty state.
- `obj.Initialize ()` - Free any memory and reset to an empty state.
- `vtkIdType = obj.GetNumberOfCells ()` - Get the number of cells in the array.
- `obj.SetNumberOfCells (vtkIdType )` - Set the number of cells in the array. DO NOT do any kind of allocation, advanced use only.
- `vtkIdType = obj.EstimateSize (vtkIdType numCells, int maxPtsPerCell)` - A cell traversal methods that is more efficient than vtkDataSet traversal methods. InitTraversal() initializes the traversal of the list of cells.
- `obj.InitTraversal ()` - A cell traversal methods that is more efficient than vtkDataSet traversal methods. InitTraversal() initializes the traversal of the list of cells.
- `vtkIdType = obj.GetSize ()` - Get the total number of entries (i.e., data values) in the connectivity array. This may be much less than the allocated size (i.e., return value from GetSize().)
- `vtkIdType = obj.GetNumberOfConnectivityEntries ()` - Internal method used to retrieve a cell given an offset into the internal array.
- `vtkIdType = obj.InsertNextCell (vtkCell cell)` - Insert a cell object. Return the cell id of the cell.
- `vtkIdType = obj.InsertNextCell (vtkIdList pts)` - Create a cell by specifying a list of point ids. Return the cell id of the cell.
- `vtkIdType = obj.InsertNextCell (int npts)` - Create cells by specifying count, and then adding points one at a time using method InsertCellPoint(). If you don't know the count initially, use the method UpdateCellCount() to complete the cell. Return the cell id of the cell.
- `obj.InsertCellPoint (vtkIdType id)` - Used in conjunction with InsertNextCell(int npts) to add another point to the list of cells.
- `obj.UpdateCellCount (int npts)` - Used in conjunction with InsertNextCell(int npts) and InsertCellPoint() to update the number of points defining the cell.
- `vtkIdType = obj.GetInsertLocation (int npts)` - Computes the current insertion location within the internal array. Used in conjunction with GetCell(int loc,...).
- `vtkIdType = obj.GetTraversalLocation ()` - Get/Set the current traversal location.
- `obj.SetTraversalLocation (vtkIdType loc)` - Computes the current traversal location within the internal array. Used in conjunction with GetCell(int loc,...).

- `vtkIdType = obj.GetTraversalLocation (vtkIdType npts)` - Special method inverts ordering of current cell. Must be called carefully or the cell topology may be corrupted.
- `obj.ReverseCell (vtkIdType loc)` - Special method inverts ordering of current cell. Must be called carefully or the cell topology may be corrupted.
- `int = obj.GetMaxCellSize ()` - Returns the size of the largest cell. The size is the number of points defining the cell.
- `vtkIdType = obj.GetPointer ()` - Get pointer to data array for purpose of direct writes of data. Size is the total storage consumed by the cell array. ncells is the number of cells represented in the array.
- `vtkIdType = obj.WritePointer (vtkIdType ncells, vtkIdType size)` - Get pointer to data array for purpose of direct writes of data. Size is the total storage consumed by the cell array. ncells is the number of cells represented in the array.
- `obj.SetCells (vtkIdType ncells, vtkIdTypeArray cells)` - Define multiple cells by providing a connectivity list. The list is in the form (npts,p0,p1,...p(npts-1), repeated for each cell). Be careful using this method because it discards the old cells, and anything referring these cells becomes invalid (for example, if BuildCells() has been called see vtkPolyData). The traversal location is reset to the beginning of the list; the insertion location is set to the end of the list.
- `obj.DeepCopy (vtkCellArray ca)` - Perform a deep copy (no reference counting) of the given cell array.
- `vtkIdTypeArray = obj.GetData ()` - Reuse list. Reset to initial condition.
- `obj.Reset ()` - Reuse list. Reset to initial condition.
- `obj.Squeeze ()` - Return the memory in kilobytes consumed by this cell array. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.
- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this cell array. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.

31.28 vtkCellData

31.28.1 Usage

vtkCellData is a class that is used to represent and manipulate cell attribute data (e.g., scalars, vectors, normals, texture coordinates, etc.) Special methods are provided to work with filter objects, such as passing data through filter, copying data from one cell to another, and interpolating data given cell interpolation weights.

To create an instance of class vtkCellData, simply invoke its constructor as follows

```
obj = vtkCellData
```

31.28.2 Methods

The class vtkCellData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCellData = obj.NewInstance ()`
- `vtkCellData = obj.SafeDownCast (vtkObject o)`

31.29 vtkCellLinks

31.29.1 Usage

vtkCellLinks is a supplemental object to vtkCellArray and vtkCellTypes, enabling access from points to the cells using the points. vtkCellLinks is a list of Links, each link represents a dynamic list of cell id's using the point. The information provided by this object can be used to determine neighbors and construct other local topological information.

To create an instance of class vtkCellLinks, simply invoke its constructor as follows

```
obj = vtkCellLinks
```

31.29.2 Methods

The class vtkCellLinks has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellLinks class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCellLinks = obj.NewInstance ()`
- `vtkCellLinks = obj.SafeDownCast (vtkObject o)`
- `obj.Allocate (vtkIdType numLinks, vtkIdType ext)` - Allocate the specified number of links (i.e., number of points) that will be built.
- `short = obj.GetNcells (vtkIdType ptId)` - Get the number of cells using the point specified by ptId.
- `obj.BuildLinks (vtkDataSet data)` - Build the link list array.
- `obj.BuildLinks (vtkDataSet data, vtkCellArray Connectivity)` - Build the link list array.
- `vtkIdType = obj.GetCells (vtkIdType ptId)` - Return a list of cell ids using the point.
- `vtkIdType = obj.InsertNextPoint (int numLinks)` - Insert a new point into the cell-links data structure. The size parameter is the initial size of the list.
- `obj.InsertNextCellReference (vtkIdType ptId, vtkIdType cellId)` - Insert a cell id into the list of cells (at the end) using the cell id provided. (Make sure to extend the link list (if necessary) using the method ResizeCellList().)
- `obj.DeletePoint (vtkIdType ptId)` - Delete point (and storage) by destroying links to using cells.
- `obj.RemoveCellReference (vtkIdType cellId, vtkIdType ptId)` - Delete the reference to the cell (cellId) from the point (ptId). This removes the reference to the cellId from the cell list, but does not resize the list (recover memory with ResizeCellList(), if necessary).

- `obj.AddCellReference (vtkIdType cellId, vtkIdType ptId)` - Add the reference to the cell (cellId) from the point (ptId). This adds a reference to the cellId from the cell list, but does not resize the list (extend memory with ResizeCellList(), if necessary).
- `obj.ResizeCellList (vtkIdType ptId, int size)` - Change the length of a point's link list (i.e., list of cells using a point) by the size specified.
- `obj.Squeeze ()` - Reclaim any unused memory.
- `obj.Reset ()` - Reset to a state of no entries without freeing the memory.
- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this cell links array. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.
- `obj.DeepCopy (vtkCellLinks src)` - Standard DeepCopy method. Since this object contains no reference to other objects, there is no ShallowCopy.

31.30 vtkCellLocator

31.30.1 Usage

vtkCellLocator is a spatial search object to quickly locate cells in 3D. vtkCellLocator uses a uniform-level octree subdivision, where each octant (an octant is also referred to as a bucket) carries an indication of whether it is empty or not, and each leaf octant carries a list of the cells inside of it. (An octant is not empty if it has one or more cells inside of it.) Typical operations are intersection with a line to return candidate cells, or intersection with another vtkCellLocator to return candidate cells.

To create an instance of class vtkCellLocator, simply invoke its constructor as follows

```
obj = vtkCellLocator
```

31.30.2 Methods

The class vtkCellLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCellLocator = obj.NewInstance ()`
- `vtkCellLocator = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfCellsPerBucket (int N)` - Specify the average number of cells in each octant.
- `int = obj.GetNumberOfCellsPerBucket ()` - reimplemented from vtkAbstractCellLocator to support bad compilers
- `int = obj.IntersectWithLine (double a0[3], double a1[3], vtkPoints points, vtkIdList cellIds)` - Return intersection point (if any) AND the cell which was intersected by the finite line. The cell is returned as a cell id and as a generic cell. For other IntersectWithLine signatures, see vtkAbstractCellLocator
- `vtkIdList = obj.GetCells (int bucket)` - Get the cells in a particular bucket.

- `int = obj.GetNumberOfBuckets (void )` - Return number of buckets available. Insure that the locator has been built before attempting to access buckets (octants).

- `vtkIdType = obj.FindCell (double x[3])` - Find the cell containing a given point. returns -1 if no cell found the cell parameters are copied into the supplied variables, a cell must be provided to store the information.

- `vtkIdType = obj.FindCell (double x[3], double tol2, vtkGenericCell GenCell, double pcoords[3], doub` - Find the cell containing a given point. returns -1 if no cell found the cell parameters are copied into the supplied variables, a cell must be provided to store the information.

- `obj.FindCellsWithinBounds (double bbox, vtkIdList cells)` - Return a list of unique cell ids inside of a given bounding box. The user must provide the vtkIdList to populate. This method returns data only after the locator has been built.

- `obj.FindCellsAlongLine (double p1[3], double p2[3], double tolerance, vtkIdList cells)` - Given a finite line defined by the two points (p1,p2), return the list of unique cell ids in the buckets containing the line. It is possible that an empty cell list is returned. The user must provide the vtkIdList to populate. This method returns data only after the locator has been built.

- `obj.FreeSearchStructure ()` - Satisfy vtkLocator abstract interface.

- `obj.BuildLocator ()` - Satisfy vtkLocator abstract interface.

- `obj.BuildLocatorIfNeeded ()` - Satisfy vtkLocator abstract interface.

- `obj.ForceBuildLocator ()` - Satisfy vtkLocator abstract interface.

- `obj.BuildLocatorInternal ()` - Satisfy vtkLocator abstract interface.

- `obj.GenerateRepresentation (int level, vtkPolyData pd)` - Satisfy vtkLocator abstract interface.

31.31 vtkCellLocatorInterpolatedVelocityField

31.31.1 Usage

vtkCellLocatorInterpolatedVelocityField acts as a continuous velocity field via cell interpolation on a vtkDataSet, NumberOfIndependentVariables = 4 (x,y,z,t) and NumberOfFunctions = 3 (u,v,w). As a concrete sub-class of vtkAbstractInterpolatedVelocityField, it adopts vtkAbstractCellLocator's sub-classes, e.g., vtkCellLocator and vtkModifiedBSPTree, without the use of vtkPointLocator (employed by vtkDataSet/vtkPointSet::FindCell() in vtkInterpolatedVelocityField). vtkCellLocatorInterpolatedVelocityField adopts one level of cell caching. Specifically, if the next point is still within the previous cell, cell location is then simply skipped and vtkCell::EvaluatePosition() is called to obtain the new parametric coordinates and weights that are used to interpolate the velocity function values across the vertices of this cell. Otherwise a global cell (the target containing the next point) location is instead directly invoked, without exploiting the clue that vtkInterpolatedVelocityField makes use of from the previous cell (an immediate neighbor). Although ignoring the neighbor cell may incur a relatively high computational cost, vtkCellLocatorInterpolatedVelocityField is more robust in locating the target cell than its sibling class vtkInterpolatedVelocityField.

To create an instance of class vtkCellLocatorInterpolatedVelocityField, simply invoke its constructor as follows

```
obj = vtkCellLocatorInterpolatedVelocityField
```

31.31.2 Methods

The class vtkCellLocatorInterpolatedVelocityField has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellLocatorInterpolatedVelocityField class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCellLocatorInterpolatedVelocityField = obj.NewInstance ()`
- `vtkCellLocatorInterpolatedVelocityField = obj.SafeDownCast (vtkObject o)`
- `vtkAbstractCellLocator = obj.GetLastCellLocator ()` - Get the cell locator attached to the most recently visited dataset.
- `vtkAbstractCellLocator = obj.GetCellLocatorPrototype ()` - Get the prototype of the cell locator that is used for interpolating the velocity field during integration.
- `obj.SetCellLocatorPrototype (vtkAbstractCellLocator prototype)` - Set a prototype of the cell locator that is used for interpolating the velocity field during integration.
- `obj.CopyParameters (vtkAbstractInterpolatedVelocityField from)` - Import parameters. Subclasses can add more after chaining.
- `obj.AddDataSet (vtkDataSet dataset)` - Add a dataset coupled with a cell locator (of type vtkAbstractCellLocator) for vector function evaluation. Note the use of a vtkAbstractCellLocator enables robust cell location. If more than one dataset is added, the evaluation point is searched in all until a match is found. THIS FUNCTION DOES NOT CHANGE THE REFERENCE COUNT OF dataset FOR THREAD SAFETY REASONS.
- `int = obj.FunctionValues (double x, double f)` - Evaluate the velocity field f at point (x, y, z).
- `obj.SetLastCellId (vtkIdType c, int dataindex)` - Set the cell id cached by the last evaluation within a specified dataset.
- `obj.SetLastCellId (vtkIdType c)`

31.32 vtkCellTypes

31.32.1 Usage

This class is a supplemental object to vtkCellArray to allow random access into cells as well as representing cell type information. The "location" field is the location in the vtkCellArray list in terms of an integer offset. An integer offset was used instead of a pointer for easy storage and inter-process communication. The type information is defined in the file vtkCellType.h.

To create an instance of class vtkCellTypes, simply invoke its constructor as follows

```
obj = vtkCellTypes
```

31.32.2 Methods

The class vtkCellTypes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellTypes class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkCellTypes = obj.NewInstance ()`
- `vtkCellTypes = obj.SafeDownCast (vtkObject o)`
- `int = obj.Allocate (int sz, int ext)` - Allocate memory for this array. Delete old storage only if necessary.
- `obj.InsertCell (int id, char type, int loc)` - Add a cell at specified id.
- `int = obj.InsertNextCell (char type, int loc)` - Add a cell to the object in the next available slot.
- `obj.SetCellTypes (int ncells, vtkUnsignedCharArray cellTypes, vtkIntArray cellLocations)` - Specify a group of cell types.
- `int = obj.GetCellLocation (int cellId)` - Return the location of the cell in the associated vtkCellArray.
- `obj.DeleteCell (vtkIdType cellId)` - Delete cell by setting to NULL cell type.
- `int = obj.GetNumberOfTypes ()` - Return the number of types in the list.
- `int = obj.IsType (char type)` - Return 1 if type specified is contained in list; 0 otherwise.
- `int = obj.InsertNextType (char type)` - Add the type specified to the end of the list. Range checking is performed.
- `char = obj.GetCellType (int cellId)` - Return the type of cell.
- `obj.Squeeze ()` - Reclaim any extra memory.
- `obj.Reset ()` - Initialize object without releasing memory.
- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this cell type array. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object. The information returned is valid only after the pipeline has been updated.
- `obj.DeepCopy (vtkCellTypes src)` - Standard DeepCopy method. Since this object contains no reference to other objects, there is no ShallowCopy.

31.33 vtkColorTransferFunction

31.33.1 Usage

vtkColorTransferFunction is a color mapping in RGB or HSV space that uses piecewise hermite functions to allow interpolation that can be piecewise constant, piecewise linear, or somewhere in-between (a modified piecewise hermite function that squishes the function according to a sharpness parameter). The function also allows for the specification of the midpoint (the place where the function reaches the average of the two bounding nodes) as a normalize distance between nodes. See the description of class vtkPiecewiseFunction for an explanation of midpoint and sharpness.

To create an instance of class vtkColorTransferFunction, simply invoke its constructor as follows

```
obj = vtkColorTransferFunction
```

31.33.2 Methods

The class vtkColorTransferFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkColorTransferFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkColorTransferFunction = obj.NewInstance ()`
- `vtkColorTransferFunction = obj.SafeDownCast (vtkObject o)`
- `obj.DeepCopy (vtkColorTransferFunction f)`
- `obj.ShallowCopy (vtkColorTransferFunction f)`
- `int = obj.GetSize ()` - How many points are there defining this function?
- `int = obj.AddRGBPoint (double x, double r, double g, double b)` - Add/Remove a point to/from the function defined in RGB or HSV Return the index of the point (0 based), or -1 on error. See the description of class vtkPiecewiseFunction for an explanation of midpoint and sharpness.
- `int = obj.AddRGBPoint (double x, double r, double g, double b, double midpoint, double sharpness)` - Add/Remove a point to/from the function defined in RGB or HSV Return the index of the point (0 based), or -1 on error. See the description of class vtkPiecewiseFunction for an explanation of midpoint and sharpness.
- `int = obj.AddHSVPoint (double x, double h, double s, double v)` - Add/Remove a point to/from the function defined in RGB or HSV Return the index of the point (0 based), or -1 on error. See the description of class vtkPiecewiseFunction for an explanation of midpoint and sharpness.
- `int = obj.AddHSVPoint (double x, double h, double s, double v, double midpoint, double sharpness)` - Add/Remove a point to/from the function defined in RGB or HSV Return the index of the point (0 based), or -1 on error. See the description of class vtkPiecewiseFunction for an explanation of midpoint and sharpness.
- `int = obj.RemovePoint (double x)` - Add/Remove a point to/from the function defined in RGB or HSV Return the index of the point (0 based), or -1 on error. See the description of class vtkPiecewiseFunction for an explanation of midpoint and sharpness.
- `obj.AddRGBSegment (double x1, double r1, double g1, double b1, double x2, double r2, double g2, dou` - Add two points to the function and remove all the points between them
- `obj.AddHSVSegment (double x1, double h1, double s1, double v1, double x2, double h2, double s2, dou` - Add two points to the function and remove all the points between them
- `obj.RemoveAllPoints ()` - Remove all points
- `double = obj.GetColor (double x)` - Returns an RGB color for the specified scalar value
- `obj.GetColor (double x, double rgb[3])` - Returns an RGB color for the specified scalar value
- `double = obj.GetRedValue (double x)` - Get the color components individually.
- `double = obj.GetGreenValue (double x)` - Get the color components individually.
- `double = obj.GetBlueValue (double x)` - Get the color components individually.

- `int = obj.GetNodeValue (int index, double val[6])` - For the node specified by index, set/get the location (X), R, G, and B values, midpoint, and sharpness values at the node.
- `int = obj.SetNodeValue (int index, double val[6])` - For the node specified by index, set/get the location (X), R, G, and B values, midpoint, and sharpness values at the node.
- `double = obj. GetRange ()` - Returns min and max position of all function points.
- `int = obj.AdjustRange (double range[2])` - Remove all points out of the new range, and make sure there is a point at each end of that range. Return 1 on success, 0 otherwise.
- `obj.GetTable (double x1, double x2, int n, double table)` - Fills in a table of n function values between x1 and x2
- `obj.GetTable (double x1, double x2, int n, float table)` - Fills in a table of n function values between x1 and x2
- `obj.BuildFunctionFromTable (double x1, double x2, int size, double table)` - Construct a color transfer function from a table. Function range is is set to [x1, x2], each function size is set to size, and function points are regularly spaced between x1 and x2. Parameter "table" is assumed to be a block of memory of size [3*size]
- `obj.SetClamping (int )` - Sets and gets the clamping value for this transfer function.
- `int = obj.GetClampingMinValue ()` - Sets and gets the clamping value for this transfer function.
- `int = obj.GetClampingMaxValue ()` - Sets and gets the clamping value for this transfer function.
- `int = obj.GetClamping ()` - Sets and gets the clamping value for this transfer function.
- `obj.ClampingOn ()` - Sets and gets the clamping value for this transfer function.
- `obj.ClampingOff ()` - Sets and gets the clamping value for this transfer function.
- `obj.SetColorSpace (int )` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.
- `int = obj.GetColorSpaceMinValue ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.
- `int = obj.GetColorSpaceMaxValue ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.
- `obj.SetColorSpaceToRGB ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.

- `obj.SetColorSpaceToHSV ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.

- `obj.SetColorSpaceToLab ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.

- `obj.SetColorSpaceToDiverging ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.

- `int = obj.GetColorSpace ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.

- `obj.SetHSVWrap (int )` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.

- `int = obj.GetHSVWrap ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.

- `obj.HSVWrapOn ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.

- `obj.HSVWrapOff ()` - Set/Get the color space used for interpolation: RGB, HSV, CIELAB, or Diverging. In HSV mode, if HSVWrap is on, it will take the shortest path in Hue (going back through 0 if that is the shortest way around the hue circle) whereas if HSVWrap is off it will not go through 0 (in order the match the current functionality of vtkLookupTable). Diverging is a special mode where colors will pass through white when interpolating between two saturated colors.

- `obj.SetScale (int )` - Set the type of scale to use, linear or logarithmic. The default is linear. If the scale is logarithmic, and the range contains zero, the color mapping will be linear.

- `obj.SetScaleToLinear ()` - Set the type of scale to use, linear or logarithmic. The default is linear. If the scale is logarithmic, and the range contains zero, the color mapping will be linear.

- `obj.SetScaleToLog10 ()` - Set the type of scale to use, linear or logarithmic. The default is linear. If the scale is logarithmic, and the range contains zero, the color mapping will be linear.

- `int = obj.GetScale ()` - Set the type of scale to use, linear or logarithmic. The default is linear. If the scale is logarithmic, and the range contains zero, the color mapping will be linear.
- `obj.FillFromDataPointer (int , double )` - Returns a list of all nodes Fills from a pointer to data stored in a similar list of nodes.
- `obj.SetAllowDuplicateScalars (int )` - Toggle whether to allow duplicate scalar values in the color transfer function (off by default).
- `int = obj.GetAllowDuplicateScalars ()` - Toggle whether to allow duplicate scalar values in the color transfer function (off by default).
- `obj.AllowDuplicateScalarsOn ()` - Toggle whether to allow duplicate scalar values in the color transfer function (off by default).
- `obj.AllowDuplicateScalarsOff ()` - Toggle whether to allow duplicate scalar values in the color transfer function (off by default).

31.34 vtkCompositeDataIterator

31.34.1 Usage

vtkCompositeDataIterator provides an interface for accessing datasets in a collection (vtkCompositeDataIterator).

To create an instance of class vtkCompositeDataIterator, simply invoke its constructor as follows

```
obj = vtkCompositeDataIterator
```

31.34.2 Methods

The class vtkCompositeDataIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositeDataIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositeDataIterator = obj.NewInstance ()`
- `vtkCompositeDataIterator = obj.SafeDownCast (vtkObject o)`
- `obj.SetDataSet (vtkCompositeDataSet ds)` - Set the composite dataset this iterator is iterating over. Must be set before traversal begins.
- `vtkCompositeDataSet = obj.GetDataSet ()` - Set the composite dataset this iterator is iterating over. Must be set before traversal begins.
- `obj.InitTraversal ()` - Begin iterating over the composite dataset structure.
- `obj.InitReverseTraversal ()` - Begin iterating over the composite dataset structure in reverse order.
- `obj.GoToFirstItem ()` - Move the iterator to the beginning of the collection.
- `obj.GoToNextItem ()` - Move the iterator to the next item in the collection.
- `int = obj.IsDoneWithTraversal ()` - Test whether the iterator is finished with the traversal. Returns 1 for yes, and 0 for no. It is safe to call any of the GetCurrent...() methods only when IsDoneWithTraversal() returns 0.

- `vtkDataObject = obj.GetCurrentDataObject ()` - Returns the current item. Valid only when IsDoneWithTraversal() returns 0.
- `vtkInformation = obj.GetCurrentMetaData ()` - Returns the meta-data associated with the current item. This will allocate a new vtkInformation object is none is already present. Use HasCurrentMetaData to avoid unnecessary creation of vtkInformation objects.
- `int = obj.HasCurrentMetaData ()` - Returns if the a meta-data information object is present for the current item. Return 1 on success, 0 otherwise.
- `obj.SetVisitOnlyLeaves (int )` - If VisitOnlyLeaves is true, the iterator will only visit nodes (sub-datasets) that are not composite. If it encounters a composite data set, it will automatically traverse that composite dataset until it finds non-composite datasets. With this options, it is possible to visit all non-composite datasets in tree of composite datasets (composite of composite of composite for example :-)) If VisitOnlyLeaves is false, GetCurrentDataObject() may return vtkCompositeDataSet. By default, VisitOnlyLeaves is 1.
- `int = obj.GetVisitOnlyLeaves ()` - If VisitOnlyLeaves is true, the iterator will only visit nodes (sub-datasets) that are not composite. If it encounters a composite data set, it will automatically traverse that composite dataset until it finds non-composite datasets. With this options, it is possible to visit all non-composite datasets in tree of composite datasets (composite of composite of composite for example :-)) If VisitOnlyLeaves is false, GetCurrentDataObject() may return vtkCompositeDataSet. By default, VisitOnlyLeaves is 1.
- `obj.VisitOnlyLeavesOn ()` - If VisitOnlyLeaves is true, the iterator will only visit nodes (sub-datasets) that are not composite. If it encounters a composite data set, it will automatically traverse that composite dataset until it finds non-composite datasets. With this options, it is possible to visit all non-composite datasets in tree of composite datasets (composite of composite of composite for example :-)) If VisitOnlyLeaves is false, GetCurrentDataObject() may return vtkCompositeDataSet. By default, VisitOnlyLeaves is 1.
- `obj.VisitOnlyLeavesOff ()` - If VisitOnlyLeaves is true, the iterator will only visit nodes (sub-datasets) that are not composite. If it encounters a composite data set, it will automatically traverse that composite dataset until it finds non-composite datasets. With this options, it is possible to visit all non-composite datasets in tree of composite datasets (composite of composite of composite for example :-)) If VisitOnlyLeaves is false, GetCurrentDataObject() may return vtkCompositeDataSet. By default, VisitOnlyLeaves is 1.
- `obj.SetTraverseSubTree (int )` - If TraverseSubTree is set to true, the iterator will visit the entire tree structure, otherwise it only visits the first level children. Set to 1 by default.
- `int = obj.GetTraverseSubTree ()` - If TraverseSubTree is set to true, the iterator will visit the entire tree structure, otherwise it only visits the first level children. Set to 1 by default.
- `obj.TraverseSubTreeOn ()` - If TraverseSubTree is set to true, the iterator will visit the entire tree structure, otherwise it only visits the first level children. Set to 1 by default.
- `obj.TraverseSubTreeOff ()` - If TraverseSubTree is set to true, the iterator will visit the entire tree structure, otherwise it only visits the first level children. Set to 1 by default.
- `obj.SetSkipEmptyNodes (int )` - If SkipEmptyNodes is true, then NULL datasets will be skipped. Default is true.
- `int = obj.GetSkipEmptyNodes ()` - If SkipEmptyNodes is true, then NULL datasets will be skipped. Default is true.
- `obj.SkipEmptyNodesOn ()` - If SkipEmptyNodes is true, then NULL datasets will be skipped. Default is true.

- `obj.SkipEmptyNodesOff ()` - If SkipEmptyNodes is true, then NULL datasets will be skipped. Default is true.
- `int = obj.GetCurrentFlatIndex ()` - Flat index is an index obtained by traversing the tree in pre-order. This can be used to uniquely identify nodes in the tree. Not valid if IsDoneWithTraversal() returns true.
- `int = obj.GetReverse ()` - Returns if the iteration is in reverse order.

31.35 vtkCompositeDataPipeline

31.35.1 Usage

vtkCompositeDataPipeline is an executive that supports the processing of composite dataset. It supports algorithms that are aware of composite dataset as well as those that are not. Type checking is performed at run time. Algorithms that are not composite dataset-aware have to support all dataset types contained in the composite dataset. The pipeline execution can be summarized as follows:

* REQUEST_INFORMATION: The producers have to provide information about the contents of the composite dataset in this pass. Sources that can produce more than one piece (note that a piece is different than a block; each piece consistes of 0 or more blocks) should set MAXIMUM_NUMBER_OF_PIECES to -1.

* REQUEST_UPDATE_EXTENT: This pass is identical to the one implemented in vtkStreamingDemandDrivenPipeline

* REQUEST_DATA: This is where the algorithms execute. If the vtkCompositeDataPipeline is assigned to a simple filter, it will invoke the vtkStreamingDemandDrivenPipeline passes in a loop, passing a different block each time and will collect the results in a composite dataset. .SECTION See also vtkCompositeDataSet

To create an instance of class vtkCompositeDataPipeline, simply invoke its constructor as follows

```
obj = vtkCompositeDataPipeline
```

31.35.2 Methods

The class vtkCompositeDataPipeline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositeDataPipeline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositeDataPipeline = obj.NewInstance ()`
- `vtkCompositeDataPipeline = obj.SafeDownCast (vtkObject o)`
- `vtkDataObject = obj.GetCompositeOutputData (int port)` - Returns the data object stored with the DATA_OBJECT() in the output port

31.36 vtkCompositeDataSet

31.36.1 Usage

vtkCompositeDataSet is an abstract class that represents a collection of datasets (including other composite datasets). It provides an interface to access the datasets through iterators. vtkCompositeDataSet provides methods that are used by subclasses to store the datasets. vtkCompositeDataSet provides the datastructure for a full tree representation. Subclasses provide the semantics for it and control how this tree is built.

To create an instance of class vtkCompositeDataSet, simply invoke its constructor as follows

```
obj = vtkCompositeDataSet
```

31.36.2 Methods

The class vtkCompositeDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositeDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositeDataSet = obj.NewInstance ()`
- `vtkCompositeDataSet = obj.SafeDownCast (vtkObject o)`
- `vtkCompositeDataIterator = obj.NewIterator ()` - Return a new iterator (the iterator has to be deleted by user).
- `int = obj.GetDataObjectType ()` - Get the port currently producing this object.
- `vtkAlgorithmOutput = obj.GetProducerPort ()` - Get the port currently producing this object.
- `obj.CopyStructure (vtkCompositeDataSet input)` - Copies the tree structure from the input. All pointers to non-composite data objects are intialized to NULL. This also shallow copies the meta data associated with all the nodes.
- `obj.SetDataSet (vtkCompositeDataIterator iter, vtkDataObject dataObj)` - Sets the data set at the location pointed by the iterator. The iterator does not need to be iterating over this dataset itself. It can be any composite datasite with similar structure (achieved by using CopyStructure).
- `vtkDataObject = obj.GetDataSet (vtkCompositeDataIterator iter)` - Returns the dataset located at the positiong pointed by the iterator. The iterator does not need to be iterating over this dataset itself. It can be an iterator for composite dataset with similar structure (achieved by using CopyStructure).
- `vtkInformation = obj.GetMetaData (vtkCompositeDataIterator iter)` - Returns the meta-data associated with the position pointed by the iterator. This will create a new vtkInformation object if none already exists. Use HasMetaData to avoid creating the vtkInformation object unnecessarily. The iterator does not need to be iterating over this dataset itself. It can be an iterator for composite dataset with similar structure (achieved by using CopyStructure).
- `int = obj.HasMetaData (vtkCompositeDataIterator iter)` - Returns if any meta-data associated with the position pointed by the iterator. The iterator does not need to be iterating over this dataset itself. It can be an iterator for composite dataset with similar structure (achieved by using CopyStructure).
- `obj.Initialize ()` - Restore data object to initial state,
- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.
- `vtkIdType = obj.GetNumberOfPoints ()` - Returns the total number of points of all blocks. This will iterate over all blocks and call GetNumberOfPoints() so it might be expansive.

31.37 vtkCompositeDataSetAlgorithm

31.37.1 Usage

Algorithms that take any type of data object (including composite dataset) and produce a vtkCompositeDataSet in the output can subclass from this class.

To create an instance of class vtkCompositeDataSetAlgorithm, simply invoke its constructor as follows

```
obj = vtkCompositeDataSetAlgorithm
```

31.37.2 Methods

The class vtkCompositeDataSetAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositeDataSetAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositeDataSetAlgorithm = obj.NewInstance ()`
- `vtkCompositeDataSetAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkCompositeDataSet = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkCompositeDataSet = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.38 vtkCone

31.38.1 Usage

vtkCone computes the implicit function and function gradient for a cone. vtkCone is a concrete implementation of vtkImplicitFunction. The cone vertex is located at the origin with axis of rotation coincident with x-axis. (Use the superclass' vtkImplicitFunction transformation matrix if necessary to reposition.) The angle specifies the angle between the axis of rotation and the side of the cone.

To create an instance of class vtkCone, simply invoke its constructor as follows

```
obj = vtkCone
```

31.38.2 Methods

The class vtkCone has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCone class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCone = obj.NewInstance ()`
- `vtkCone = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double g[3])`
- `obj.SetAngle (double )` - Set/Get the cone angle (expressed in degrees).
- `double = obj.GetAngleMinValue ()` - Set/Get the cone angle (expressed in degrees).
- `double = obj.GetAngleMaxValue ()` - Set/Get the cone angle (expressed in degrees).
- `double = obj.GetAngle ()` - Set/Get the cone angle (expressed in degrees).

31.39 vtkConvexPointSet

31.39.1 Usage

vtkConvexPointSet is a concrete implementation that represents a 3D cell defined by a convex set of points. An example of such a cell is an octant (from an octree). vtkConvexPointSet uses the ordered triangulations approach (vtkOrderedTriangulator) to create triangulations guaranteed to be compatible across shared faces. This allows a general approach to processing complex, convex cell types.

To create an instance of class vtkConvexPointSet, simply invoke its constructor as follows

```
obj = vtkConvexPointSet
```

31.39.2 Methods

The class vtkConvexPointSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkConvexPointSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkConvexPointSet = obj.NewInstance ()`
- `vtkConvexPointSet = obj.SafeDownCast (vtkObject o)`
- `int = obj.HasFixedTopology ()` - See vtkCell3D API for description of these methods.
- `int = obj.GetCellType ()` - This cell requires that it be initialized prior to access.
- `int = obj.RequiresInitialization ()` - This cell requires that it be initialized prior to access.
- `obj.Initialize ()` - This cell requires that it be initialized prior to access.

- `int = obj.GetNumberOfEdges ()` - A convex point set has no explicit cell edge or faces; however implicitly (after triangulation) it does. Currently the method GetNumberOfEdges() always returns 0 while the GetNumberOfFaces() returns the number of boundary triangles of the triangulation of the convex point set. The method GetNumberOfFaces() triggers a triangulation of the convex point set; repeated calls to GetFace() then return the boundary faces. (Note: GetNumberOfEdges() currently returns 0 because it is a rarely used method and hard to implement. It can be changed in the future.

- `vtkCell = obj.GetEdge (int )` - A convex point set has no explicit cell edge or faces; however implicitly (after triangulation) it does. Currently the method GetNumberOfEdges() always returns 0 while the GetNumberOfFaces() returns the number of boundary triangles of the triangulation of the convex point set. The method GetNumberOfFaces() triggers a triangulation of the convex point set; repeated calls to GetFace() then return the boundary faces. (Note: GetNumberOfEdges() currently returns 0 because it is a rarely used method and hard to implement. It can be changed in the future.

- `int = obj.GetNumberOfFaces ()` - A convex point set has no explicit cell edge or faces; however implicitly (after triangulation) it does. Currently the method GetNumberOfEdges() always returns 0 while the GetNumberOfFaces() returns the number of boundary triangles of the triangulation of the convex point set. The method GetNumberOfFaces() triggers a triangulation of the convex point set; repeated calls to GetFace() then return the boundary faces. (Note: GetNumberOfEdges() currently returns 0 because it is a rarely used method and hard to implement. It can be changed in the future.

- `vtkCell = obj.GetFace (int faceId)` - A convex point set has no explicit cell edge or faces; however implicitly (after triangulation) it does. Currently the method GetNumberOfEdges() always returns 0 while the GetNumberOfFaces() returns the number of boundary triangles of the triangulation of the convex point set. The method GetNumberOfFaces() triggers a triangulation of the convex point set; repeated calls to GetFace() then return the boundary faces. (Note: GetNumberOfEdges() currently returns 0 because it is a rarely used method and hard to implement. It can be changed in the future.

- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - Satisfy the vtkCell API. This method contours by triangulating the cell and then contouring the resulting tetrahedra.

- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Satisfy the vtkCell API. This method contours by triangulating the cell and then adding clip-edge intersection points into the triangulation; extracting the clipped region.

- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - Triangulate using methods of vtkOrderedTriangulator.

- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - Computes derivatives by triangulating and from subId and pcoords, evaluating derivatives on the resulting tetrahedron.

- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - Returns the set of points forming a face of the triangulation of these points that are on the boundary of the cell that are closest parametrically to the point specified.

- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the cell in parametric coordinates.

- `int = obj.IsPrimaryCell ()` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)

- `obj.InterpolateFunctions (double pcoords[3], double sf)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)

- `obj.InterpolateDerivs (double pcoords[3], double derivs)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)

31.40 vtkCoordinate

31.40.1 Usage

vtkCoordinate represents position in a variety of coordinate systems, and converts position to other coordinate systems. It also supports relative positioning, so you can create a cascade of vtkCoordinate objects (no loops please!) that refer to each other. The typical usage of this object is to set the coordinate system in which to represent a position (e.g., SetCoordinateSystemToNormalizedDisplay()), set the value of the coordinate (e.g., SetValue()), and then invoke the appropriate method to convert to another coordinate system (e.g., GetComputedWorldValue()).

The coordinate systems in vtk are as follows: ¡PRE¿ DISPLAY - x-y pixel values in window NORMALIZED DISPLAY - x-y (0,1) normalized values VIEWPORT - x-y pixel values in viewport NORMALIZED VIEWPORT - x-y (0,1) normalized value in viewport VIEW - x-y-z (-1,1) values in camera coordinates. (z is depth) WORLD - x-y-z global coordinate values USERDEFINED - x-y-z in User defined space ¡/PRE¿

If you cascade vtkCoordinate objects, you refer to another vtkCoordinate object which in turn can refer to others, and so on. This allows you to create composite groups of things like vtkActor2D that are positioned relative to one another. Note that in cascaded sequences, each vtkCoordinate object may be specified in different coordinate systems!

To create an instance of class vtkCoordinate, simply invoke its constructor as follows

```
obj = vtkCoordinate
```

31.40.2 Methods

The class vtkCoordinate has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCoordinate class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCoordinate = obj.NewInstance ()`
- `vtkCoordinate = obj.SafeDownCast (vtkObject o)`
- `obj.SetCoordinateSystem (int )` - Set/get the coordinate system which this coordinate is defined in. The options are Display, Normalized Display, Viewport, Normalized Viewport, View, and World.
- `int = obj.GetCoordinateSystem ()` - Set/get the coordinate system which this coordinate is defined in. The options are Display, Normalized Display, Viewport, Normalized Viewport, View, and World.
- `obj.SetCoordinateSystemToDisplay ()` - Set/get the coordinate system which this coordinate is defined in. The options are Display, Normalized Display, Viewport, Normalized Viewport, View, and World.
- `obj.SetCoordinateSystemToNormalizedDisplay ()` - Set/get the coordinate system which this coordinate is defined in. The options are Display, Normalized Display, Viewport, Normalized Viewport, View, and World.
- `obj.SetCoordinateSystemToViewport ()` - Set/get the coordinate system which this coordinate is defined in. The options are Display, Normalized Display, Viewport, Normalized Viewport, View, and World.
- `obj.SetCoordinateSystemToNormalizedViewport ()` - Set/get the coordinate system which this coordinate is defined in. The options are Display, Normalized Display, Viewport, Normalized Viewport, View, and World.

- `obj.SetCoordinateSystemToView ()` - Set/get the coordinate system which this coordinate is defined in. The options are Display, Normalized Display, Viewport, Normalized Viewport, View, and World.

- `obj.SetCoordinateSystemToWorld ()`

- `string = obj.GetCoordinateSystemAsString ()`

- `obj.SetValue (double , double , double )` - Set/get the value of this coordinate. This can be thought of as the position of this coordinate in its coordinate system.

- `obj.SetValue (double a[3])` - Set/get the value of this coordinate. This can be thought of as the position of this coordinate in its coordinate system.

- `double = obj. GetValue ()` - Set/get the value of this coordinate. This can be thought of as the position of this coordinate in its coordinate system.

- `obj.SetValue (double a, double b)` - If this coordinate is relative to another coordinate, then specify that coordinate as the ReferenceCoordinate. If this is NULL the coordinate is assumed to be absolute.

- `obj.SetReferenceCoordinate (vtkCoordinate )` - If this coordinate is relative to another coordinate, then specify that coordinate as the ReferenceCoordinate. If this is NULL the coordinate is assumed to be absolute.

- `vtkCoordinate = obj.GetReferenceCoordinate ()` - If this coordinate is relative to another coordinate, then specify that coordinate as the ReferenceCoordinate. If this is NULL the coordinate is assumed to be absolute.

- `obj.SetViewport (vtkViewport viewport)` - If you want this coordinate to be relative to a specific vtkViewport (vtkRenderer) then you can specify that here. NOTE: this is a raw pointer, not a weak pointer not a reference counted object to avoid reference cycle loop between rendering classes and filter classes.

- `vtkViewport = obj.GetViewport ()` - If you want this coordinate to be relative to a specific vtkViewport (vtkRenderer) then you can specify that here. NOTE: this is a raw pointer, not a weak pointer not a reference counted object to avoid reference cycle loop between rendering classes and filter classes.

- `double = obj.GetComputedWorldValue (vtkViewport )` - Return the computed value in a specified coordinate system.

- `int = obj.GetComputedViewportValue (vtkViewport )` - Return the computed value in a specified coordinate system.

- `int = obj.GetComputedDisplayValue (vtkViewport )` - Return the computed value in a specified coordinate system.

- `int = obj.GetComputedLocalDisplayValue (vtkViewport )` - Return the computed value in a specified coordinate system.

- `double = obj.GetComputedDoubleViewportValue (vtkViewport )`

- `double = obj.GetComputedDoubleDisplayValue (vtkViewport )`

31.41 vtkCubicLine

31.41.1 Usage

vtkCubicLine is a concrete implementation of vtkNonLinearCell to represent a 1D Cubic line. The Cubic Line is the 4 nodes isoparametric parabolic line . The interpolation is the standard finite element, cubic isoparametric shape function. The cell includes two mid-edge nodes. The ordering of the four points defining the cell is point ids (0,1,2,3) where id #2 and #3 are the mid-edge nodes. Please note that the parametric coordinates lie between -1 and 1 in accordance with most standard documentations. .SECTION Thanks ¡verbatim¿ This file has been developed by Oxalya - www.oxalya.com Copyright (c) EDF - www.edf.fr ¡/verbatim¿

To create an instance of class vtkCubicLine, simply invoke its constructor as follows

```
obj = vtkCubicLine
```

31.41.2 Methods

The class vtkCubicLine has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCubicLine class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCubicLine = obj.NewInstance ()`
- `vtkCubicLine = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `double = obj.GetParametricDistance (double pcoords[3])` - Return the distance of the parametric coordinate provided to the cell. If inside the cell, a distance of zero is returned.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this line using scalar value provided. Like contouring, except that it cuts the line to produce other lines.

- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the triangle in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[4])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[4])`

31.42 vtkCylinder

31.42.1 Usage

vtkCylinder computes the implicit function and function gradient for a cylinder. vtkCylinder is a concrete implementation of vtkImplicitFunction. Cylinder is centered at Center and axes of rotation is along the y-axis. (Use the superclass' vtkImplicitFunction transformation matrix if necessary to reposition.)

To create an instance of class vtkCylinder, simply invoke its constructor as follows

```
obj = vtkCylinder
```

31.42.2 Methods

The class vtkCylinder has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCylinder class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCylinder = obj.NewInstance ()`
- `vtkCylinder = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double g[3])`
- `obj.SetRadius (double )` - Set/Get cylinder radius.
- `double = obj.GetRadius ()` - Set/Get cylinder radius.
- `obj.SetCenter (double , double , double )` - Set/Get cylinder center
- `obj.SetCenter (double  a[3])` - Set/Get cylinder center
- `double = obj. GetCenter ()` - Set/Get cylinder center

31.43 vtkDataObject

31.43.1 Usage

vtkDataObject is an general representation of visualization data. It serves to encapsulate instance variables and methods for visualization network execution, as well as representing data consisting of a field (i.e., just an unstructured pile of data). This is to be compared with a vtkDataSet, which is data with geometric and/or topological structure.

vtkDataObjects are used to represent arbitrary repositories of data via the vtkFieldData instance variable. These data must be eventually mapped into a concrete subclass of vtkDataSet before they can actually be displayed.

To create an instance of class vtkDataObject, simply invoke its constructor as follows

```
obj = vtkDataObject
```

31.43.2 Methods

The class vtkDataObject has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObject class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataObject = obj.NewInstance ()`
- `vtkDataObject = obj.SafeDownCast (vtkObject o)`
- `vtkSource = obj.GetSource ()` - Set/Get the source object creating this data object.
- `obj.SetSource (vtkSource s)` - Set/Get the source object creating this data object.
- `vtkInformation = obj.GetInformation ()` - Set/Get the information object associated with this data object.
- `obj.SetInformation (vtkInformation )` - Set/Get the information object associated with this data object.
- `vtkInformation = obj.GetPipelineInformation ()` - Get/Set the pipeline information object that owns this data object.
- `obj.SetPipelineInformation (vtkInformation )` - Get/Set the pipeline information object that owns this data object.
- `vtkAlgorithmOutput = obj.GetProducerPort ()` - Get the port currently producing this object.
- `long = obj.GetMTime ()` - Data objects are composite objects and need to check each part for MTime. The information object also needs to be considered.
- `obj.Initialize ()` - Restore data object to initial state,
- `obj.ReleaseData ()` - Release data back to system to conserve memory resource. Used during visualization network execution. Releasing this data does not make down-stream data invalid, so it does not modify the MTime of this data object.
- `int = obj.ShouldIReleaseData ()` - Return flag indicating whether data should be released after use by a filter.
- `int = obj.GetDataReleased ()` - Get the flag indicating the data has been released.
- `obj.SetReleaseDataFlag (int )` - Turn on/off flag to control whether this object's data is released after being used by a filter.
- `int = obj.GetReleaseDataFlag ()` - Turn on/off flag to control whether this object's data is released after being used by a filter.
- `obj.ReleaseDataFlagOn ()` - Turn on/off flag to control whether this object's data is released after being used by a filter.
- `obj.ReleaseDataFlagOff ()` - Turn on/off flag to control whether this object's data is released after being used by a filter.

- `obj.GlobalReleaseDataFlagOn ()` - Turn on/off flag to control whether every object releases its data after being used by a filter.
- `obj.GlobalReleaseDataFlagOff ()` - Turn on/off flag to control whether every object releases its data after being used by a filter.
- `obj.SetFieldData (vtkFieldData )` - Assign or retrieve a general field data to this data object.
- `vtkFieldData = obj.GetFieldData ()` - Assign or retrieve a general field data to this data object.
- `obj.Register (vtkObjectBase o)`
- `obj.UnRegister (vtkObjectBase o)`
- `obj.Update ()` - Provides opportunity for the data object to insure internal consistency before access. Also causes owning source/filter (if any) to update itself. The Update() method is composed of UpdateInformation(), PropagateUpdateExtent(), TriggerAsynchronousUpdate(), and UpdateData().
- `obj.UpdateInformation ()` - WARNING: INTERNAL METHOD - NOT FOR GENERAL USE. THIS METHOD IS PART OF THE PIPELINE UPDATE FUNCTIONALITY. Update all the "easy to update" information about the object such as the extent which will be used to control the update. This propagates all the way up then back down the pipeline. As a by-product the PipelineMTime is updated.
- `obj.PropagateUpdateExtent ()` - WARNING: INTERNAL METHOD - NOT FOR GENERAL USE. THIS METHOD IS PART OF THE PIPELINE UPDATE FUNCTIONALITY. The update extent for this object is propagated up the pipeline. This propagation may early terminate based on the PipelineMTime.
- `obj.TriggerAsynchronousUpdate ()` - WARNING: INTERNAL METHOD - NOT FOR GENERAL USE. THIS METHOD IS PART OF THE PIPELINE UPDATE FUNCTIONALITY. Propagate back up the pipeline for ports and trigger the update on the other side of the port to allow for asynchronous parallel processing in the pipeline. This propagation may early terminate based on the PipelineMTime.
- `obj.UpdateData ()` - WARNING: INTERNAL METHOD - NOT FOR GENERAL USE. THIS METHOD IS PART OF THE PIPELINE UPDATE FUNCTIONALITY. Propagate the update back up the pipeline, and perform the actual work of updating on the way down. When the propagate arrives at a port, block and wait for the asynchronous update to finish on the other side. This propagation may early terminate based on the PipelineMTime.
- `long = obj.GetEstimatedMemorySize ()` - Get the estimated size of this data object itself. Should be called after UpdateInformation() and PropagateUpdateExtent() have both been called. Should be overridden in a subclass - otherwise the default is to assume that this data object requires no memory. The size is returned in kilobytes.
- `obj.SetUpdateExtent (int piece, int numPieces, int ghostLevel)` - A generic way of specifying an update extent. Subclasses must decide what a piece is. When the NumberOfPieces is zero, then no data is requested, and the source will not execute.
- `obj.SetUpdateExtent (int piece, int numPieces)` - Set the update extent for data objects that use 3D extents. Using this method on data objects that set extents as pieces (such as vtkPolyData or vtkUnstructuredGrid) has no real effect. Don't use the set macro to set the update extent since we don't want this object to be modified just due to a change in update extent. When the volume of the extent is zero (0, -1,..), then no data is requested, and the source will not execute.
- `obj.SetUpdateExtent (int x0, int x1, int y0, int y1, int z0, int z1)` - Set the update extent for data objects that use 3D extents. Using this method on data objects that set extents as pieces (such as vtkPolyData or vtkUnstructuredGrid) has no real effect. Don't use the set macro to set the update extent since we don't want this object to be modified just due to a change in update extent.

When the volume of the extent is zero (0, -1,..), then no data is requested, and the source will not execute.

- `obj.SetUpdateExtent (int extent[6])` - Set the update extent for data objects that use 3D extents. Using this method on data objects that set extents as pieces (such as vtkPolyData or vtkUnstructuredGrid) has no real effect. Don't use the set macro to set the update extent since we don't want this object to be modified just due to a change in update extent. When the volume of the extent is zero (0, -1,..), then no data is requested, and the source will not execute.

- `int = obj.GetUpdateExtent ()` - Set the update extent for data objects that use 3D extents. Using this method on data objects that set extents as pieces (such as vtkPolyData or vtkUnstructuredGrid) has no real effect. Don't use the set macro to set the update extent since we don't want this object to be modified just due to a change in update extent. When the volume of the extent is zero (0, -1,..), then no data is requested, and the source will not execute.

- `obj.GetUpdateExtent (int extent[6])` - Set the update extent for data objects that use 3D extents. Using this method on data objects that set extents as pieces (such as vtkPolyData or vtkUnstructuredGrid) has no real effect. Don't use the set macro to set the update extent since we don't want this object to be modified just due to a change in update extent. When the volume of the extent is zero (0, -1,..), then no data is requested, and the source will not execute.

- `int = obj.GetDataObjectType ()` - Used by Threaded ports to determine if they should initiate an asynchronous update (still in development).

- `long = obj.GetUpdateTime ()` - Used by Threaded ports to determine if they should initiate an asynchronous update (still in development).

- `obj.SetUpdateExtentToWholeExtent ()` - If the whole input extent is required to generate the requested output extent, this method can be called to set the input update extent to the whole input extent. This method assumes that the whole extent is known (that UpdateInformation has been called)

- `long = obj.GetPipelineMTime ()` - Get the cumulative modified time of everything upstream. Does not include the MTime of this object.

- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value).

- `obj.CopyInformation (vtkDataObject data)` - Copy the generic information (WholeExtent ...)

- `obj.CopyTypeSpecificInformation (vtkDataObject data)` - By default, there is no type specific information

- `obj.SetUpdatePiece (int piece)` - Set / Get the update piece and the update number of pieces. Similar to update extent in 3D.

- `obj.SetUpdateNumberOfPieces (int num)` - Set / Get the update piece and the update number of pieces. Similar to update extent in 3D.

- `int = obj.GetUpdatePiece ()` - Set / Get the update piece and the update number of pieces. Similar to update extent in 3D.

- `int = obj.GetUpdateNumberOfPieces ()` - Set / Get the update piece and the update number of pieces. Similar to update extent in 3D.

- `obj.SetUpdateGhostLevel (int level)` - Set / Get the update ghost level and the update number of ghost levels. Similar to update extent in 3D.

- `int = obj.GetUpdateGhostLevel ()` - Set / Get the update ghost level and the update number of ghost levels. Similar to update extent in 3D.
- `obj.SetRequestExactExtent (int flag)` - This request flag indicates whether the requester can handle more data than requested. Right now it is used in vtkImageData. Image filters can return more data than requested. The the consumer cannot handle this (i.e. DataSetToDataSetFitler) the image will crop itself. This functionality used to be in ImageToStructuredPoints.
- `int = obj.GetRequestExactExtent ()` - This request flag indicates whether the requester can handle more data than requested. Right now it is used in vtkImageData. Image filters can return more data than requested. The the consumer cannot handle this (i.e. DataSetToDataSetFitler) the image will crop itself. This functionality used to be in ImageToStructuredPoints.
- `obj.RequestExactExtentOn ()` - This request flag indicates whether the requester can handle more data than requested. Right now it is used in vtkImageData. Image filters can return more data than requested. The the consumer cannot handle this (i.e. DataSetToDataSetFitler) the image will crop itself. This functionality used to be in ImageToStructuredPoints.
- `obj.RequestExactExtentOff ()` - This request flag indicates whether the requester can handle more data than requested. Right now it is used in vtkImageData. Image filters can return more data than requested. The the consumer cannot handle this (i.e. DataSetToDataSetFitler) the image will crop itself. This functionality used to be in ImageToStructuredPoints.
- `obj.SetWholeExtent (int x0, int x1, int y0, int y1, int z0, int z1)` - Set/Get the whole extent of this data object. The whole extent is meta data for structured data sets. It gets set by the source during the update information call.
- `obj.SetWholeExtent (int extent[6])` - Set/Get the whole extent of this data object. The whole extent is meta data for structured data sets. It gets set by the source during the update information call.
- `int = obj.GetWholeExtent ()` - Set/Get the whole extent of this data object. The whole extent is meta data for structured data sets. It gets set by the source during the update information call.
- `obj.GetWholeExtent (int extent[6])` - Set/Get the whole extent of this data object. The whole extent is meta data for structured data sets. It gets set by the source during the update information call.
- `obj.SetWholeBoundingBox (double x0, double x1, double y0, double y1, double z0, double z1)` - Set/Get the whole bounding box of this data object. The whole whole bounding box is meta data for data sets It gets set by the source during the update information call.
- `obj.SetWholeBoundingBox (double bb[6])` - Set/Get the whole bounding box of this data object. The whole whole bounding box is meta data for data sets It gets set by the source during the update information call.
- `double = obj.GetWholeBoundingBox ()` - Set/Get the whole bounding box of this data object. The whole whole bounding box is meta data for data sets It gets set by the source during the update information call.
- `obj.GetWholeBoundingBox (double extent[6])` - Set/Get the whole bounding box of this data object. The whole whole bounding box is meta data for data sets It gets set by the source during the update information call.
- `obj.SetMaximumNumberOfPieces (int )` - Set/Get the maximum number of pieces that can be requested. The maximum number of pieces is meta data for unstructured data sets. It gets set by the source during the update information call. A value of -1 indicates that there is no maximum. A value of

- `int = obj.GetMaximumNumberOfPieces ()` - Set/Get the maximum number of pieces that can be requested. The maximum number of pieces is meta data for unstructured data sets. It gets set by the source during the update information call. A value of -1 indicates that there is no maximum. A value of

- `obj.CopyInformationToPipeline (vtkInformation request, vtkInformation input, vtkInformation output,` - Copy information about this data object to the output information from its own Information for the given request. If the second argument is not NULL then it is the pipeline information object for the input to this data object's producer. If forceCopy is true, information is copied even if it exists in the output.

- `obj.CopyInformationToPipeline (vtkInformation request, vtkInformation input)` - Copy information about this data object from the PipelineInformation to its own Information for the given request.

- `obj.CopyInformationFromPipeline (vtkInformation request)` - Copy information about this data object from the PipelineInformation to its own Information for the given request.

- `obj.DataHasBeenGenerated ()` - This method is called by the source when it executes to generate data. It is sort of the opposite of ReleaseData. It sets the DataReleased flag to 0, and sets a new UpdateTime.

- `obj.PrepareForNewData ()` - make the output data ready for new data to be inserted. For most objects we just call Initialize. But for vtkImageData we leave the old data in case the memory can be reused.

- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy. These copy the data, but not any of the pipeline connections.

- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy. These copy the data, but not any of the pipeline connections.

- `obj.SetExtentTranslator (vtkExtentTranslator translator)` - An object that will translate pieces into structured extents.

- `vtkExtentTranslator = obj.GetExtentTranslator ()` - An object that will translate pieces into structured extents.

- `int = obj.GetExtentType ()` - The ExtentType will be left as VTK_PIECES_EXTENT for data objects such as vtkPolyData and vtkUnstructuredGrid. The ExtentType will be changed to VTK_3D_EXTENT for data objects with 3D structure such as vtkImageData (and its subclass vtkStructuredPoints), vtkRectilinearGrid, and vtkStructuredGrid. The default is the have an extent in pieces, with only one piece (no streaming possible).

- `obj.Crop ()` - This method crops the data object (if necesary) so that the extent matches the update extent.

- `vtkDataSetAttributes = obj.GetAttributes (int type)` - Returns the attributes of the data object of the specified attribute type. The type may be: ¡ul¿ ¡li¿POINT - Defined in vtkDataSet subclasses. ¡li¿CELL - Defined in vtkDataSet subclasses. ¡li¿VERTEX - Defined in vtkGraph subclasses. ¡li¿EDGE - Defined in vtkGraph subclasses. ¡li¿ROW - Defined in vtkTable. ¡/ul¿ The other attribute type, FIELD, will return NULL since field data is stored as a vtkFieldData instance, not a vtkDataSetAttributes instance. To retrieve field data, use GetAttributesAsFieldData.

- `vtkFieldData = obj.GetAttributesAsFieldData (int type)` - Returns the attributes of the data object as a vtkFieldData. This returns non-null values in all the same cases as GetAttributes, in addition to the case of FIELD, which will return the field data for any vtkDataObject subclass.

- `int = obj.GetAttributeTypeForArray (vtkAbstractArray arr)` - Retrieves the attribute type that an array came from. This is useful for obtaining which attribute type a input array to an algorithm came from (retrieved from GetInputAbstractArrayToProcesss).
- `vtkIdType = obj.GetNumberOfElements (int type)` - Get the number of elements for a specific attribute type (POINT, CELL, etc.).

31.44 vtkDataObjectAlgorithm

31.44.1 Usage

To create an instance of class vtkDataObjectAlgorithm, simply invoke its constructor as follows

```
obj = vtkDataObjectAlgorithm
```

31.44.2 Methods

The class vtkDataObjectAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObjectAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataObjectAlgorithm = obj.NewInstance ()`
- `vtkDataObjectAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkDataObject = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput ()`
- `vtkDataObject = obj.GetInput (int port)`
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.45 vtkDataObjectCollection

31.45.1 Usage

vtkDataObjectCollection is an object that creates and manipulates lists of data objects. See also vtkCollection and subclasses.

To create an instance of class vtkDataObjectCollection, simply invoke its constructor as follows

```
obj = vtkDataObjectCollection
```

31.45.2 Methods

The class vtkDataObjectCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObjectCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataObjectCollection = obj.NewInstance ()`
- `vtkDataObjectCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkDataObject ds)` - Get the next data object in the list.
- `vtkDataObject = obj.GetNextItem ()` - Get the ith data object in the list.
- `vtkDataObject = obj.GetItem (int i)`

31.46 vtkDataObjectSource

31.46.1 Usage

vtkDataObjectSource is an abstract object that specifies behavior and interface of field source objects. Field source objects are source objects that create vtkFieldData (field data) on output.

Concrete subclasses of vtkDataObjectSource must define Update() and Execute() methods. The public method Update() invokes network execution and will bring the network up-to-date. The protected Execute() method actually does the work of data creation/generation. The difference between the two methods is that Update() implements input consistency checks and modified time comparisons and then invokes the Execute() which is an implementation of a particular algorithm.

vtkDataObjectSource provides a mechanism for invoking the methods StartMethod() and EndMethod() before and after object execution (via Execute()). These are convenience methods you can use for any purpose (e.g., debugging info, highlighting/notifying user interface, etc.) These methods accept a single void* pointer that can be used to send data to the methods. It is also possible to specify a function to delete the argument via StartMethodArgDelete and EndMethodArgDelete.

Another method, ProgressMethod() can be specified. Some filters invoke this method periodically during their execution. The use is similar to that of StartMethod() and EndMethod().

An important feature of subclasses of vtkDataObjectSource is that it is possible to control the memory-management model (i.e., retain output versus delete output data). If enabled the ReleaseDataFlag enables the deletion of the output data once the downstream process object finishes processing the data (please see text).

To create an instance of class vtkDataObjectSource, simply invoke its constructor as follows

```
obj = vtkDataObjectSource
```

31.46.2 Methods

The class vtkDataObjectSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObjectSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataObjectSource = obj.NewInstance ()`
- `vtkDataObjectSource = obj.SafeDownCast (vtkObject o)`
- `vtkDataObject = obj.GetOutput ()` - Get the output field of this source.
- `vtkDataObject = obj.GetOutput (int idx)`
- `obj.SetOutput (vtkDataObject )`

31.47 vtkDataObjectTypes

31.47.1 Usage

vtkDataObjectTypes is a helper class that supports conversion between integer types defined in vtkType.h and string names as well as creation of data objects from either integer or string types. This class has to be updated every time a new data type is added to VTK.

To create an instance of class vtkDataObjectTypes, simply invoke its constructor as follows

```
obj = vtkDataObjectTypes
```

31.47.2 Methods

The class vtkDataObjectTypes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObjectTypes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataObjectTypes = obj.NewInstance ()`
- `vtkDataObjectTypes = obj.SafeDownCast (vtkObject o)`

31.48 vtkDataSet

31.48.1 Usage

vtkDataSet is an abstract class that specifies an interface for dataset objects. vtkDataSet also provides methods to provide informations about the data, such as center, bounding box, and representative length.

In vtk a dataset consists of a structure (geometry and topology) and attribute data. The structure is defined implicitly or explicitly as a collection of cells. The geometry of the structure is contained in point coordinates plus the cell interpolation functions. The topology of the dataset structure is defined by cell types and how the cells share their defining points.

Attribute data in vtk is either point data (data at points) or cell data (data at cells). Typically filters operate on point data, but some may operate on cell data, both cell and point data, either one, or none.

To create an instance of class vtkDataSet, simply invoke its constructor as follows

```
obj = vtkDataSet
```

31.48.2 Methods

The class vtkDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSet = obj.NewInstance ()`
- `vtkDataSet = obj.SafeDownCast (vtkObject o)`
- `obj.CopyStructure (vtkDataSet ds)` - Copy the geometric and topological structure of an object. Note that the invoking object and the object pointed to by the parameter ds must be of the same type. THIS METHOD IS NOT THREAD SAFE.
- `obj.CopyAttributes (vtkDataSet ds)` - Copy the attributes associated with the specified dataset to this instance of vtkDataSet. THIS METHOD IS NOT THREAD SAFE.
- `vtkIdType = obj.GetNumberOfPoints ()` - Determine the number of points composing the dataset. THIS METHOD IS THREAD SAFE
- `vtkIdType = obj.GetNumberOfCells ()` - Determine the number of cells composing the dataset. THIS METHOD IS THREAD SAFE
- `double = obj.GetPoint (vtkIdType ptId)` - Get point coordinates with ptId such that: 0 ¡= ptId ¡ NumberOfPoints. THIS METHOD IS NOT THREAD SAFE.
- `obj.GetPoint (vtkIdType id, double x[3])` - Copy point coordinates into user provided array x[3] for specified point id. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `vtkCell = obj.GetCell (vtkIdType cellId)` - Get cell with cellId such that: 0 ¡= cellId ¡ NumberOfCells. THIS METHOD IS NOT THREAD SAFE.
- `obj.GetCell (vtkIdType cellId, vtkGenericCell cell)` - Get cell with cellId such that: 0 ¡= cellId ¡ NumberOfCells. This is a thread-safe alternative to the previous GetCell() method. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `obj.GetCellBounds (vtkIdType cellId, double bounds[6])` - Get the bounds of the cell with cellId such that: 0 ¡= cellId ¡ NumberOfCells. A subclass may be able to determine the bounds of cell without using an expensive GetCell() method. A default implementation is provided that actually uses a GetCell() call. This is to ensure the method is available to all datasets. Subclasses should override this method to provide an efficient implementation. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `int = obj.GetCellType (vtkIdType cellId)` - Get type of cell with cellId such that: 0 ¡= cellId ¡ NumberOfCells. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `obj.GetCellTypes (vtkCellTypes types)` - Get a list of types of cells in a dataset. The list consists of an array of types (not necessarily in any order), with a single entry per type. For example a dataset 5 triangles, 3 lines, and 100 hexahedra would result a list of three entries, corresponding to the types VTK_TRIANGLE, VTK_LINE, and VTK_HEXAHEDRON. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `obj.GetCellPoints (vtkIdType cellId, vtkIdList ptIds)` - Topological inquiry to get points defining cell. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `obj.GetPointCells (vtkIdType ptId, vtkIdList cellIds)` - Topological inquiry to get cells using point. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `obj.GetCellNeighbors (vtkIdType cellId, vtkIdList ptIds, vtkIdList cellIds)` - Topological inquiry to get all cells using list of points exclusive of cell specified (e.g., cellId). Note that the list consists of only cells that use ALL the points provided. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `vtkIdType = obj.FindPoint (double x, double y, double z)` - Locate the closest point to the global coordinate x. Return the point id. If point id ¡ 0; then no point found. (This may arise when point is outside of dataset.) THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `vtkIdType = obj.FindPoint (double x[3])` - Locate the closest point to the global coordinate x. Return the point id. If point id ¡ 0; then no point found. (This may arise when point is outside of dataset.) THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `long = obj.GetMTime ()` - Datasets are composite objects and need to check each part for MTime THIS METHOD IS THREAD SAFE

- `vtkCellData = obj.GetCellData ()` - Return a pointer to this dataset's cell data. THIS METHOD IS THREAD SAFE

- `vtkPointData = obj.GetPointData ()` - Return a pointer to this dataset's point data. THIS METHOD IS THREAD SAFE

- `obj.Squeeze ()` - Reclaim any extra memory used to store data. THIS METHOD IS NOT THREAD SAFE.

- `obj.ComputeBounds ()` - Compute the data bounding box from data points. THIS METHOD IS NOT THREAD SAFE.

- `double = obj.GetBounds ()` - Return a pointer to the geometry bounding box in the form (xmin,xmax, ymin,ymax, zmin,zmax). THIS METHOD IS NOT THREAD SAFE.

- `obj.GetBounds (double bounds[6])` - Return a pointer to the geometry bounding box in the form (xmin,xmax, ymin,ymax, zmin,zmax). THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `double = obj.GetCenter ()` - Get the center of the bounding box. THIS METHOD IS NOT THREAD SAFE.

- `obj.GetCenter (double center[3])` - Get the center of the bounding box. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `double = obj.GetLength ()` - Return the length of the diagonal of the bounding box. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `obj.Initialize ()` - Restore data object to initial state, THIS METHOD IS NOT THREAD SAFE.

- `obj.GetScalarRange (double range[2])` - Convenience method to get the range of the scalar data (if there is any scalar data). Returns the (min/max) range of combined point and cell data. If there are no point or cell scalars the method will return (0,1). Note: Update needs to be called to create the scalars. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED

- `double = obj.GetScalarRange ()` - Convenience method to get the range of the scalar data (if there is any scalar data). THIS METHOD IS NOT THREAD SAFE.

- `int = obj.GetMaxCellSize ()` - Convenience method returns largest cell size in dataset. This is generally used to allocate memory for supporting data structures. THIS METHOD IS THREAD SAFE

- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value). THIS METHOD IS THREAD SAFE.

- `int = obj.GetDataObjectType ()` - Shallow and Deep copy.

- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.

- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.

- `int = obj.CheckAttributes ()` - This method checks to see if the cell and point attributes match the geometry. Many filters will crash if the number of tupples in an array is less than the number of points/cells. This method returns 1 if there is a mismatch, and 0 if everything is ok. It prints an error if an array is too short, and a warning if an array is too long.

- `obj.GenerateGhostLevelArray ()` - Normally called by pipeline executives or algoritgms only. This method computes the ghost arrays for a given dataset.

- `vtkFieldData = obj.GetAttributesAsFieldData (int type)` - Returns the attributes of the data object as a vtkFieldData. This returns non-null values in all the same cases as GetAttributes, in addition to the case of FIELD, which will return the field data for any vtkDataObject subclass.

- `vtkIdType = obj.GetNumberOfElements (int type)` - Get the number of elements for a specific attribute type (POINT, CELL, etc.).

31.49 vtkDataSetAlgorithm

31.49.1 Usage

vtkDataSetAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline architecture. Ther are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this classes contstructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be DataSet. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into seperate functions such as RequestDataObject RequestData and RequestInformation. The default implementation of RequestDataObject will create an output data of the same type as the input.

To create an instance of class vtkDataSetAlgorithm, simply invoke its constructor as follows

```
obj = vtkDataSetAlgorithm
```

31.49.2 Methods

The class vtkDataSetAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetAlgorithm = obj.NewInstance ()`
- `vtkDataSetAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkDataSet = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkDataSet = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput ()` - Get the input data object. This method is not recommended for use, but lots of old style filters use it.
- `vtkPolyData = obj.GetPolyDataOutput ()` - Get the output as vtkPolyData.
- `vtkStructuredPoints = obj.GetStructuredPointsOutput ()` - Get the output as vtkStructuredPoints.
- `vtkImageData = obj.GetImageDataOutput ()` - Get the output as vtkStructuredPoints.
- `vtkStructuredGrid = obj.GetStructuredGridOutput ()` - Get the output as vtkStructuredGrid.
- `vtkUnstructuredGrid = obj.GetUnstructuredGridOutput ()` - Get the output as vtkUnstructuredGrid.
- `vtkRectilinearGrid = obj.GetRectilinearGridOutput ()` - Get the output as vtkRectilinearGrid.
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (vtkDataSet )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataSet )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (vtkDataSet )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataSet )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.50 vtkDataSetAttributes

31.50.1 Usage

vtkDataSetAttributes is a class that is used to represent and manipulate attribute data (e.g., scalars, vectors, normals, texture coordinates, tensors, global ids, pedigree ids, and field data).

This adds to vtkFieldData the ability to pick one of the arrays from the field as the currently active array for each attribute type. In other words, you pick one array to be called "THE" Scalars, and then filters down the pipeline will treat that array specially. For example vtkContourFilter will contour "THE" Scalar array unless a different array is asked for.

Additionally vtkDataSetAttributes provides methods that filters call to pass data through, copy data into, and interpolate from Fields. PassData passes entire arrays from the source to the destination. Copy passes through some subset of the tuples from the source to the destination. Interpolate interpolates from the chosen tuple(s) in the source data, using the provided weights, to produce new tuples in the destination. Each attribute type has pass, copy and interpolate "copy" flags that can be set in the destination to choose which attribute arrays will be transfered from the source to the destination.

Finally this class provides a mechanism to determine which attributes a group of sources have in common, and to copy tuples from a source into the destination, for only those attributes that are held by all.

To create an instance of class vtkDataSetAttributes, simply invoke its constructor as follows

```
obj = vtkDataSetAttributes
```

31.50.2 Methods

The class vtkDataSetAttributes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetAttributes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetAttributes = obj.NewInstance ()`
- `vtkDataSetAttributes = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Initialize all of the object's data to NULL Also, clear the copy flags.
- `obj.Update ()` - Deep copy of data (i.e., create new data arrays and copy from input data). Ignores the copy flags but preserves them in the output.

- `obj.DeepCopy (vtkFieldData pd)` - Deep copy of data (i.e., create new data arrays and copy from input data). Ignores the copy flags but preserves them in the output.
- `obj.ShallowCopy (vtkFieldData pd)` - Shallow copy of data (i.e., use reference counting). Ignores the copy flags but preserves them in the output.
- `int = obj.SetScalars (vtkDataArray da)` - Set/Get the scalar data.
- `int = obj.SetActiveScalars (string name)` - Set/Get the scalar data.
- `vtkDataArray = obj.GetScalars ()` - Set/Get the scalar data.
- `int = obj.SetVectors (vtkDataArray da)` - Set/Get the vector data.
- `int = obj.SetActiveVectors (string name)` - Set/Get the vector data.
- `vtkDataArray = obj.GetVectors ()` - Set/Get the vector data.
- `int = obj.SetNormals (vtkDataArray da)` - Set/get the normal data.
- `int = obj.SetActiveNormals (string name)` - Set/get the normal data.
- `vtkDataArray = obj.GetNormals ()` - Set/get the normal data.
- `int = obj.SetTCoords (vtkDataArray da)` - Set/Get the texture coordinate data.
- `int = obj.SetActiveTCoords (string name)` - Set/Get the texture coordinate data.
- `vtkDataArray = obj.GetTCoords ()` - Set/Get the texture coordinate data.
- `int = obj.SetTensors (vtkDataArray da)` - Set/Get the tensor data.
- `int = obj.SetActiveTensors (string name)` - Set/Get the tensor data.
- `vtkDataArray = obj.GetTensors ()` - Set/Get the tensor data.
- `int = obj.SetGlobalIds (vtkDataArray da)` - Set/Get the global id data.
- `int = obj.SetActiveGlobalIds (string name)` - Set/Get the global id data.
- `vtkDataArray = obj.GetGlobalIds ()` - Set/Get the global id data.
- `int = obj.SetPedigreeIds (vtkAbstractArray da)` - Set/Get the pedigree id data.
- `int = obj.SetActivePedigreeIds (string name)` - Set/Get the pedigree id data.
- `vtkAbstractArray = obj.GetPedigreeIds ()` - Set/Get the pedigree id data.
- `vtkDataArray = obj.GetScalars (string name)` - This will first look for an array with the correct name. If one exists, it is returned. Otherwise, the name argument is ignored, and the active attribute is returned.
- `vtkDataArray = obj.GetVectors (string name)` - This will first look for an array with the correct name. If one exists, it is returned. Otherwise, the name argument is ignored, and the active attribute is returned.
- `vtkDataArray = obj.GetNormals (string name)` - This will first look for an array with the correct name. If one exists, it is returned. Otherwise, the name argument is ignored, and the active attribute is returned.
- `vtkDataArray = obj.GetTCoords (string name)` - This will first look for an array with the correct name. If one exists, it is returned. Otherwise, the name argument is ignored, and the active attribute is returned.

- `vtkDataArray = obj.GetTensors (string name)` - This will first look for an array with the correct name. If one exists, it is returned. Otherwise, the name argument is ignored, and the active attribute is returned.

- `vtkDataArray = obj.GetGlobalIds (string name)` - This will first look for an array with the correct name. If one exists, it is returned. Otherwise, the name argument is ignored, and the active attribute is returned.

- `vtkAbstractArray = obj.GetPedigreeIds (string name)` - This will first look for an array with the correct name. If one exists, it is returned. Otherwise, the name argument is ignored, and the active attribute is returned.

- `int = obj.SetActiveAttribute (string name, int attributeType)` - Make the array with the given name the active attribute. Attribute types are: vtkDataSetAttributes::SCALARS = 0 vtkDataSetAttributes::VECTORS = 1 vtkDataSetAttributes::NORMALS = 2 vtkDataSetAttributes::TCOORDS = 3 vtkDataSetAttributes::TENSORS = 4 vtkDataSetAttributes::GLOBALIDS = 5 vtkDataSetAttributes::PEDIGREEIDS = 6 Returns the index of the array if succesful, -1 if the array is not in the list of arrays.

- `int = obj.SetActiveAttribute (int index, int attributeType)` - Make the array with the given index the active attribute.

- `obj.GetAttributeIndices (int indexArray)` - Get the field data array indices corresponding to scalars, vectors, tensors, etc.

- `int = obj.IsArrayAnAttribute (int idx)` - Determine whether a data array of index idx is considered a data set attribute (i.e., scalar, vector, tensor, etc). Return less-than zero if it is, otherwise an index 0¡=idx¡NUM_ATTRIBUTES to indicate which attribute.

- `vtkDataArray = obj.GetAttribute (int attributeType)` - Return an attribute given the attribute type (see vtkDataSetAttributes::AttributeTypes). Some attributes (such as PEDIGREEIDS) may not be vtkDataArray subclass, so in that case use GetAbstractAttribute().

- `vtkAbstractArray = obj.GetAbstractAttribute (int attributeType)` - Return an attribute given the attribute type (see vtkDataSetAttributes::AttributeTypes). This is the same as GetAttribute(), except that the returned array is a vtkAbstractArray instead of vtkDataArray. Some attributes (such as PEDIGREEIDS) may not be vtkDataArray subclass.

- `obj.RemoveArray (string name)` - Remove an array (with the given name) from the list of arrays.

- `obj.SetCopyAttribute (int index, int value, int ctypeALLCOPY)` - Specify whether to copy the data attribute referred to by index. ctype selects from the AttributeCopyOperations. If ctype is set to ALLCOPY, then COPYTUPLE, INTERPOLATE, and PASSDATA are set to value. If value is 0, copying is disallowed. otherwise it is allowed.

- `obj.SetCopyScalars (int i, int ctypeALLCOPY)` - Turn on/off the copying of scalar data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `int = obj.GetCopyScalars (int ctypeALLCOPY)` - Turn on/off the copying of scalar data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyScalarsOn ()` - Turn on/off the copying of scalar data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyScalarsOff ()` - Turn on/off the copying of scalar data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.SetCopyVectors (int i, int ctypeALLCOPY)` - Turn on/off the copying of vector data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `int = obj.GetCopyVectors (int ctypeALLCOPY)` - Turn on/off the copying of vector data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyVectorsOn ()` - Turn on/off the copying of vector data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyVectorsOff ()` - Turn on/off the copying of vector data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.SetCopyNormals (int i, int ctypeALLCOPY)` - Turn on/off the copying of normals data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `int = obj.GetCopyNormals (int ctypeALLCOPY)` - Turn on/off the copying of normals data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyNormalsOn ()` - Turn on/off the copying of normals data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyNormalsOff ()` - Turn on/off the copying of normals data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.SetCopyTCoords (int i, int ctypeALLCOPY)` - Turn on/off the copying of texture coordinates data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `int = obj.GetCopyTCoords (int ctypeALLCOPY)` - Turn on/off the copying of texture coordinates data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyTCoordsOn ()` - Turn on/off the copying of texture coordinates data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyTCoordsOff ()` - Turn on/off the copying of texture coordinates data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.SetCopyTensors (int i, int ctypeALLCOPY)` - Turn on/off the copying of tensor data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `int = obj.GetCopyTensors (int ctypeALLCOPY)` - Turn on/off the copying of tensor data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyTensorsOn ()` - Turn on/off the copying of tensor data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyTensorsOff ()` - Turn on/off the copying of tensor data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.SetCopyGlobalIds (int i, int ctypeALLCOPY)` - Turn on/off the copying of global id data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `int = obj.GetCopyGlobalIds (int ctypeALLCOPY)` - Turn on/off the copying of global id data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyGlobalIdsOn ()` - Turn on/off the copying of global id data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyGlobalIdsOff ()` - Turn on/off the copying of global id data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.SetCopyPedigreeIds (int i, int ctypeALLCOPY)` - Turn on/off the copying of pedigree id data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `int = obj.GetCopyPedigreeIds (int ctypeALLCOPY)` - Turn on/off the copying of pedigree id data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyPedigreeIdsOn ()` - Turn on/off the copying of pedigree id data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyPedigreeIdsOff ()` - Turn on/off the copying of pedigree id data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyAllOn (int ctypeALLCOPY)` - Turn on copying of all data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.CopyAllOff (int ctypeALLCOPY)` - Turn off copying of all data. ctype is one of the AttributeCopyOperations, and controls copy, interpolate and passdata behavior. For set, ctype=ALLCOPY means set all three flags to the same value. For get, ctype=ALLCOPY returns true only if all three flags are true.

  During copying, interpolation and passdata, the following rules are followed for each array: 1. If the copy/interpolate/pass flag for an attribute is set (on or off), it is applied. This overrides rules 2 and 3. 2. If the copy flag for an array is set (on or off), it is applied This overrides rule 3. 3. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

- `obj.PassData (vtkFieldData fd)` - Pass entire arrays of input data through to output. Obey the "copy" flags. When passing a field, the following copying rules are followed: 1) Check if a field is an attribute, if yes and if there is a PASSDATA copy flag for that attribute (on or off), obey the flag for that attribute, ignore (2) and (3), 2) if there is a copy field for that field (on or off), obey the flag, ignore (3) 3) obey CopyAllOn/Off

- `obj.CopyAllocate (vtkDataSetAttributes pd, vtkIdType sze, vtkIdType ext)` - Allocates point data for point-by-point (or cell-by-cell) copy operation. If sze=0, then use the input DataSetAttributes to create (i.e., find initial size of) new objects; otherwise use the sze variable. Note that pd HAS to be the vtkDataSetAttributes object which will later be used with CopyData. If this is not the case, consider using the alternative forms of CopyAllocate and CopyData. ext is no longer used. If shallowCopyArrays is true, input arrays are copied to the output instead of new ones being allocated.

- `obj.CopyAllocate (vtkDataSetAttributes pd, vtkIdType sze, vtkIdType ext, int shallowCopyArrays)` - Allocates point data for point-by-point (or cell-by-cell) copy operation. If sze=0, then use the input DataSetAttributes to create (i.e., find initial size of) new objects; otherwise use the sze variable. Note that pd HAS to be the vtkDataSetAttributes object which will later be used with CopyData. If this is not the case, consider using the alternative forms of CopyAllocate and CopyData. ext is no longer used. If shallowCopyArrays is true, input arrays are copied to the output instead of new ones being allocated.

- `obj.CopyStructuredData (vtkDataSetAttributes inDsa, int inExt, int outExt)` - This method is used to copy data arrays in images. You should call "CopyAllocate" before calling this method.

- `obj.CopyData (vtkDataSetAttributes fromPd, vtkIdType fromId, vtkIdType toId)` - Copy the attribute data from one id to another. Make sure CopyAllocate() has been invoked before using this method. When copying a field, the following copying rules are followed: 1) Check if a field is an attribute, if yes and if there is a COPYTUPLE copy flag for that attribute (on or off), obey the flag for that attribute, ignore (2) and (3), 2) if there is a copy field for that field (on or off), obey the flag, ignore (3) 3) obey CopyAllOn/Off

- `obj.CopyTuple (vtkAbstractArray fromData, vtkAbstractArray toData, vtkIdType fromId, vtkIdType toId` - Copy a tuple of data from one data array to another. This method assumes that the fromData and toData objects are of the same type, and have the same number of components. This is true if you invoke CopyAllocate() or InterpolateAllocate().

- `obj.InterpolateAllocate (vtkDataSetAttributes pd, vtkIdType sze, vtkIdType ext)` - Initialize point interpolation method. Note that pd HAS to be the vtkDataSetAttributes object which will later be used with InterpolatePoint or InterpolateEdge. ext is no longer used. If shallowCopyArrays is true, input arrays are copied to the output instead of new ones being allocated.

- `obj.InterpolateAllocate (vtkDataSetAttributes pd, vtkIdType sze, vtkIdType ext, int shallowCopyArra` - Initialize point interpolation method. Note that pd HAS to be the vtkDataSetAttributes object which will later be used with InterpolatePoint or InterpolateEdge. ext is no longer used. If shallowCopyArrays is true, input arrays are copied to the output instead of new ones being allocated.

- `obj.InterpolatePoint (vtkDataSetAttributes fromPd, vtkIdType toId, vtkIdList ids, double weights)` - Interpolate data set attributes from other data set attributes given cell or point ids and associated interpolation weights. If the INTERPOLATION copy flag is set to 0 for an array, interpolation is prevented. If the flag is set to 1, weighted interpolation occurs. If the flag is set to 2, nearest neighbor interpolation is used.

- `obj.InterpolateEdge (vtkDataSetAttributes fromPd, vtkIdType toId, vtkIdType p1, vtkIdType p2, doubl` - Interpolate data from the two points p1,p2 (forming an edge) and an interpolation factor, t, along the edge. The weight ranges from (0,1), with t=0 located at p1. Make sure that the method InterpolateAllocate() has been invoked before using this method. If the INTERPOLATION copy flag is set to 0 for an array, interpolation is prevented. If the flag is set to 1, weighted interpolation occurs. If the flag is set to 2, nearest neighbor interpolation is used.

- `obj.InterpolateTime (vtkDataSetAttributes from1, vtkDataSetAttributes from2, vtkIdType id, double t` - Interpolate data from the same id (point or cell) at different points in time (parameter t). Two input data set attributes objects are input. The parameter t lies between (0¡=t¡=1). IMPORTANT: it is assumed that the number of attributes and number of components is the same for both from1 and from2, and the type of data for from1 and from2 are the same. Make sure that the method InterpolateAllocate() has been invoked before using this method. If the INTERPOLATION copy flag is set to 0 for an array, interpolation is prevented. If the flag is set to 1, weighted interpolation occurs. If the flag is set to 2, nearest neighbor interpolation is used.

31.51 vtkDataSetCollection

31.51.1 Usage

vtkDataSetCollection is an object that creates and manipulates lists of datasets. See also vtkCollection and subclasses.

To create an instance of class vtkDataSetCollection, simply invoke its constructor as follows

```
obj = vtkDataSetCollection
```

31.51.2 Methods

The class vtkDataSetCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetCollection = obj.NewInstance ()`
- `vtkDataSetCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkDataSet ds)` - Get the next dataset in the list.
- `vtkDataSet = obj.GetNextItem ()` - Get the next dataset in the list.
- `vtkDataSet = obj.GetNextDataSet ()` - Get the next dataset in the list.
- `vtkDataSet = obj.GetItem (int i)` - Get the ith dataset in the list.
- `vtkDataSet = obj.GetDataSet (int i)` - Get the ith dataset in the list.

31.52 vtkDataSetSource

31.52.1 Usage

vtkDataSetSource is an abstract class whose subclasses generate datasets.

To create an instance of class vtkDataSetSource, simply invoke its constructor as follows

```
obj = vtkDataSetSource
```

31.52.2 Methods

The class vtkDataSetSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetSource = obj.NewInstance ()`
- `vtkDataSetSource = obj.SafeDownCast (vtkObject o)`
- `vtkDataSet = obj.GetOutput ()` - Get the output of this source.
- `vtkDataSet = obj.GetOutput (int idx)` - Get the output of this source.
- `obj.SetOutput (vtkDataSet )`

31.53 vtkDataSetToDataSetFilter

31.53.1 Usage

vtkDataSetToDataSetFilter is an abstract filter class. Subclasses of vtkDataSetToDataSetFilter take a dataset as input and create a dataset as output. The form of the input geometry is not changed in these filters, only the point attributes (e.g. scalars, vectors, etc.).

This is an abstract filter type. What that means is that the output of the filter is an abstract type (i.e., vtkDataSet), no matter what the input of the filter is. This can cause problems connecting together filters due to the change in dataset type. (For example, in a series of filters processing vtkPolyData, when a vtkDataSetToDataSetFilter or subclass is introduced into the pipeline, if the filter downstream of it takes vtkPolyData as input, the pipeline connection cannot be made.) To get around this problem, use one of the convenience methods to return a concrete type (e.g., vtkGetPolyDataOutput(), GetStructuredPointsOutput(), etc.).

To create an instance of class vtkDataSetToDataSetFilter, simply invoke its constructor as follows

```
obj = vtkDataSetToDataSetFilter
```

31.53.2 Methods

The class vtkDataSetToDataSetFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetToDataSetFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetToDataSetFilter = obj.NewInstance ()`
- `vtkDataSetToDataSetFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkDataSet input)` - Specify the input data or filter.
- `vtkDataSet = obj.GetOutput ()` - Get the output of this filter. If output is NULL then input hasn't been set which is necessary for abstract objects.
- `vtkDataSet = obj.GetOutput (int idx)` - Get the output of this filter. If output is NULL then input hasn't been set which is necessary for abstract objects.
- `vtkPolyData = obj.GetPolyDataOutput ()` - Get the output as vtkPolyData.
- `vtkStructuredPoints = obj.GetStructuredPointsOutput ()` - Get the output as vtkStructuredPoints.
- `vtkStructuredGrid = obj.GetStructuredGridOutput ()` - Get the output as vtkStructuredGrid.
- `vtkUnstructuredGrid = obj.GetUnstructuredGridOutput ()` - Get the output as vtkUnstructuredGrid.
- `vtkRectilinearGrid = obj.GetRectilinearGridOutput ()` - Get the output as vtkRectilinearGrid.
- `vtkDataSet = obj.GetInput ()` - Get the input data or filter.
- `obj.ComputeInputUpdateExtents (vtkDataObject output)` - By default copy the output update extent to the input

31.54 vtkDataSetToImageFilter

31.54.1 Usage

vtkDataSetToImageFilter is an abstract filter class whose subclasses take as input any dataset and generate image data on output.

To create an instance of class vtkDataSetToImageFilter, simply invoke its constructor as follows

```
obj = vtkDataSetToImageFilter
```

31.54.2 Methods

The class vtkDataSetToImageFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetToImageFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetToImageFilter = obj.NewInstance ()`
- `vtkDataSetToImageFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkDataSet input)` - Set / get the input data or filter.
- `vtkDataSet = obj.GetInput ()` - Set / get the input data or filter.

31.55 vtkDataSetToPolyDataFilter

31.55.1 Usage

vtkDataSetToPolyDataFilter is an abstract filter class whose subclasses take as input any dataset and generate polygonal data on output.

To create an instance of class vtkDataSetToPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkDataSetToPolyDataFilter
```

31.55.2 Methods

The class vtkDataSetToPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetToPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetToPolyDataFilter = obj.NewInstance ()`
- `vtkDataSetToPolyDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkDataSet input)` - Set / get the input data or filter.
- `vtkDataSet = obj.GetInput ()` - Set / get the input data or filter.
- `obj.ComputeInputUpdateExtents (vtkDataObject output)` - Do not let images return more than requested.

31.56 vtkDataSetToStructuredGridFilter

31.56.1 Usage

vtkDataSetToStructuredGridFilter is an abstract filter class whose subclasses take as input any dataset and generate a structured grid on output.

To create an instance of class vtkDataSetToStructuredGridFilter, simply invoke its constructor as follows

```
obj = vtkDataSetToStructuredGridFilter
```

31.56.2 Methods

The class vtkDataSetToStructuredGridFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetToStructuredGridFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetToStructuredGridFilter = obj.NewInstance ()`
- `vtkDataSetToStructuredGridFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkDataSet input)` - Set / get the input data or filter.
- `vtkDataSet = obj.GetInput ()` - Set / get the input data or filter.

31.57 vtkDataSetToStructuredPointsFilter

31.57.1 Usage

vtkDataSetToStructuredPointsFilter is an abstract filter class whose subclasses take as input any dataset and generate structured points data on output.

To create an instance of class vtkDataSetToStructuredPointsFilter, simply invoke its constructor as follows

```
obj = vtkDataSetToStructuredPointsFilter
```

31.57.2 Methods

The class vtkDataSetToStructuredPointsFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetToStructuredPointsFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetToStructuredPointsFilter = obj.NewInstance ()`
- `vtkDataSetToStructuredPointsFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkDataSet input)` - Set / get the input data or filter.
- `vtkDataSet = obj.GetInput ()` - Set / get the input data or filter.

31.58 vtkDataSetToUnstructuredGridFilter

31.58.1 Usage

vtkDataSetToUnstructuredGridFilter is an abstract filter class whose subclasses take as input any dataset and generate an unstructured grid on output.

To create an instance of class vtkDataSetToUnstructuredGridFilter, simply invoke its constructor as follows

```
obj = vtkDataSetToUnstructuredGridFilter
```

31.58.2 Methods

The class vtkDataSetToUnstructuredGridFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetToUnstructuredGridFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetToUnstructuredGridFilter = obj.NewInstance ()`
- `vtkDataSetToUnstructuredGridFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkDataSet input)` - Set / get the input data or filter.
- `vtkDataSet = obj.GetInput ()` - Set / get the input data or filter.

31.59 vtkDemandDrivenPipeline

31.59.1 Usage

vtkDemandDrivenPipeline is an executive that will execute an algorithm only when its outputs are out-of-date with respect to its inputs.

To create an instance of class vtkDemandDrivenPipeline, simply invoke its constructor as follows

```
obj = vtkDemandDrivenPipeline
```

31.59.2 Methods

The class vtkDemandDrivenPipeline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDemandDrivenPipeline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDemandDrivenPipeline = obj.NewInstance ()`
- `vtkDemandDrivenPipeline = obj.SafeDownCast (vtkObject o)`
- `int = obj.Update ()` - Bring the algorithm's outputs up-to-date. Returns 1 for success and 0 for failure.
- `int = obj.Update (int port)` - Bring the algorithm's outputs up-to-date. Returns 1 for success and 0 for failure.

- `long = obj.GetPipelineMTime ()` - Get the PipelineMTime for this exective.
- `int = obj.SetReleaseDataFlag (int port, int n)` - Set whether the given output port releases data when it is consumed. Returns 1 if the the value changes and 0 otherwise.
- `int = obj.GetReleaseDataFlag (int port)` - Get whether the given output port releases data when it is consumed.
- `int = obj.UpdatePipelineMTime ()` - Bring the PipelineMTime up to date.
- `int = obj.UpdateDataObject ()` - Bring the output data object's existence up to date. This does not actually produce data, but does create the data object that will store data produced during the UpdateData step.
- `int = obj.UpdateInformation ()` - Bring the output information up to date.
- `int = obj.UpdateData (int outputPort)` - Bring the output data up to date. This should be called only when information is up to date. Use the Update method if it is not known that the information is up to date.

31.60 vtkDirectedAcyclicGraph

31.60.1 Usage

vtkDirectedAcyclicGraph is a connected directed graph with no cycles. A tree is a type of directed graph, so works with all graph algorithms.

vtkDirectedAcyclicGraph is a read-only data structure. To construct a tree, create an instance of vtkMutableDirectedGraph. Add vertices and edges with AddVertex() and AddEdge(). You may alternately start by adding a single vertex as the root then call graph-¿AddChild(parent) which adds a new vertex and connects the parent to the child. The tree MUST have all edges in the proper direction, from parent to child. After building the tree, call tree-¿CheckedShallowCopy(graph) to copy the structure into a vtkDirectedAcyclicGraph. This method will return false if the graph is an invalid tree.

vtkDirectedAcyclicGraph provides some convenience methods for obtaining the parent and children of a vertex, for finding the root, and determining if a vertex is a leaf (a vertex with no children).

To create an instance of class vtkDirectedAcyclicGraph, simply invoke its constructor as follows

```
obj = vtkDirectedAcyclicGraph
```

31.60.2 Methods

The class vtkDirectedAcyclicGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDirectedAcyclicGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDirectedAcyclicGraph = obj.NewInstance ()`
- `vtkDirectedAcyclicGraph = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()`

31.61 vtkDirectedGraph

31.61.1 Usage

vtkDirectedGraph is a collection of vertices along with a collection of directed edges (edges that have a source and target). ShallowCopy() and DeepCopy() (and CheckedShallowCopy(), CheckedDeepCopy()) accept instances of vtkTree and vtkMutableDirectedGraph.

vtkDirectedGraph is read-only. To create an undirected graph, use an instance of vtkMutableDirectedGraph, then you may set the structure to a vtkDirectedGraph using ShallowCopy().

To create an instance of class vtkDirectedGraph, simply invoke its constructor as follows

```
obj = vtkDirectedGraph
```

31.61.2 Methods

The class vtkDirectedGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDirectedGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDirectedGraph = obj.NewInstance ()`
- `vtkDirectedGraph = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()`

31.62 vtkDirectedGraphAlgorithm

31.62.1 Usage

vtkDirectedGraphAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline edgehitecture. There are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this class constructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be Graph. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into separate functions such as ExecuteData and ExecuteInformation. For new algorithms you should implement RequestData(request, inputVec, outputVec) but for older filters there is a default implementation that calls the old ExecuteData(output) signature. For even older filters that don't implement ExecuteData the default implementation calls the even older Execute() signature.

.SECTION Thanks Thanks to Patricia Crossno, Ken Moreland, Andrew Wilson and Brian Wylie from Sandia National Laboratories for their help in developing this class.

To create an instance of class vtkDirectedGraphAlgorithm, simply invoke its constructor as follows

```
obj = vtkDirectedGraphAlgorithm
```

31.62.2 Methods

The class vtkDirectedGraphAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDirectedGraphAlgorithm class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkDirectedGraphAlgorithm = obj.NewInstance ()`
- `vtkDirectedGraphAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkDirectedGraph = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkDirectedGraph = obj.GetOutput (int index)` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int index, vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.63 vtkDiscretizableColorTransferFunction

31.63.1 Usage

This is a cross between a vtkColorTransferFunction and a vtkLookupTable selectively combiniting the functionality of both. NOTE: One must call Build() after making any changes to the points in the ColorTransferFunction to ensure that the discrete and non-discrete version match up.

To create an instance of class vtkDiscretizableColorTransferFunction, simply invoke its constructor as follows

```
obj = vtkDiscretizableColorTransferFunction
```

31.63.2 Methods

The class vtkDiscretizableColorTransferFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDiscretizableColorTransferFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDiscretizableColorTransferFunction = obj.NewInstance ()`
- `vtkDiscretizableColorTransferFunction = obj.SafeDownCast (vtkObject o)`
- `obj.Build ()` - Generate discretized lookup table, if applicable. This method must be called after changes to the ColorTransferFunction otherwise the discretized version will be inconsitent with the non-discretized one.
- `obj.SetDiscretize (int )` - Set if the values are to mapped after discretization. The number of discrete values is set by using SetNumberOfValues(). Not set by default, i.e. color value is determined by interpolating at the scalar value.

- `int = obj.GetDiscretize ()` - Set if the values are to mapped after discretization. The number of discrete values is set by using SetNumberOfValues(). Not set by default, i.e. color value is determined by interpolating at the scalar value.
- `obj.DiscretizeOn ()` - Set if the values are to mapped after discretization. The number of discrete values is set by using SetNumberOfValues(). Not set by default, i.e. color value is determined by interpolating at the scalar value.
- `obj.DiscretizeOff ()` - Set if the values are to mapped after discretization. The number of discrete values is set by using SetNumberOfValues(). Not set by default, i.e. color value is determined by interpolating at the scalar value.
- `obj.SetUseLogScale (int useLogScale)` - Get/Set if log scale must be used while mapping scalars to colors. The default is 0.
- `int = obj.GetUseLogScale ()` - Get/Set if log scale must be used while mapping scalars to colors. The default is 0.
- `obj.SetNumberOfValues (vtkIdType number)` - Set the number of values i.e. colors to be generated in the discrete lookup table. This has no effect if Discretize is off. The default is 256.
- `vtkIdType = obj.GetNumberOfValues ()` - Set the number of values i.e. colors to be generated in the discrete lookup table. This has no effect if Discretize is off. The default is 256.
- `obj.GetColor (double v, double rgb[3])` - Map one value through the lookup table and return the color as an RGB array of doubles between 0 and 1.
- `vtkUnsignedCharArray = obj.MapScalars (vtkDataArray scalars, int colorMode, int component)` - An internal method maps a data array into a 4-component, unsigned char RGBA array. The color mode determines the behavior of mapping. If VTK_COLOR_MODE_DEFAULT is set, then unsigned char data arrays are treated as colors (and converted to RGBA if necessary); otherwise, the data is mapped through this instance of ScalarsToColors. The offset is used for data arrays with more than one component; it indicates which component to use to do the blending. When the component argument is -1, then the this object uses its own selected technique to change a vector into a scalar to map.
- `obj.SetAlpha (double alpha)` - Specify an additional opacity (alpha) value to blend with. Values != 1 modify the resulting color consistent with the requested form of the output. This is typically used by an actor in order to blend its opacity. Overridden to pass the alpha to the internal vtkLookupTable.
- `int = obj.UsingLogScale ()`

31.64 vtkDistributedGraphHelper

31.64.1 Usage

A distributed graph helper can be attached to an empty vtkGraph object to turn the vtkGraph into a distributed graph, whose vertices and edges are distributed across several different processors. vtkDistributedGraphHelper is an abstract class. Use a subclass of vtkDistributedGraphHelper, such as vtkPBGLDistributedGraphHelper, to build distributed graphs.

The distributed graph helper provides facilities used by vtkGraph to communicate with other processors that store other parts of the same distributed graph. The only user-level functionality provided by vtkDistributedGraphHelper involves this communication among processors and the ability to map between "distributed" vertex and edge IDs and their component parts (processor and local index). For example, the Synchronize() method provides a barrier that allows all processors to catch up to the same point in the code before any processor can leave that Synchronize() call. For example, one would call Synchronize() after adding many edges to a distributed graph, so that all processors can handle the addition of inter-processor edges and continue, after the Synchronize() call, with a consistent view of the distributed graph. For more information about manipulating (distributed) graphs, see the vtkGraph documentation.

To create an instance of class vtkDistributedGraphHelper, simply invoke its constructor as follows

```
obj = vtkDistributedGraphHelper
```

31.64.2 Methods

The class vtkDistributedGraphHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDistributedGraphHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDistributedGraphHelper = obj.NewInstance ()`
- `vtkDistributedGraphHelper = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetVertexOwner (vtkIdType v) const` - Returns owner of vertex v, by extracting top ceil(log2 P) bits of v.
- `vtkIdType = obj.GetVertexIndex (vtkIdType v) const` - Returns local index of vertex v, by masking off top ceil(log2 P) bits of v.
- `vtkIdType = obj.GetEdgeOwner (vtkIdType e\_id) const` - Returns owner of edge with ID e_id, by extracting top ceil(log2 P) bits of e_id.
- `vtkIdType = obj.GetEdgeIndex (vtkIdType e\_id) const` - Returns local index of edge with ID e_id, by masking off top ceil(log2 P) bits of e_id.
- `vtkIdType = obj.MakeDistributedId (int owner, vtkIdType local)` - Builds a distributed ID consisting of the given owner and the local ID.
- `obj.Synchronize ()` - Synchronizes all of the processors involved in this distributed graph, so that all processors have a consistent view of the distributed graph for the computation that follows. This routine should be invoked after adding new edges into the distributed graph, so that other processors will see those edges (or their corresponding back-edges).
- `vtkDistributedGraphHelper = obj.Clone ()` - Clones the distributed graph helper, returning another distributed graph helper of the same kind that can be used in another vtkGraph.

31.65 vtkEdgeListIterator

31.65.1 Usage

vtkEdgeListIterator iterates through all the edges in a graph, by traversing the adjacency list for each vertex. You may instantiate this class directly and call SetGraph() to traverse a certain graph. You may also call the graph's GetEdges() method to set up the iterator for a certain graph.

Note that this class does NOT guarantee that the edges will be processed in order of their ids (i.e. it will not necessarily return edge 0, then edge 1, etc.).

To create an instance of class vtkEdgeListIterator, simply invoke its constructor as follows

```
obj = vtkEdgeListIterator
```

31.65.2 Methods

The class vtkEdgeListIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEdgeListIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEdgeListIterator = obj.NewInstance ()`
- `vtkEdgeListIterator = obj.SafeDownCast (vtkObject o)`
- `vtkGraph = obj.GetGraph ()`
- `obj.SetGraph (vtkGraph graph)`
- `vtkGraphEdge = obj.NextGraphEdge ()` - Just like Next(), but returns heavy-weight vtkGraphEdge object instead of the vtkEdgeType struct, for use with wrappers. The graph edge is owned by this iterator, and changes after each call to NextGraphEdge().
- `bool = obj.HasNext ()` - Whether this iterator has more edges.

31.66 vtkEmptyCell

31.66.1 Usage

vtkEmptyCell is a concrete implementation of vtkCell. It is used during processing to represented a deleted element.

To create an instance of class vtkEmptyCell, simply invoke its constructor as follows

```
obj = vtkEmptyCell
```

31.66.2 Methods

The class vtkEmptyCell has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEmptyCell class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEmptyCell = obj.NewInstance ()`
- `vtkEmptyCell = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.

- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.InterpolateFunctions (double pcoords[3], double weights)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)

31.67 vtkExecutive

31.67.1 Usage

vtkExecutive is the superclass for all pipeline executives in VTK. A VTK executive is responsible for controlling one instance of vtkAlgorithm. A pipeline consists of one or more executives that control data flow. Every reader, source, writer, or data processing algorithm in the pipeline is implemented in an instance of vtkAlgorithm.

To create an instance of class vtkExecutive, simply invoke its constructor as follows

```
obj = vtkExecutive
```

31.67.2 Methods

The class vtkExecutive has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExecutive class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExecutive = obj.NewInstance ()`
- `vtkExecutive = obj.SafeDownCast (vtkObject o)`
- `vtkAlgorithm = obj.GetAlgorithm ()` - Get the algorithm to which this executive has been assigned.
- `int = obj.Update ()` - Bring the algorithm's outputs up-to-date. Returns 1 for success and 0 for failure.
- `int = obj.Update (int port)` - Bring the algorithm's outputs up-to-date. Returns 1 for success and 0 for failure.
- `int = obj.GetNumberOfInputPorts ()` - Get the number of input/output ports for the algorithm associated with this executive. Returns 0 if no algorithm is set.
- `int = obj.GetNumberOfOutputPorts ()` - Get the number of input/output ports for the algorithm associated with this executive. Returns 0 if no algorithm is set.

- `int = obj.GetNumberOfInputConnections (int port)` - Get the number of input connections on the given port.
- `vtkInformation = obj.GetOutputInformation (int port)` - Get the pipeline information object for the given output port.
- `vtkInformationVector = obj.GetOutputInformation ()` - Get the pipeline information object for all output ports.
- `vtkInformation = obj.GetInputInformation (int port, int connection)` - Get the pipeline information for the given input connection.
- `vtkInformationVector = obj.GetInputInformation (int port)` - Get the pipeline information vectors for the given input port.
- `vtkExecutive = obj.GetInputExecutive (int port, int connection)` - Get the executive managing the given input connection.
- `vtkDataObject = obj.GetOutputData (int port)` - Get/Set the data object for an output port of the algorithm.
- `obj.SetOutputData (int port, vtkDataObject , vtkInformation info)` - Get/Set the data object for an output port of the algorithm.
- `obj.SetOutputData (int port, vtkDataObject )` - Get/Set the data object for an output port of the algorithm.
- `vtkDataObject = obj.GetInputData (int port, int connection)` - Get the data object for an input port of the algorithm.
- `vtkAlgorithmOutput = obj.GetProducerPort (vtkDataObject )` - Get the output port that produces the given data object.
- `obj.SetSharedOutputInformation (vtkInformationVector outInfoVec)` - Set a pointer to an outside instance of input or output information vectors. No references are held to the given vectors, and setting this does not change the executive object modification time. This is a preliminary interface to use in implementing filters with internal pipelines, and may change without notice when a future interface is created.
- `obj.Register (vtkObjectBase o)` - Participate in garbage collection.
- `obj.UnRegister (vtkObjectBase o)` - Participate in garbage collection.

31.68 vtkExecutiveCollection

31.68.1 Usage

vtkExecutiveCollection is an object that creates and manipulates lists of objects that are (inherited from) vtkExecutives.

To create an instance of class vtkExecutiveCollection, simply invoke its constructor as follows

```
obj = vtkExecutiveCollection
```

31.68.2 Methods

The class vtkExecutiveCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExecutiveCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExecutiveCollection = obj.NewInstance ()`
- `vtkExecutiveCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkExecutive exec)` - Get the next executive in the list.
- `vtkExecutive = obj.GetNextItem ()`

31.69 vtkExplicitCell

31.69.1 Usage

vtkExplicitCell is an abstract superclass for cells that cannot be represented implicitly. An implicit representation requires only a cell type and connectivity list (e.g., triangle). Explicit cells require information beyond this; e.g., a NURBS surface or cells that require explicit face/edge descriptions. Most cells in VTK are implicitly represented.

To create an instance of class vtkExplicitCell, simply invoke its constructor as follows

```
obj = vtkExplicitCell
```

31.69.2 Methods

The class vtkExplicitCell has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExplicitCell class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExplicitCell = obj.NewInstance ()`
- `vtkExplicitCell = obj.SafeDownCast (vtkObject o)`
- `int = obj.IsExplicitCell ()` - Set/Get the cell id. This is necessary for explicit cells because they often need to keep extra information (typically contained in the cell data of a point set). This information might be things like knot points/weights, boundaries, etc.
- `obj.SetCellId (vtkIdType )` - Set/Get the cell id. This is necessary for explicit cells because they often need to keep extra information (typically contained in the cell data of a point set). This information might be things like knot points/weights, boundaries, etc.
- `vtkIdType = obj.GetCellId ()` - Set/Get the cell id. This is necessary for explicit cells because they often need to keep extra information (typically contained in the cell data of a point set). This information might be things like knot points/weights, boundaries, etc.

- `obj.SetDataSet (vtkDataSet )` - Set/Get the mesh that owns this cell. This is necessary for explicit cells because they often need to keep extra information (typically contained in the cell data of a point set). This information might be things like knot points/weights, boundaries, etc.
- `vtkDataSet = obj.GetDataSet ()` - Set/Get the mesh that owns this cell. This is necessary for explicit cells because they often need to keep extra information (typically contained in the cell data of a point set). This information might be things like knot points/weights, boundaries, etc.

31.70 vtkFieldData

31.70.1 Usage

vtkFieldData represents and manipulates fields of data. The model of a field is a m x n matrix of data values, where m is the number of tuples, and n is the number of components. (A tuple is a row of n components in the matrix.) The field is assumed to be composed of a set of one or more data arrays, where the data in the arrays are of different types (e.g., int, double, char, etc.), and there may be variable numbers of components in each array. Note that each data array is assumed to be "m" in length (i.e., number of tuples), which typically corresponds to the number of points or cells in a dataset. Also, each data array must have a character-string name. (This is used to manipulate data.)

There are two ways of manipulating and interfacing to fields. You can do it generically by manipulating components/tuples via a double-type data exchange, or you can do it by grabbing the arrays and manipulating them directly. The former is simpler but performs type conversion, which is bad if your data has non-castable types like (void) pointers, or you lose information as a result of the cast. The, more efficient method means managing each array in the field. Using this method you can create faster, more efficient algorithms that do not lose information.

To create an instance of class vtkFieldData, simply invoke its constructor as follows

```
obj = vtkFieldData
```

31.70.2 Methods

The class vtkFieldData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFieldData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFieldData = obj.NewInstance ()`
- `vtkFieldData = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Release all data but do not delete object. Also, clear the copy flags.
- `int = obj.Allocate (vtkIdType sz, vtkIdType ext)` - Allocate data for each array. Note that ext is no longer used.
- `obj.CopyStructure (vtkFieldData )` - Copy data array structure from a given field. The same arrays will exist with the same types, but will contain nothing in the copy.
- `obj.AllocateArrays (int num)` - AllocateOfArrays actually sets the number of vtkAbstractArray pointers in the vtkFieldData object, not the number of used pointers (arrays). Adding more arrays will cause the object to dynamically adjust the number of pointers if it needs to extend. Although AllocateArrays can be used if the number of arrays which will be added is known, it can be omitted with a small computation cost.

- `int = obj.GetNumberOfArrays ()` - Add an array to the array list. If an array with the same name already exists - then the added array will replace it.
- `int = obj.AddArray (vtkAbstractArray array)` - Add an array to the array list. If an array with the same name already exists - then the added array will replace it.
- `obj.RemoveArray (string name)` - Return the ith array in the field. A NULL is returned if the index i is out of range. A NULL is returned if the array at the given index is not a vtkDataArray.
- `vtkDataArray = obj.GetArray (int i)` - Return the ith array in the field. A NULL is returned if the index i is out of range. A NULL is returned if the array at the given index is not a vtkDataArray.
- `vtkDataArray = obj.GetArray (string arrayName)` - Returns the ith array in the field. Unlike GetArray(), this method returns a vtkAbstractArray. A NULL is returned only if the index i is out of range.
- `vtkAbstractArray = obj.GetAbstractArray (int i)` - Returns the ith array in the field. Unlike GetArray(), this method returns a vtkAbstractArray. A NULL is returned only if the index i is out of range.
- `vtkAbstractArray = obj.GetAbstractArray (string arrayName)` - Return 1 if an array with the given name could be found. 0 otherwise.
- `int = obj.HasArray (string name)` - Get the name of ith array. Note that this is equivalent to: GetAbstractArray(i)-¿GetName() if ith array pointer is not NULL
- `string = obj.GetArrayName (int i)` - Pass entire arrays of input data through to output. Obey the "copy" flags.
- `obj.PassData (vtkFieldData fd)` - Pass entire arrays of input data through to output. Obey the "copy" flags.
- `obj.CopyFieldOn (string name)` - Turn on/off the copying of the field specified by name. During the copying/passing, the following rules are followed for each array: 1. If the copy flag for an array is set (on or off), it is applied This overrides rule 2. 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array
- `obj.CopyFieldOff (string name)` - Turn on copying of all data. During the copying/passing, the following rules are followed for each array: 1. If the copy flag for an array is set (on or off), it is applied This overrides rule 2. 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array
- `obj.CopyAllOn (int unused)` - Turn on copying of all data. During the copying/passing, the following rules are followed for each array: 1. If the copy flag for an array is set (on or off), it is applied This overrides rule 2. 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array
- `obj.CopyAllOff (int unused)` - Turn off copying of all data. During the copying/passing, the following rules are followed for each array: 1. If the copy flag for an array is set (on or off), it is applied This overrides rule 2. 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array
- `obj.DeepCopy (vtkFieldData da)` - Copy a field by creating new data arrays (i.e., duplicate storage).
- `obj.ShallowCopy (vtkFieldData da)` - Copy a field by reference counting the data arrays.
- `obj.Squeeze ()` - Squeezes each data array in the field (Squeeze() reclaims unused memory.)
- `obj.Reset ()` - Resets each data array in the field (Reset() does not release memory but it makes the arrays look like they are empty.)

- `long = obj.GetActualMemorySize ()` - Return the memory in kilobytes consumed by this field data. Used to support streaming and reading/writing data. The value returned is guaranteed to be greater than or equal to the memory required to actually represent the data represented by this object.
- `long = obj.GetMTime ()` - Check object's components for modified times.
- `obj.GetField (vtkIdList ptId, vtkFieldData f)` - Get a field from a list of ids. Supplied field f should have same types and number of data arrays as this one (i.e., like CopyStructure() creates). This method should not be used if the instance is from a subclass of vtkFieldData (vtkPointData or vtkCellData). This is because in those cases, the attribute data is stored with the other fields and will cause the method to behave in an unexpected way.
- `int = obj.GetNumberOfComponents ()` - Get the number of components in the field. This is determined by adding up the components in each non-NULL array. This method should not be used if the instance is from a subclass of vtkFieldData (vtkPointData or vtkCellData). This is because in those cases, the attribute data is stored with the other fields and will cause the method to behave in an unexpected way.
- `vtkIdType = obj.GetNumberOfTuples ()` - Get the number of tuples in the field. Note: some fields have arrays with different numbers of tuples; this method returns the number of tuples in the first array. Mixed-length arrays may have to be treated specially. This method should not be used if the instance is from a subclass of vtkFieldData (vtkPointData or vtkCellData). This is because in those cases, the attribute data is stored with the other fields and will cause the method to behave in an unexpected way.
- `obj.SetNumberOfTuples (vtkIdType number)` - Set the number of tuples for each data array in the field. This method should not be used if the instance is from a subclass of vtkFieldData (vtkPointData or vtkCellData). This is because in those cases, the attribute data is stored with the other fields and will cause the method to behave in an unexpected way.
- `obj.SetTuple (vtkIdType i, vtkIdType j, vtkFieldData source)` - Set the jth tuple in source field data at the ith location. Set operations mean that no range checking is performed, so they're faster.
- `obj.InsertTuple (vtkIdType i, vtkIdType j, vtkFieldData source)` - Insert the jth tuple in source field data at the ith location. Range checking is performed and memory allocates as necessary.
- `vtkIdType = obj.InsertNextTuple (vtkIdType j, vtkFieldData source)` - Insert the jth tuple in source field data at the end of the tuple matrix. Range checking is performed and memory is allocated as necessary.
- `obj.GetTuple (vtkIdType i, double tuple)` - Copy the ith tuple value into a user provided tuple array. Make sure that you've allocated enough space for the copy. @deprecated as of VTK 5.2. Using this method for FieldData having arrays that are not subclasses of vtkDataArray may yield unexpected results.
- `obj.SetTuple (vtkIdType i, double tuple)` - Set the tuple value at the ith location. Set operations mean that no range checking is performed, so they're faster. @deprecated as of VTK 5.2. Using this method for FieldData having arrays that are not subclasses of vtkDataArray may yield unexpected results.
- `obj.InsertTuple (vtkIdType i, double tuple)` - Insert the tuple value at the ith location. Range checking is performed and memory allocates as necessary. @deprecated as of VTK 5.2. Using this method for FieldData having arrays that are not subclasses of vtkDataArray may yield unexpected results.
- `vtkIdType = obj.InsertNextTuple (double tuple)` - Insert the tuple value at the end of the tuple matrix. Range checking is performed and memory is allocated as necessary. @deprecated as of VTK

5.2. Using this method for FieldData having arrays that are not subclasses of vtkDataArray may yield unexpected results.

- `double = obj.GetComponent (vtkIdType i, int j)` - Get the component value at the ith tuple (or row) and jth component (or column). @deprecated as of VTK 5.2. Using this method for FieldData having arrays that are not subclasses of vtkDataArray may yield unexpected results.
- `obj.SetComponent (vtkIdType i, int j, double c)` - Set the component value at the ith tuple (or row) and jth component (or column). Range checking is not performed, so set the object up properly before invoking. @deprecated as of VTK 5.2. Using this method for FieldData having arrays that are not subclasses of vtkDataArray may yield unexpected results.
- `obj.InsertComponent (vtkIdType i, int j, double c)` - Insert the component value at the ith tuple (or row) and jth component (or column). Range checking is performed and memory allocated as necessary o hold data. @deprecated as of VTK 5.2. Using this method for FieldData having arrays that are not subclasses of vtkDataArray may yield unexpected results.

31.71 vtkGenericAdaptorCell

31.71.1 Usage

In VTK, spatial-temporal data is defined in terms of a dataset which is composed of cells. The cells are topological entities over which an interpolation field is applied. Cells are defined in terms of a topology (e.g., vertices, lines, triangles, polygons, tetrahedra, etc.), points that instantiate the geometry of the cells, and interpolation fields (in the general case one interpolation field is for geometry, the other is for attribute data associated with the cell).

Currently most algorithms in VTK use vtkCell and vtkDataSet, which make assumptions about the nature of datasets, cells, and attributes. In particular, this abstraction assumes that cell interpolation functions are linear, or products of linear functions. Further, VTK implements most of the interpolation functions. This implementation starts breaking down as the complexity of the interpolation (or basis) functions increases.

vtkGenericAdaptorCell addresses these issues by providing more general abstraction for cells. It also adopts modern C++ practices including using iterators. The vtkGenericAdaptorCell is designed to fit within the adaptor framework; meaning that it is meant to adapt VTK to external simulation systems (see the GenericFiltering/README.html).

Please note that most cells are defined in terms of other cells (the boundary cells). They are also defined in terms of points, which are not the same as vertices (vertices are a 0-D cell; points represent a position in space).

Another important concept is the notion of DOFNodes. These concept supports cell types with complex interpolation functions. For example, higher-order p-method finite elements may have different functions on each of their topological features (edges, faces, region). The coefficients of these polynomial functions are associated with DOFNodes. (There is a single DOFNode for each topological feature.) Note that from this perspective, points are used to establish the topological form of the cell; mid-side nodes and such are considered DOFNodes.

To create an instance of class vtkGenericAdaptorCell, simply invoke its constructor as follows

```
obj = vtkGenericAdaptorCell
```

31.71.2 Methods

The class vtkGenericAdaptorCell has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericAdaptorCell class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkGenericAdaptorCell = obj.NewInstance ()`
- `vtkGenericAdaptorCell = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetId ()` - Unique identification number of the cell over the whole data set. This unique key may not be contiguous.
- `int = obj.IsInDataSet ()` - Does 'this' a cell of a dataset? (otherwise, it is a boundary cell)
- `int = obj.GetType ()` - Return the type of the current cell.
  (result==VTK_HIGHER_ORDER_TRIANGLE)—— (result==VTK_HIGHER_ORDER_TETRAHEDRON)
- `int = obj.GetDimension ()` - Return the topological dimension of the current cell.
- `int = obj.GetGeometryOrder ()` - Return the interpolation order of the geometry.
- `int = obj.IsGeometryLinear ()` - Does the cell have a non-linear interpolation for the geometry?
- `int = obj.GetAttributeOrder (vtkGenericAttribute a)` - Return the interpolation order of attribute 'a' on the cell (may differ by cell).
- `int = obj.GetHighestOrderAttribute (vtkGenericAttributeCollection ac)` - Return the index of the first point centered attribute with the highest order in 'ac'.
- `int = obj.IsAttributeLinear (vtkGenericAttribute a)` - Does the attribute 'a' have a non-linear interpolation?
- `int = obj.IsPrimary ()` - Is the cell primary (i.e. not composite) ?
- `int = obj.GetNumberOfPoints ()` - Return the number of corner points that compose the cell.
- `int = obj.GetNumberOfBoundaries (int dim)` - Return the number of boundaries of dimension 'dim' (or all dimensions greater than 0 and less than GetDimension() if -1) of the cell. When dim is -1, the number of vertices is not included in the count because vertices are a special case: a vertex will have at most a single field value associated with it; DOF nodes may have an arbitrary number of field values associated with them.
- `int = obj.GetNumberOfDOFNodes ()` - Accumulated number of DOF nodes of the current cell. A DOF node is a component of cell with a given topological dimension. e.g.: a triangle has 4 DOF: 1 face and 3 edges. An hexahedron has 19 DOF: 1 region, 6 faces, and 12 edges.

  The number of vertices is not included in the count because vertices are a special case: a vertex will have at most a single field value associated with it; DOF nodes may have an arbitrary number of field values associated with them.
- `obj.GetPointIterator (vtkGenericPointIterator it)` - Return the points of cell into 'it'.
- `vtkGenericCellIterator = obj.NewCellIterator ()` - Create an empty cell iterator. The user is responsible for deleting it.
- `obj.GetBoundaryIterator (vtkGenericCellIterator boundaries, int dim)` - Return the 'boundaries' cells of dimension 'dim' (or all dimensions less than GetDimension() if -1) that are part of the boundary of the cell.
- `int = obj.CountNeighbors (vtkGenericAdaptorCell boundary)` - Number of cells (dimension¿boundary-¿GetDimension()) of the dataset that share the boundary 'boundary' of 'this'. 'this' IS NOT INCLUDED.
- `obj.CountEdgeNeighbors (int sharing)` - Number of cells (dimension¿boundary-¿GetDimension()) of the dataset that share the boundary 'boundary' of 'this'. 'this' IS NOT INCLUDED.

- `obj.GetNeighbors (vtkGenericAdaptorCell boundary, vtkGenericCellIterator neighbors)` - Put into 'neighbors' the cells (dimension¿boundary-¿GetDimension()) of the dataset that share the boundary 'boundary' with this cell. 'this' IS NOT INCLUDED.

- `obj.EvaluateLocation (int subId, double pcoords[3], double x[3])` - Determine the global coordinates 'x' from sub-cell 'subId' and parametric coordinates 'pcoords' in the cell.

  &&(pcoords[1]¡=1)&&(0¡=pcoords[2])&&(pcoords[2]¡=1)

- `obj.InterpolateTuple (vtkGenericAttribute a, double pcoords[3], double val)` - Interpolate the attribute 'a' at local position 'pcoords' of the cell into 'val'.

  pcoords[1]¡=1 && pcoords[2]¿=0 && pcoords[2]¡=1

- `obj.InterpolateTuple (vtkGenericAttributeCollection c, double pcoords[3], double val)` - Interpolate the whole collection of attributes 'c' at local position 'pcoords' of the cell into 'val'. Only point centered attributes are taken into account.

  pcoords[1]¡=1 && pcoords[2]¿=0 && pcoords[2]¡=1

- `obj.Contour (vtkContourValues values, vtkImplicitFunction f, vtkGenericAttributeCollection attribut` - Generate a contour (contouring primitives) for each 'values' or with respect to an implicit function 'f'. Contouring is performed on the scalar attribute ('attributes-¿GetActiveAttribute()' 'attributes-¿GetActiveComponent()'). Contouring interpolates the 'attributes-¿GetNumberOfattributesToInterpolate()' attributes 'attributes-¿GetAttributesToInterpolate()'. The 'locator', 'verts', 'lines', 'polys', 'outPd' and 'outCd' are cumulative data arrays over cell iterations: they store the result of each call to Contour(): - 'locator' is a points list that merges points as they are inserted (i.e., prevents duplicates). - 'verts' is an array of generated vertices - 'lines' is an array of generated lines - 'polys' is an array of generated polygons - 'outPd' is an array of interpolated point data along the edge (if not-NULL) - 'outCd' is an array of copied cell data of the current cell (if not-NULL) 'internalPd', 'secondaryPd' and 'secondaryCd' are initialized by the filter that call it from 'attributes'. - 'internalPd' stores the result of the tessellation pass: the higher-order cell is tessellated into linear sub-cells. - 'secondaryPd' and 'secondaryCd' are used internally as inputs to the Contour() method on linear sub-cells. Note: the CopyAllocate() method must be invoked on both 'outPd' and 'outCd', from 'secondaryPd' and 'secondaryCd'.

  NOTE: 'vtkGenericAttributeCollection *attributes' will be replaced by a 'vtkInformation'.

- `obj.Clip (double value, vtkImplicitFunction f, vtkGenericAttributeCollection attributes, vtkGeneric` - Cut (or clip) the current cell with respect to the contour defined by the 'value' or the implicit function 'f' of the scalar attribute ('attributes-¿GetActiveAttribute()','attributes-¿GetActiveComponent()'). If 'f' exists, 'value' is not used. The output is the part of the current cell which is inside the contour. The output is a set of zero, one or more cells of the same topological dimension as the current cell. Normally, cell points whose scalar value is greater than "value" are considered inside. If 'insideOut' is on, this is reversed. Clipping interpolates the 'attributes-¿GetNumberOfattributesToInterpolate()' attributes 'attributes-¿GetAttributesToInterpolate()'. 'locator', 'connectivity', 'outPd' and 'outCd' are cumulative data arrays over cell iterations: they store the result of each call to Clip(): - 'locator' is a points list that merges points as they are inserted (i.e., prevents duplicates). - 'connectivity' is an array of generated cells - 'outPd' is an array of interpolated point data along the edge (if not-NULL) - 'outCd' is an array of copied cell data of the current cell (if not-NULL) 'internalPd', 'secondaryPd' and 'secondaryCd' are initialized by the filter that call it from 'attributes'. - 'internalPd' stores the result of the tessellation pass: the higher-order cell is tessellated into linear sub-cells. - 'secondaryPd' and 'secondaryCd' are used internally as inputs to the Clip() method on linear sub-cells. Note: the CopyAllocate() method must be invoked on both 'outPd' and 'outCd', from 'secondaryPd' and 'secondaryCd'.

  NOTE: 'vtkGenericAttributeCollection *attributes' will be replaced by a 'vtkInformation'.

- `obj.Derivatives (int subId, double pcoords[3], vtkGenericAttribute attribute, double derivs)` - Compute derivatives 'derivs' of the attribute 'attribute' (from its values at the corner points of the

cell) given sub-cell 'subId' (0 means primary cell) and parametric coordinates 'pcoords'. Derivatives are in the x-y-z coordinate directions for each data value.

&&(pcoords[1]¡=1)&&(0¡=pcoords[2])

- `obj.GetBounds (double bounds[6])` - Compute the bounding box of the current cell in 'bounds' in global coordinates. THREAD SAFE

- `double = obj.GetLength2 ()` - Return the bounding box diagonal squared of the current cell.

- `int = obj.GetParametricCenter (double pcoords[3])` - Get the center of the current cell (in parametric coordinates) and place it in 'pcoords'. If the current cell is a composite, the return value is the sub-cell id that the center is in. (result¿=0) && (IsPrimary() implies result==0)

- `double = obj.GetParametricDistance (double pcoords[3])` - Return the distance of the parametric coordinate 'pcoords' to the current cell. If inside the cell, a distance of zero is returned. This is used during picking to get the correct cell picked. (The tolerance will occasionally allow cells to be picked who are not really intersected "inside" the cell.)

- `obj.Tessellate (vtkGenericAttributeCollection attributes, vtkGenericCellTessellator tess, vtkPoints` - Tessellate the cell if it is not linear or if at least one attribute of 'attributes' is not linear. The output are linear cells of the same dimension than the cell. If the cell is linear and all attributes are linear, the output is just a copy of the current cell. 'points', 'cellArray', 'pd' and 'cd' are cumulative output data arrays over cell iterations: they store the result of each call to Tessellate(). 'internalPd' is initialized by the calling filter and stores the result of the tessellation. If it is not null, 'types' is filled with the types of the linear cells. 'types' is null when it is called from vtkGenericGeometryFilter and not null when it is called from vtkGenericDatasetTessellator.

- `int = obj.IsFaceOnBoundary (vtkIdType faceId)` - Is the face 'faceId' of the current cell on the exterior boundary of the dataset?

- `int = obj.IsOnBoundary ()` - Is the cell on the exterior boundary of the dataset?

- `obj.TriangulateFace (vtkGenericAttributeCollection attributes, vtkGenericCellTessellator tess, int` - Tessellate face 'index' of the cell. See Tessellate() for further explanations.

- `int = obj.GetNumberOfVerticesOnFace (int faceId)` - Return the number of vertices defining face 'faceId'.

31.72 vtkGenericAttribute

31.72.1 Usage

vtkGenericAttribute is an abstract class that defines an API for attribute data. Attribute data is data associated with the topology or geometry of a dataset (i.e., points, cells, etc.). vtkGenericAttribute is part of the adaptor framework (see GenericFiltering/README.html).

vtkGenericAttribute provides a more general interface to attribute data than its counterpart vtkDataArray (which assumes a linear, contiguous array). It adopts an iterator interface, and allows attributes to be associated with points, edges, faces, or edges.

To create an instance of class vtkGenericAttribute, simply invoke its constructor as follows

```
obj = vtkGenericAttribute
```

31.72.2 Methods

The class vtkGenericAttribute has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericAttribute class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenericAttribute = obj.NewInstance ()`
- `vtkGenericAttribute = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetName ()` - Name of the attribute. (e.g. "velocity")
- `int = obj.GetNumberOfComponents ()` - Dimension of the attribute. (1 for scalar, 3 for velocity)
- `int = obj.GetCentering ()` - Is the attribute centered either on points, cells or boundaries?
- `int = obj.GetType ()` - Type of the attribute: scalar, vector, normal, texture coordinate, tensor
  ——(result==vtkDataSetAttributes::VECTORS) ——(result==vtkDataSetAttributes::NORMALS) ——(result==vtkDataSetAttributes::TCOORDS) ——(result==vtkDataSetAttributes::TENSORS)
- `int = obj.GetComponentType ()` - Type of the components of the attribute: int, float, double
  ——(result==VTK_UNSIGNED_CHAR) ——(result==VTK_SHORT) ——(result==VTK_UNSIGNED_SHORT)——
  ——(result==VTK_UNSIGNED_INT) ——(result==VTK_LONG) ——(result==VTK_UNSIGNED_LONG)
  ——(result==VTK_FLOAT) ——(result==VTK_DOUBLE) ——(result==VTK_ID_TYPE)
- `vtkIdType = obj.GetSize ()` - Number of tuples.
- `long = obj.GetActualMemorySize ()` - Size in kilobytes taken by the attribute.
- `obj.GetRange (int component, double range[2])` - Range of the attribute component 'component'. If 'component'==-1, it returns the range of the magnitude (euclidean norm). THREAD SAFE
- `double = obj.GetMaxNorm ()` - Return the maximum euclidean norm for the tuples.
- `obj.GetTuple (vtkGenericAdaptorCell c, double tuple)` - Put attribute at all points of cell 'c' in 'tuple'.
- `obj.GetTuple (vtkGenericCellIterator c, double tuple)` - Put attribute at all points of cell 'c' in 'tuple'.
- `obj.GetTuple (vtkGenericPointIterator p, double tuple)` - Put the value of the attribute at position 'p' into 'tuple'.
- `obj.GetComponent (int i, vtkGenericCellIterator c, double values)` - Put component 'i' of the attribute at all points of cell 'c' in 'values'.
- `double = obj.GetComponent (int i, vtkGenericPointIterator p)` - Value of the component 'i' of the attribute at position 'p'.
- `obj.DeepCopy (vtkGenericAttribute other)` - Recursive duplication of 'other' in 'this'.
- `obj.ShallowCopy (vtkGenericAttribute other)` - Update 'this' using fields of 'other'.

31.73 vtkGenericAttributeCollection

31.73.1 Usage

vtkGenericAttributeCollection is a class that collects attributes (represented by vtkGenericAttribute).

To create an instance of class vtkGenericAttributeCollection, simply invoke its constructor as follows

```
obj = vtkGenericAttributeCollection
```

31.73.2 Methods

The class vtkGenericAttributeCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericAttributeCollection class.

- `string = obj.GetClassName ()` - Standard type definition and print methods for a VTK class.
- `int = obj.IsA (string name)` - Standard type definition and print methods for a VTK class.
- `vtkGenericAttributeCollection = obj.NewInstance ()` - Standard type definition and print methods for a VTK class.
- `vtkGenericAttributeCollection = obj.SafeDownCast (vtkObject o)` - Standard type definition and print methods for a VTK class.
- `int = obj.GetNumberOfAttributes ()` - Return the number of attributes (e.g., instances of vtkGenericAttribute) in the collection.
- `int = obj.GetNumberOfComponents ()` - Return the number of components. This is the sum of all components found in all attributes.
- `int = obj.GetNumberOfPointCenteredComponents ()` - Return the number of components. This is the sum of all components found in all point centered attributes.
- `int = obj.GetMaxNumberOfComponents ()` - Maximum number of components encountered among all attributes.
- `long = obj.GetActualMemorySize ()` - Actual size of the data in kilobytes; only valid after the pipeline has updated. It is guaranteed to be greater than or equal to the memory required to represent the data.
- `int = obj.IsEmpty ()` - Indicate whether the collection contains any attributes.
- `vtkGenericAttribute = obj.GetAttribute (int i)` - Return a pointer to the ith instance of vtkGenericAttribute.
- `int = obj.FindAttribute (string name)` - Return the index of the attribute named 'name'. Return the non-negative index if found. Return -1 otherwise.
- `int = obj.GetAttributeIndex (int i)` - Return the index of the first component of attribute 'i' in an array of format attrib0comp0 attrib0comp1 ... attrib4comp0 ...
- `obj.InsertNextAttribute (vtkGenericAttribute a)` - Add the attribute 'a' to the end of the collection.
- `obj.InsertAttribute (int i, vtkGenericAttribute a)` - Replace the attribute at index 'i' by 'a'.
- `obj.RemoveAttribute (int i)` - Remove the attribute at 'i'.
- `obj.Reset ()` - Remove all attributes.

- `obj.DeepCopy (vtkGenericAttributeCollection other)` - Copy, without reference counting, the other attribute array.
- `obj.ShallowCopy (vtkGenericAttributeCollection other)` - Copy, via reference counting, the other attribute array.
- `long = obj.GetMTime ()` - vtkAttributeCollection is a composite object and needs to check each member of its collection for modified time.
- `int = obj.GetActiveAttribute ()` - Index of the attribute to be processed (not necessarily scalar).
- `int = obj.GetActiveComponent ()` - Component of the active attribute to be processed. -1 means module.

  result¡GetAttribute(GetActiveAttribute())-¿GetNumberOfComponents()
- `obj.SetActiveAttribute (int attribute, int component)` - Set the scalar attribute to be processed. -1 means module.

  component¡GetAttribute(attribute)-¿GetNumberOfComponents()

  GetActiveComponent()==component
- `int = obj.GetNumberOfAttributesToInterpolate ()` - Number of attributes to interpolate.
- `int = obj.HasAttribute (int size, int attributes, int attribute)`
- `obj.SetAttributesToInterpolate (int size, int attributes)` - Set the attributes to interpolate.

  !HasAttributes(size,attributes,GetActiveAttribute())

  (GetAttributesToInterpolate()==attributes)
- `obj.SetAttributesToInterpolateToAll ()` - Set the attributes to interpolate.

  !HasAttributes(size,attributes,GetActiveAttribute())

  (GetAttributesToInterpolate()==attributes)

31.74 vtkGenericCell

31.74.1 Usage

vtkGenericCell is a class that provides access to concrete types of cells. It's main purpose is to allow thread-safe access to cells, supporting the vtkDataSet::GetCell(vtkGenericCell *) method. vtkGenericCell acts like any type of cell, it just dereferences an internal representation. The SetCellType() methods use #define constants; these are defined in the file vtkCellType.h.

To create an instance of class vtkGenericCell, simply invoke its constructor as follows

```
obj = vtkGenericCell
```

31.74.2 Methods

The class vtkGenericCell has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericCell class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkGenericCell = obj.NewInstance ()`
- `vtkGenericCell = obj.SafeDownCast (vtkObject o)`
- `obj.ShallowCopy (vtkCell c)` - See the vtkCell API for descriptions of these methods.
- `obj.DeepCopy (vtkCell c)` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.IsLinear ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.RequiresInitialization ()` - See the vtkCell API for descriptions of these methods.
- `obj.Initialize ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetParametricCenter (double pcoords[3])` - See the vtkCell API for descriptions of these methods.
- `int = obj.IsPrimaryCell ()` - See the vtkCell API for descriptions of these methods.
- `obj.InterpolateFunctions (double pcoords[3], double weights)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.SetCellType (int cellType)` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToEmptyCell ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.

- `obj.SetCellTypeToVertex ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToPolyVertex ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToLine ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToPolyLine ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToTriangle ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToTriangleStrip ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToPolygon ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToPixel ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToQuad ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToTetra ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToVoxel ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToHexahedron ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.

- `obj.SetCellTypeToWedge ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToPyramid ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToPentagonalPrism ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToHexagonalPrism ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToConvexPointSet ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToQuadraticEdge ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToCubicLine ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToQuadraticTriangle ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToBiQuadraticTriangle ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCel *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToQuadraticQuad ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToQuadraticTetra ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToQuadraticHexahedron ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCel *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.

- `obj.SetCellTypeToQuadraticWedge ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToQuadraticPyramid ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToQuadraticLinearQuad ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCel *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToBiQuadraticQuad ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericCell *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToQuadraticLinearWedge ()` - This method is used to support the vtkDataSet::GetCell(vtkGenericC *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToBiQuadraticQuadraticWedge ()` - This method is used to support the vtkDataSet::GetCell(vtkGen *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToTriQuadraticHexahedron ()` - This method is used to support the vtkDataSet::GetCell(vtkGeneric *) method. It allows vtkGenericCell to act like any cell type by dereferencing an internal instance of a concrete cell type. When you set the cell type, you are resetting a pointer to an internal cell which is then used for computation.
- `obj.SetCellTypeToBiQuadraticQuadraticHexahedron ()` - Instantiate a new vtkCell based on it's cell type value

31.75 vtkGenericCellIterator

31.75.1 Usage

This class (and subclasses) are used to iterate over cells. Use it only in conjunction with vtkGenericDataSet (i.e., the adaptor framework).

Typical use is:

```
vtkGenericDataSet *dataset;
vtkGenericCellIterator *it = dataset->NewCellIterator(2);
for (it->Begin(); !it->IsAtEnd(); it->Next());
  {
  spec=it->GetCell();
  }
```

To create an instance of class vtkGenericCellIterator, simply invoke its constructor as follows

```
obj = vtkGenericCellIterator
```

31.75.2 Methods

The class vtkGenericCellIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericCellIterator class.

- `string = obj.GetClassName ()` - Standard VTK construction and type macros.
- `int = obj.IsA (string name)` - Standard VTK construction and type macros.
- `vtkGenericCellIterator = obj.NewInstance ()` - Standard VTK construction and type macros.
- `vtkGenericCellIterator = obj.SafeDownCast (vtkObject o)` - Standard VTK construction and type macros.
- `obj.Begin ()` - Move iterator to first position if any (loop initialization).
- `int = obj.IsAtEnd ()` - Is the iterator at the end of traversal?
- `vtkGenericAdaptorCell = obj.NewCell ()` - Create an empty cell. The user is responsible for deleting it.
- `obj.GetCell (vtkGenericAdaptorCell c)` - Get the cell at current position. The cell should be instantiated with the NewCell() method.

  THREAD SAFE
- `vtkGenericAdaptorCell = obj.GetCell ()` - Get the cell at the current traversal position. NOT THREAD SAFE
- `obj.Next ()` - Move the iterator to the next position in the list.

31.76 vtkGenericCellTessellator

31.76.1 Usage

vtkGenericCellTessellator is a helper class to perform adaptive tessellation of particular cell topologies. The major purpose for this class is to transform higher-order cell types (e.g., higher-order finite elements) into linear cells that can then be easily visualized by VTK. This class works in conjunction with the vtkGenericDataSet and vtkGenericAdaptorCell classes.

This algorithm is based on edge subdivision. An error metric along each edge is evaluated, and if the error is greater than some tolerance, the edge is subdivided (as well as all connected 2D and 3D cells). The process repeats until the error metric is satisfied.

A significant issue addressed by this algorithm is to insure face compatibility across neigboring cells. That is, diagonals due to face triangulation must match to insure that the mesh is compatible. The algorithm employs a precomputed table to accelerate the tessellation process. The table was generated with the help of vtkOrderedTriangulator; the basic idea is that the choice of diagonal is made by considering the relative value of the point ids.

To create an instance of class vtkGenericCellTessellator, simply invoke its constructor as follows

```
obj = vtkGenericCellTessellator
```

31.76.2 Methods

The class vtkGenericCellTessellator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericCellTessellator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenericCellTessellator = obj.NewInstance ()`
- `vtkGenericCellTessellator = obj.SafeDownCast (vtkObject o)`
- `obj.TessellateFace (vtkGenericAdaptorCell cell, vtkGenericAttributeCollection att, vtkIdType index,` - Tessellate a face of a 3D 'cell'. The face is specified by the index value. The result is a set of smaller linear triangles in 'cellArray' with 'points' and point data 'internalPd'.
- `obj.Tessellate (vtkGenericAdaptorCell cell, vtkGenericAttributeCollection att, vtkDoubleArray point` - Tessellate a 3D 'cell'. The result is a set of smaller linear tetrahedra in 'cellArray' with 'points' and point data 'internalPd'.
- `obj.Triangulate (vtkGenericAdaptorCell cell, vtkGenericAttributeCollection att, vtkDoubleArray poir` - Triangulate a 2D 'cell'. The result is a set of smaller linear triangles in 'cellArray' with 'points' and point data 'internalPd'.
- `obj.SetErrorMetrics (vtkCollection someErrorMetrics)` - Specify the list of error metrics used to decide if an edge has to be splitted or not. It is a collection of vtkGenericSubdivisionErrorMetric-s.
- `vtkCollection = obj.GetErrorMetrics ()` - Specify the list of error metrics used to decide if an edge has to be splitted or not. It is a collection of vtkGenericSubdivisionErrorMetric-s.
- `obj.Initialize (vtkGenericDataSet ds)` - Initialize the tessellator with a data set 'ds'.
- `obj.InitErrorMetrics (vtkGenericDataSet ds)` - Init the error metric with the dataset. Should be called in each filter before any tessellation of any cell.
- `int = obj.GetMeasurement ()` - If true, measure the quality of the fixed subdivision.
- `obj.SetMeasurement (int )` - If true, measure the quality of the fixed subdivision.
- `obj.GetMaxErrors (double errors)` - Get the maximum error measured after the fixed subdivision.

31.77 vtkGenericDataSet

31.77.1 Usage

In VTK, spatial-temporal data is defined in terms of a dataset. The dataset consists of geometry (e.g., points), topology (e.g., cells), and attributes (e.g., scalars, vectors, etc.) vtkGenericDataSet is an abstract class defining this abstraction.

Since vtkGenericDataSet provides a general interface to manipulate data, algorithms that process it tend to be slower than those specialized for a particular data type. For this reason, there are concrete, non-abstract subclasses that represent and provide access to data more efficiently. Note that filters to process this dataset type are currently found in the VTK/GenericFiltering/ subdirectory.

Unlike the vtkDataSet class, vtkGenericDataSet provides a more flexible interface including support for iterators. vtkGenericDataSet is also designed to interface VTK to external simulation packages without the penalty of copying memory (see VTK/GenericFiltering/README.html) for more information. Thus vtkGenericDataSet plays a central role in the adaptor framework.

Please note that this class introduces the concepts of "boundary cells". This refers to the boundaries of a cell (e.g., face of a tetrahedron) which may in turn be represented as a cell. Boundary cells are derivative topological features of cells, and are therefore never explicitly represented in the dataset. Often in visualization algorithms, looping over boundaries (edges or faces) is employed, while the actual dataset cells may not traversed. Thus there are methods to loop over these boundary cells.

Finally, as a point of clarification, points are not the same as vertices. Vertices refer to points, and points specify a position is space. Vertices are a type of 0-D cell. Also, the concept of a DOFNode, which

is where coefficients for higher-order cells are kept, is a new concept introduced by the adaptor framework (see vtkGenericAdaptorCell for more information).

To create an instance of class vtkGenericDataSet, simply invoke its constructor as follows

```
obj = vtkGenericDataSet
```

31.77.2 Methods

The class vtkGenericDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericDataSet class.

- `string = obj.GetClassName ()` - Standard VTK type and print macros.
- `int = obj.IsA (string name)` - Standard VTK type and print macros.
- `vtkGenericDataSet = obj.NewInstance ()` - Standard VTK type and print macros.
- `vtkGenericDataSet = obj.SafeDownCast (vtkObject o)` - Standard VTK type and print macros.
- `vtkIdType = obj.GetNumberOfPoints ()` - Return the number of points composing the dataset. See NewPointIterator() for more details.
- `vtkIdType = obj.GetNumberOfCells (int dim)` - Return the number of cells that explicitly define the dataset. See NewCellIterator() for more details.
- `int = obj.GetCellDimension ()` - Return -1 if the dataset is explicitly defined by cells of varying dimensions or if there are no cells. If the dataset is explicitly defined by cells of a unique dimension, return this dimension.
- `obj.GetCellTypes (vtkCellTypes types)` - Get a list of types of cells in a dataset. The list consists of an array of types (not necessarily in any order), with a single entry per type. For example a dataset 5 triangles, 3 lines, and 100 hexahedra would result a list of three entries, corresponding to the types VTK_TRIANGLE, VTK_LINE, and VTK_HEXAHEDRON. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `vtkGenericCellIterator = obj.NewCellIterator (int dim)` - Return an iterator to traverse cells of dimension 'dim' (or all dimensions if -1) that explicitly define the dataset. For instance, it will return only tetrahedra if the mesh is defined by tetrahedra. If the mesh is composed of two parts, one with tetrahedra and another part with triangles, it will return both, but will not return the boundary edges and vertices of these cells. The user is responsible for deleting the iterator.
- `vtkGenericCellIterator = obj.NewBoundaryIterator (int dim, int exteriorOnly)` - Return an iterator to traverse cell boundaries of dimension 'dim' (or all dimensions if -1) of the dataset. If 'exteriorOnly' is true, only the exterior cell boundaries of the dataset will be returned, otherwise it will return exterior and interior cell boundaries. The user is responsible for deleting the iterator.
- `vtkGenericPointIterator = obj.NewPointIterator ()` - Return an iterator to traverse the points composing the dataset; they can be points that define a cell (corner points) or isolated points. The user is responsible for deleting the iterator.
- `obj.FindPoint (double x[3], vtkGenericPointIterator p)` - Locate the closest point 'p' to position 'x' (global coordinates).
- `long = obj.GetMTime ()` - Datasets are composite objects and need to check each part for their modified time.
- `obj.ComputeBounds ()` - Compute the geometry bounding box.

- `obj.GetBounds (double bounds[6])` - Return the geometry bounding box in global coordinates in the form (xmin,xmax, ymin,ymax, zmin,zmax) in the 'bounds' array.
- `obj.GetCenter (double center[3])` - Get the center of the bounding box in global coordinates.
- `double = obj.GetLength ()` - Return the length of the diagonal of the bounding box.
- `vtkGenericAttributeCollection = obj.GetAttributes ()` - Get the collection of attributes associated with this dataset.
- `vtkDataSetAttributes = obj.GetAttributes (int type)` - Set/Get a cell tessellator if cells must be tessellated during processing.
- `obj.SetTessellator (vtkGenericCellTessellator tessellator)` - Set/Get a cell tessellator if cells must be tessellated during processing.
- `vtkGenericCellTessellator = obj.GetTessellator ()` - Set/Get a cell tessellator if cells must be tessellated during processing.
- `long = obj.GetActualMemorySize ()` - Actual size of the data in kilobytes; only valid after the pipeline has updated. It is guaranteed to be greater than or equal to the memory required to represent the data.
- `int = obj.GetDataObjectType ()` - Return the type of data object.
- `vtkIdType = obj.GetEstimatedSize ()` - Estimated size needed after tessellation (or special operation)

31.78 vtkGenericDataSetAlgorithm

31.78.1 Usage

To create an instance of class vtkGenericDataSetAlgorithm, simply invoke its constructor as follows

```
obj = vtkGenericDataSetAlgorithm
```

31.78.2 Methods

The class vtkGenericDataSetAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericDataSetAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenericDataSetAlgorithm = obj.NewInstance ()`
- `vtkGenericDataSetAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkGenericDataSet = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkGenericDataSet = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput ()`
- `vtkDataObject = obj.GetInput (int port)`

- `vtkGenericDataSet = obj.GetGenericDataSetInput (int port)`
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.79 vtkGenericEdgeTable

31.79.1 Usage

vtkGenericEdgeTable is used to indicate the existance of and hold information about edges. Similar to vtkEdgeTable, this class is more sophisticated in that it uses reference counting to keep track of when information about an edge should be deleted.

vtkGenericEdgeTable is a helper class used in the adaptor framework. It is used during the tessellation process to hold information about the error metric on each edge. This avoids recomputing the error metric each time the same edge is visited.

To create an instance of class vtkGenericEdgeTable, simply invoke its constructor as follows

```
obj = vtkGenericEdgeTable
```

31.79.2 Methods

The class vtkGenericEdgeTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericEdgeTable class.

- `string = obj.GetClassName ()` - Standard VTK type and print macros.
- `int = obj.IsA (string name)` - Standard VTK type and print macros.
- `vtkGenericEdgeTable = obj.NewInstance ()` - Standard VTK type and print macros.
- `vtkGenericEdgeTable = obj.SafeDownCast (vtkObject o)` - Standard VTK type and print macros.
- `obj.InsertEdge (vtkIdType e1, vtkIdType e2, vtkIdType cellId, int ref)` - Insert an edge but do not split it.
- `int = obj.RemoveEdge (vtkIdType e1, vtkIdType e2)` - Method to remove an edge from the table. The method returns the current reference count.
- `int = obj.IncrementEdgeReferenceCount (vtkIdType e1, vtkIdType e2, vtkIdType cellId)` - Method that increments the referencecount and returns it.

- `int = obj.CheckEdgeReferenceCount (vtkIdType e1, vtkIdType e2)` - Return the edge reference count.
- `obj.Initialize (vtkIdType start)` - To specify the starting point id. It will initialize LastPointId This is very sensitive the start point should be cautiously chosen
- `int = obj.GetNumberOfComponents ()` - Return the total number of components for the point-centered attributes.
- `obj.SetNumberOfComponents (int count)` - Set the total number of components for the point-centered attributes.
- `int = obj.CheckPoint (vtkIdType ptId)` - Check if a point is already in the point table.
- `int = obj.CheckPoint (vtkIdType ptId, double point[3], double scalar)` - Check for the existence of a point and return its coordinate value.
- `obj.InsertPoint (vtkIdType ptId, double point[3])` - Insert point associated with an edge.
- `obj.InsertPointAndScalar (vtkIdType ptId, double pt[3], double s)` - Insert point associated with an edge. re: sizeof(s)==GetNumberOfComponents()
- `obj.RemovePoint (vtkIdType ptId)` - Remove a point from the point table.
- `obj.IncrementPointReferenceCount (vtkIdType ptId)` - Increment the reference count for the indicated point.
- `obj.DumpTable ()` - For debugging purposes. It is particularly useful to dump the table and check that nothing is left after a complete iteration. LoadFactor should ideally be very low to be able to have a constant time access
- `obj.LoadFactor ()` - For debugging purposes. It is particularly useful to dump the table and check that nothing is left after a complete iteration. LoadFactor should ideally be very low to be able to have a constant time access

31.80 vtkGenericInterpolatedVelocityField

31.80.1 Usage

vtkGenericInterpolatedVelocityField acts as a continuous velocity field by performing cell interpolation on the underlying vtkDataSet. This is a concrete sub-class of vtkFunctionSet with NumberOfIndependentVariables = 4 (x,y,z,t) and NumberOfFunctions = 3 (u,v,w). Normally, every time an evaluation is performed, the cell which contains the point (x,y,z) has to be found by calling FindCell. This is a computationally expansive operation. In certain cases, the cell search can be avoided or shortened by providing a guess for the cell iterator. For example, in streamline integration, the next evaluation is usually in the same or a neighbour cell. For this reason, vtkGenericInterpolatedVelocityField stores the last cell iterator. If caching is turned on, it uses this iterator as the starting point.

To create an instance of class vtkGenericInterpolatedVelocityField, simply invoke its constructor as follows

```
obj = vtkGenericInterpolatedVelocityField
```

31.80.2 Methods

The class vtkGenericInterpolatedVelocityField has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericInterpolatedVelocityField class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenericInterpolatedVelocityField = obj.NewInstance ()`
- `vtkGenericInterpolatedVelocityField = obj.SafeDownCast (vtkObject o)`
- `int = obj.FunctionValues (double x, double f)` - Evaluate the velocity field, f, at (x, y, z, t). For now, t is ignored.
- `obj.AddDataSet (vtkGenericDataSet dataset)` - Add a dataset used for the implicit function evaluation. If more than one dataset is added, the evaluation point is searched in all until a match is found. THIS FUNCTION DOES NOT CHANGE THE REFERENCE COUNT OF dataset FOR THREAD SAFETY REASONS.
- `obj.ClearLastCell ()` - Set the last cell id to -1 so that the next search does not start from the previous cell
- `vtkGenericAdaptorCell = obj.GetLastCell ()` - Return the cell cached from last evaluation.
- `int = obj.GetLastLocalCoordinates (double pcoords[3])` - Returns the interpolation weights cached from last evaluation if the cached cell is valid (returns 1). Otherwise, it does not change w and returns 0.
- `int = obj.GetCaching ()` - Turn caching on/off.
- `obj.SetCaching (int )` - Turn caching on/off.
- `obj.CachingOn ()` - Turn caching on/off.
- `obj.CachingOff ()` - Turn caching on/off.
- `int = obj.GetCacheHit ()` - Caching statistics.
- `int = obj.GetCacheMiss ()` - Caching statistics.
- `string = obj.GetVectorsSelection ()` - If you want to work with an arbitrary vector array, then set its name here. By default this in NULL and the filter will use the active vector array.
- `obj.SelectVectors (string fieldName)` - Returns the last dataset that was visited. Can be used as a first guess as to where the next point will be as well as to avoid searching through all datasets to get more information about the point.
- `vtkGenericDataSet = obj.GetLastDataSet ()` - Returns the last dataset that was visited. Can be used as a first guess as to where the next point will be as well as to avoid searching through all datasets to get more information about the point.
- `obj.CopyParameters (vtkGenericInterpolatedVelocityField from)` - Copy the user set parameters from source. This copies the Caching parameters. Sub-classes can add more after chaining.

31.81 vtkGenericPointIterator

31.81.1 Usage

This class (and subclasses) are used to iterate over points. Use it only in conjunction with vtkGenericDataSet (i.e., the adaptor framework).

Typical use is:

```
vtkGenericDataSet *dataset;
vtkGenericPointIterator *it = dataset->NewPointIterator();
for (it->Begin(); !it->IsAtEnd(); it->Next());
  {
  x=it->GetPosition();
  }
```

To create an instance of class vtkGenericPointIterator, simply invoke its constructor as follows

```
obj = vtkGenericPointIterator
```

31.81.2 Methods

The class vtkGenericPointIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericPointIterator class.

- `string = obj.GetClassName ()` - Standard VTK construction and type macros.
- `int = obj.IsA (string name)` - Standard VTK construction and type macros.
- `vtkGenericPointIterator = obj.NewInstance ()` - Standard VTK construction and type macros.
- `vtkGenericPointIterator = obj.SafeDownCast (vtkObject o)` - Standard VTK construction and type macros.
- `obj.Begin ()` - Move iterator to first position if any (loop initialization).
- `int = obj.IsAtEnd ()` - Is the iterator at the end of traversal?
- `obj.Next ()` - Move the iterator to the next position in the list.
- `obj.GetPosition (double x[3])` - Get the coordinates of the point at the current iterator position.
- `vtkIdType = obj.GetId ()` - Return the unique identifier for the point, could be non-contiguous.

31.82 vtkGenericSubdivisionErrorMetric

31.82.1 Usage

Objects of that class answer the following question during the cell subdivision: "does the edge need to be subdivided?" through RequiresEdgeSubdivision(). The answer depends on the criterium actually used in the subclass of this abstract class: a geometric-based error metric (variation of edge from a straight line), an attribute-based error metric (variation of the active attribute/component value from a linear ramp) , a view-depend error metric, ... Cell subdivision is performed in the context of the adaptor framework: higher-order, or complex cells, are automatically tessellated into simplices so that they can be processed with conventional visualization algorithms.

To create an instance of class vtkGenericSubdivisionErrorMetric, simply invoke its constructor as follows

```
obj = vtkGenericSubdivisionErrorMetric
```

31.82.2 Methods

The class vtkGenericSubdivisionErrorMetric has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericSubdivisionErrorMetric class.

- `string = obj.GetClassName ()` - Standard VTK type and error macros.
- `int = obj.IsA (string name)` - Standard VTK type and error macros.
- `vtkGenericSubdivisionErrorMetric = obj.NewInstance ()` - Standard VTK type and error macros.
- `vtkGenericSubdivisionErrorMetric = obj.SafeDownCast (vtkObject o)` - Standard VTK type and error macros.
- `int = obj.RequiresEdgeSubdivision (double leftPoint, double midPoint, double rightPoint, double alp` - Does the edge need to be subdivided according to the implemented computation? The edge is defined by its 'leftPoint' and its 'rightPoint'. 'leftPoint', 'midPoint' and 'rightPoint' have to be initialized before calling RequiresEdgeSubdivision(). Their format is global coordinates, parametric coordinates and point centered attributes: xyx rst abc de... 'alpha' is the normalized abscissa of the midpoint along the edge. (close to 0 means close to the left point, close to 1 means close to the right point)

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6
- `double = obj.GetError (double leftPoint, double midPoint, double rightPoint, double alpha)` - Return the error at the mid-point. The type of error depends on the state of the concrete error metric. For instance, it can return an absolute or relative error metric. See RequiresEdgeSubdivision() for a description of the arguments.

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6
- `obj.SetGenericCell (vtkGenericAdaptorCell cell)` - The cell that the edge belongs to.
- `vtkGenericAdaptorCell = obj.GetGenericCell ()` - The cell that the edge belongs to.
- `obj.SetDataSet (vtkGenericDataSet ds)` - Set/Get the dataset to be tessellated.
- `vtkGenericDataSet = obj.GetDataSet ()` - Set/Get the dataset to be tessellated.

31.83 vtkGeometricErrorMetric

31.83.1 Usage

It is a concrete error metric, based on a geometric criterium: the variation of the edge from a straight line.
To create an instance of class vtkGeometricErrorMetric, simply invoke its constructor as follows

```
obj = vtkGeometricErrorMetric
```

31.83.2 Methods

The class vtkGeometricErrorMetric has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeometricErrorMetric class.

- `string = obj.GetClassName ()` - Standard VTK type and error macros.
- `int = obj.IsA (string name)` - Standard VTK type and error macros.
- `vtkGeometricErrorMetric = obj.NewInstance ()` - Standard VTK type and error macros.

- `vtkGeometricErrorMetric = obj.SafeDownCast (vtkObject o)` - Standard VTK type and error macros.
- `double = obj.GetAbsoluteGeometricTolerance ()` - Return the squared absolute geometric accuracy. See SetAbsoluteGeometricTolerance() for details.
- `obj.SetAbsoluteGeometricTolerance (double value)` - Set the geometric accuracy with a squared absolute value. This is the geometric object-based accuracy. Subdivision will be required if the square distance between the real point and the straight line passing through the vertices of the edge is greater than 'value'. For instance 0.01 will give better result than 0.1.
- `obj.SetRelativeGeometricTolerance (double value, vtkGenericDataSet ds)` - Set the geometric accuracy with a value relative to the length of the bounding box of the dataset. Internally compute the absolute tolerance. For instance 0.01 will give better result than 0.1.
- `int = obj.RequiresEdgeSubdivision (double leftPoint, double midPoint, double rightPoint, double alp` - Does the edge need to be subdivided according to the distance between the line passing through its endpoints and the mid point? The edge is defined by its 'leftPoint' and its 'rightPoint'. 'leftPoint', 'midPoint' and 'rightPoint' have to be initialized before calling RequiresEdgeSubdivision(). Their format is global coordinates, parametric coordinates and point centered attributes: xyx rst abc de... 'alpha' is the normalized abscissa of the midpoint along the edge. (close to 0 means close to the left point, close to 1 means close to the right point)

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6
- `double = obj.GetError (double leftPoint, double midPoint, double rightPoint, double alpha)` - Return the error at the mid-point. It will return an error relative to the bounding box size if GetRelative() is true, a square absolute error otherwise. See RequiresEdgeSubdivision() for a description of the arguments.

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6
- `int = obj.GetRelative ()` - Return the type of output of GetError()

31.84 vtkGraph

31.84.1 Usage

vtkGraph is the abstract base class that provides all read-only API for graph data types. A graph consists of a collection of vertices and a collection of edges connecting pairs of vertices. The vtkDirectedGraph subclass represents a graph whose edges have inherent order from source vertex to target vertex, while vtkUndirectedGraph is a graph whose edges have no inherent ordering.

Graph vertices may be traversed in two ways. In the current implementation, all vertices are assigned consecutive ids starting at zero, so they may be traversed in a simple for loop from 0 to graph-¿GetNumberOfVertices() - 1. You may alternately create a vtkVertexListIterator and call graph-¿GetVertices(it). it-¿Next() will return the id of the next vertex, while it-¿HasNext() indicates whether there are more vertices in the graph. This is the preferred method, since in the future graphs may support filtering or subsetting where the vertex ids may not be contiguous.

Graph edges must be traversed through iterators. To traverse all edges in a graph, create an instance of vtkEdgeListIterator and call graph-¿GetEdges(it). it-¿Next() returns lightweight vtkEdgeType structures, which contain the public fields Id, Source and Target. Id is the identifier for the edge, which may be used to look up values in assiciated edge data arrays. Source and Target store the ids of the source and target vertices of the edge. Note that the edge list iterator DOES NOT necessarily iterate over edges in order of ascending id. To traverse edges from wrapper code (Python, Tcl, Java), use it-¿NextGraphEdge() instead of it-¿Next(). This will return a heavyweight, wrappable vtkGraphEdge object, which has the same fields as vtkEdgeType accessible through getter methods.

To traverse all edges outgoing from a vertex, create a vtkOutEdgeIterator and call graph-¿GetOutEdges(v, it). it-¿Next() returns a lightweight vtkOutEdgeType containing the fields Id and Target. The source of the edge is always the vertex that was passed as an argument to GetOutEdges(). Incoming edges may be similarly traversed with vtkInEdgeIterator, which returns vtkInEdgeType structures with Id and Source fields. Both vtkOutEdgeIterator and vtkInEdgeIterator also provide the wrapper functions NextGraphEdge() which return vtkGraphEdge objects.

An additional iterator, vtkAdjacentVertexIterator can traverse outgoing vertices directly, instead needing to parse through edges. Initialize the iterator by calling graph-¿GetAdjacentVertices(v, it).

vtkGraph has two instances of vtkDataSetAttributes for associated vertex and edge data. It also has a vtkPoints instance which may store x,y,z locations for each vertex. This is populated by filters such as vtkGraphLayout and vtkAssignCoordinates.

All graph types share the same implementation, so the structure of one may be shared among multiple graphs, even graphs of different types. Structures from vtkUndirectedGraph and vtkMutableUndirectedGraph may be shared directly. Structures from vtkDirectedGraph, vtkMutableDirectedGraph, and vtkTree may be shared directly with the exception that setting a structure to a tree requires that a "is a tree" test passes.

For graph types that are known to be compatible, calling ShallowCopy() or DeepCopy() will work as expected. When the outcome of a conversion is unknown (i.e. setting a graph to a tree), CheckedShallowCopy() and CheckedDeepCopy() exist which are identical to ShallowCopy() and DeepCopy(), except that instead of emitting an error for an incompatible structure, the function returns false. This allows you to programmatically check structure compatibility without causing error messages.

To construct a graph, use vtkMutableDirectedGraph or vtkMutableUndirectedGraph. You may then use CheckedShallowCopy to set the contents of a mutable graph type into one of the non-mutable types vtkDirectedGraph, vtkUndirectedGraph. To construct a tree, use vtkMutableDirectedGraph, with directed edges which point from the parent to the child, then use CheckedShallowCopy to set the structure to a vtkTree.

To create an instance of class vtkGraph, simply invoke its constructor as follows

```
obj = vtkGraph
```

31.84.2 Methods

The class vtkGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraph = obj.NewInstance ()`
- `vtkGraph = obj.SafeDownCast (vtkObject o)`
- `vtkDataSetAttributes = obj.GetVertexData ()` - Get the vertex or edge data.
- `vtkDataSetAttributes = obj.GetEdgeData ()` - Get the vertex or edge data.
- `int = obj.GetDataObjectType ()` - Initialize to an empty graph.
- `obj.Initialize ()` - Initialize to an empty graph.
- `obj.GetPoint (vtkIdType ptId, double x[3])` - These methods return the point (0,0,0) until the points structure is created, when it returns the actual point position. In a distributed graph, only the points for local vertices can be retrieved.
- `vtkPoints = obj.GetPoints ()` - Returns the points array for this graph. If points is not yet constructed, generates and returns a new points array filled with (0,0,0) coordinates. In a distributed graph, only the points for local vertices can be retrieved or modified.

- `obj.SetPoints (vtkPoints points)` - Returns the points array for this graph. If points is not yet constructed, generates and returns a new points array filled with (0,0,0) coordinates. In a distributed graph, only the points for local vertices can be retrieved or modified.
- `obj.ComputeBounds ()` - Compute the bounds of the graph. In a distributed graph, this computes the bounds around the local part of the graph.
- `obj.GetBounds (double bounds[6])` - Return a pointer to the geometry bounding box in the form (xmin,xmax, ymin,ymax, zmin,zmax). In a distributed graph, this computes the bounds around the local part of the graph.
- `long = obj.GetMTime ()` - The modified time of the graph.
- `obj.GetOutEdges (vtkIdType v, vtkOutEdgeIterator it)` - Initializes the out edge iterator to iterate over all outgoing edges of vertex v. For an undirected graph, returns all incident edges. In a distributed graph, the vertex v must be local to this processor.
- `vtkIdType = obj.GetDegree (vtkIdType v)` - The total of all incoming and outgoing vertices for vertex v. For undirected graphs, this is simply the number of edges incident to v. In a distributed graph, the vertex v must be local to this processor.
- `vtkIdType = obj.GetOutDegree (vtkIdType v)` - The number of outgoing edges from vertex v. For undirected graphs, returns the same as GetDegree(). In a distributed graph, the vertex v must be local to this processor.
- `obj.GetOutEdge (vtkIdType v, vtkIdType index, vtkGraphEdge e)` - Random-access method for retrieving outgoing edges from vertex v. The method fills the vtkGraphEdge instance with the id, source, and target of the edge. This method is provided for wrappers, GetOutEdge(vtkIdType, vtkIdType) is preferred.
- `obj.GetInEdges (vtkIdType v, vtkInEdgeIterator it)` - Initializes the in edge iterator to iterate over all incoming edges to vertex v. For an undirected graph, returns all incident edges. In a distributed graph, the vertex v must be local to this processor.
- `vtkIdType = obj.GetInDegree (vtkIdType v)` - The number of incoming edges to vertex v. For undirected graphs, returns the same as GetDegree(). In a distributed graph, the vertex v must be local to this processor.
- `obj.GetInEdge (vtkIdType v, vtkIdType index, vtkGraphEdge e)` - Random-access method for retrieving incoming edges to vertex v. The method fills the vtkGraphEdge instance with the id, source, and target of the edge. This method is provided for wrappers, GetInEdge(vtkIdType, vtkIdType) is preferred.
- `obj.GetAdjacentVertices (vtkIdType v, vtkAdjacentVertexIterator it)` - Initializes the adjacent vertex iterator to iterate over all outgoing vertices from vertex v. For an undirected graph, returns all adjacent vertices. In a distributed graph, the vertex v must be local to this processor.
- `obj.GetEdges (vtkEdgeListIterator it)` - Initializes the edge list iterator to iterate over all edges in the graph. Edges may not be traversed in order of increasing edge id. In a distributed graph, this returns edges that are stored locally.
- `vtkIdType = obj.GetNumberOfEdges ()` - The number of edges in the graph. In a distributed graph, this returns the number of edges stored locally.
- `obj.GetVertices (vtkVertexListIterator it)` - Initializes the vertex list iterator to iterate over all vertices in the graph. In a distributed graph, the iterator traverses all local vertices.
- `vtkIdType = obj.GetNumberOfVertices ()` - The number of vertices in the graph. In a distributed graph, returns the number of local vertices in the graph.

- `obj.SetDistributedGraphHelper (vtkDistributedGraphHelper helper)` - Sets the distributed graph helper of this graph, turning it into a distributed graph. This operation can only be executed on an empty graph.
- `vtkDistributedGraphHelper = obj.GetDistributedGraphHelper ()` - Retrieves the distributed graph helper for this graph
- `obj.ShallowCopy (vtkDataObject obj)` - Shallow copies the data object into this graph. If it is an incompatible graph, reports an error.
- `obj.DeepCopy (vtkDataObject obj)` - Deep copies the data object into this graph. If it is an incompatible graph, reports an error.
- `obj.CopyStructure (vtkGraph g)` - Does a shallow copy of the topological information, but not the associated attributes.
- `bool = obj.CheckedShallowCopy (vtkGraph g)` - Performs the same operation as ShallowCopy(), but instead of reporting an error for an incompatible graph, returns false.
- `bool = obj.CheckedDeepCopy (vtkGraph g)` - Performs the same operation as DeepCopy(), but instead of reporting an error for an incompatible graph, returns false.
- `obj.Squeeze ()`
- `obj.ReorderOutVertices (vtkIdType v, vtkIdTypeArray vertices)` - Reorder the outgoing vertices of a vertex. The vertex list must have the same elements as the current out edge list, just in a different order. This method does not change the topology of the graph. In a distributed graph, the vertex v must be local.
- `bool = obj.IsSameStructure (vtkGraph other)` - Returns true if both graphs point to the same adjacency structure. Can be used to test the copy-on-write feature of the graph.
- `vtkIdType = obj.GetSourceVertex (vtkIdType e)` - Retrieve the source and target vertices for an edge id. NOTE: The first time this is called, the graph will build a mapping array from edge id to source/target that is the same size as the number of edges in the graph. If you have access to a vtkOutEdgeType, vtkInEdgeType, vtkEdgeType, or vtkGraphEdge, you should directly use these structures to look up the source or target instead of this method.
- `vtkIdType = obj.GetTargetVertex (vtkIdType e)` - Retrieve the source and target vertices for an edge id. NOTE: The first time this is called, the graph will build a mapping array from edge id to source/target that is the same size as the number of edges in the graph. If you have access to a vtkOutEdgeType, vtkInEdgeType, vtkEdgeType, or vtkGraphEdge, you should directly use these structures to look up the source or target instead of this method.
- `vtkIdType = obj.GetNumberOfEdgePoints (vtkIdType e)` - Get the number of edge points associated with an edge.
- `double = obj.GetEdgePoint (vtkIdType e, vtkIdType i)` - Get the x,y,z location of a point along edge e.
- `obj.ClearEdgePoints (vtkIdType e)` - Clear all points associated with an edge.
- `obj.SetEdgePoint (vtkIdType e, vtkIdType i, double x[3])` - Set an x,y,z location of a point along an edge. This assumes there is already a point at location i, and simply overwrites it.
- `obj.SetEdgePoint (vtkIdType e, vtkIdType i, double x, double y, double z)` - Adds a point to the end of the list of edge points for a certain edge.
- `obj.AddEdgePoint (vtkIdType e, double x[3])` - Adds a point to the end of the list of edge points for a certain edge.

- `obj.AddEdgePoint (vtkIdType e, double x, double y, double z)` - Copy the internal edge point data from another graph into this graph. Both graphs must have the same number of edges.
- `obj.ShallowCopyEdgePoints (vtkGraph g)` - Copy the internal edge point data from another graph into this graph. Both graphs must have the same number of edges.
- `obj.DeepCopyEdgePoints (vtkGraph g)` - Copy the internal edge point data from another graph into this graph. Both graphs must have the same number of edges.
- `vtkGraphInternals = obj.GetGraphInternals (bool modifying)` - Returns the internal representation of the graph. If modifying is true, then the returned vtkGraphInternals object will be unique to this vtkGraph object.
- `obj.GetInducedEdges (vtkIdTypeArray verts, vtkIdTypeArray edges)` - Fills a list of edge indices with the edges contained in the induced subgraph formed by the vertices in the vertex list.
- `vtkFieldData = obj.GetAttributesAsFieldData (int type)` - Returns the attributes of the data object as a vtkFieldData. This returns non-null values in all the same cases as GetAttributes, in addition to the case of FIELD, which will return the field data for any vtkDataObject subclass.
- `vtkIdType = obj.GetNumberOfElements (int type)` - Get the number of elements for a specific attribute type (VERTEX, EDGE, etc.).
- `obj.Dump ()` - Dump the contents of the graph to standard output.

31.85 vtkGraphAlgorithm

31.85.1 Usage

vtkGraphAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline architecture. There are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this class constructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be Graph. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into separate functions such as ExecuteData and ExecuteInformation. For new algorithms you should implement RequestData(request, inputVec, outputVec) but for older filters there is a default implementation that calls the old ExecuteData(output) signature. For even older filters that don't implement ExecuteData the default implementation calls the even older Execute() signature.

.SECTION Thanks Thanks to Patricia Crossno, Ken Moreland, Andrew Wilson and Brian Wylie from Sandia National Laboratories for their help in developing this class.

To create an instance of class vtkGraphAlgorithm, simply invoke its constructor as follows

```
obj = vtkGraphAlgorithm
```

31.85.2 Methods

The class vtkGraphAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphAlgorithm = obj.NewInstance ()`

- `vtkGraphAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkGraph = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkGraph = obj.GetOutput (int index)` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int index, vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.86 vtkGraphEdge

31.86.1 Usage

A heavy-weight (vtkObject subclass) graph edge object that may be used instead of the vtkEdgeType struct, for use with wrappers. The edge contains the source and target vertex ids, and the edge id.

To create an instance of class vtkGraphEdge, simply invoke its constructor as follows

```
obj = vtkGraphEdge
```

31.86.2 Methods

The class vtkGraphEdge has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphEdge class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphEdge = obj.NewInstance ()`
- `vtkGraphEdge = obj.SafeDownCast (vtkObject o)`
- `obj.SetSource (vtkIdType )` - The source of the edge.
- `vtkIdType = obj.GetSource ()` - The source of the edge.
- `obj.SetTarget (vtkIdType )` - The target of the edge.
- `vtkIdType = obj.GetTarget ()` - The target of the edge.
- `obj.SetId (vtkIdType )` - The id of the edge.
- `vtkIdType = obj.GetId ()` - The id of the edge.

31.87 vtkGraphInternals

31.87.1 Usage

This is the internal representation of vtkGraph, used only in rare cases where one must modify that representation.

To create an instance of class vtkGraphInternals, simply invoke its constructor as follows

```
obj = vtkGraphInternals
```

31.88 vtkHexagonalPrism

31.88.1 Usage

vtkHexagonalPrism is a concrete implementation of vtkCell to represent a linear 3D prism with hexagonal base. Such prism is defined by the twelve points (0-12) where (0,1,2,3,4,5) is the base of the prism which, using the right hand rule, forms a hexagon whose normal points is in the direction of the opposite face (6,7,8,9,10,11).

To create an instance of class vtkHexagonalPrism, simply invoke its constructor as follows

```
obj = vtkHexagonalPrism
```

31.88.2 Methods

The class vtkHexagonalPrism has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHexagonalPrism class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHexagonalPrism = obj.NewInstance ()`
- `vtkHexagonalPrism = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the wedge in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[12])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[36])` - Return the ids of the vertices defining edge/face (‘edgeId‘/‘faceId’). Ids are related to the cell, not to the dataset.

31.89 vtkHexahedron

31.89.1 Usage

vtkHexahedron is a concrete implementation of vtkCell to represent a linear, 3D rectangular hexahedron (e.g., "brick" topology). vtkHexahedron uses the standard isoparametric shape functions for a linear hexahedron. The hexahedron is defined by the eight points (0-7) where (0,1,2,3) is the base of the hexahedron which, using the right hand rule, forms a quadrilaterial whose normal points in the direction of the opposite face (4,5,6,7).

To create an instance of class vtkHexahedron, simply invoke its constructor as follows

```
obj = vtkHexahedron
```

31.89.2 Methods

The class vtkHexahedron has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHexahedron class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHexahedron = obj.NewInstance ()`
- `vtkHexahedron = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.InterpolateFunctions (double pcoords[3], double weights[8])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[24])` - Return the ids of the vertices defining edge/face ('edgeId'/'faceId'). Ids are related to the cell, not to the dataset.

31.90 vtkHierarchicalBoxDataIterator

31.90.1 Usage

To create an instance of class vtkHierarchicalBoxDataIterator, simply invoke its constructor as follows

```
obj = vtkHierarchicalBoxDataIterator
```

31.90.2 Methods

The class vtkHierarchicalBoxDataIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalBoxDataIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHierarchicalBoxDataIterator = obj.NewInstance ()`
- `vtkHierarchicalBoxDataIterator = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCurrentLevel ()` - Returns the level for the current dataset.
- `int = obj.GetCurrentIndex ()` - Returns the dataset index for the current data object. Valid only if the current data is a leaf node i.e. no a composite dataset.

31.91 vtkHierarchicalBoxDataSet

31.91.1 Usage

vtkHierarchicalBoxDataSet is a concrete implementation of vtkCompositeDataSet. The dataset type is restricted to vtkUniformGrid. Each dataset has an associated vtkAMRBox that represents it's region (similar to extent) in space.

.SECTION Warning To compute the cellId of a cell within a vtkUniformGrid with AMRBox=box, you should not use vtkUniformGrid::ComputeCellId(x,y,z) but instead use the following pseudo code: for (int i=0; i¡3; i++) cellDims[i] = box.HiCorner[i] - box.LoCorner[i] + 1; vtkIdType cellId = (z-box.LoCorner[2])*cellDims[0]*cellDims[1] + (y-box.LoCorner[1])*cellDims[0] + (x-box.LoCorner[0]);

NOTE vtkAMRBox is used to compute cell visibility, therefor it should be dimensioned according to the visible region.

To create an instance of class vtkHierarchicalBoxDataSet, simply invoke its constructor as follows

```
obj = vtkHierarchicalBoxDataSet
```

31.91.2 Methods

The class vtkHierarchicalBoxDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalBoxDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHierarchicalBoxDataSet = obj.NewInstance ()`
- `vtkHierarchicalBoxDataSet = obj.SafeDownCast (vtkObject o)`
- `vtkCompositeDataIterator = obj.NewIterator ()` - Return a new iterator (the iterator has to be deleted by user).
- `int = obj.GetDataObjectType ()` - Set the number of refinement levels. This call might cause allocation if the new number of levels is larger than the current one.
- `obj.SetNumberOfLevels (int numLevels)` - Set the number of refinement levels. This call might cause allocation if the new number of levels is larger than the current one.

- `int = obj.GetNumberOfLevels ()` - Returns the number of levels.
- `obj.SetNumberOfDataSets (int level, int numdatasets)` - Set the number of data set at a given level.
- `int = obj.GetNumberOfDataSets (int level)` - Returns the number of data sets available at any level.
- `obj.SetDataSet (vtkCompositeDataIterator iter, vtkDataObject dataObj)` - Set the dataset pointer for a given node. This will resize the number of levels and the number of datasets in the level to fit level, id requested.
- `obj.SetDataSet (int level, int id, int LoCorner[3], int HiCorner[3], vtkUniformGrid dataSet)` - Set the dataset pointer for a given node. This will resize the number of levels and the number of datasets in the level to fit level, id requested.
- `vtkInformation = obj.GetLevelMetaData (int level)` - Returns if meta-data exists for a given level.
- `int = obj.HasLevelMetaData (int level)` - Get meta-data associated with a dataset. This may allocate a new vtkInformation object if none is already present. Use HasMetaData to avoid unnecessary allocations.
- `vtkInformation = obj.GetMetaData (int level, int index)` - Get meta-data associated with a dataset. This may allocate a new vtkInformation object if none is already present. Use HasMetaData to avoid unnecessary allocations.
- `int = obj.HasMetaData (int level, int index)` - Returns if meta-data exists for a given dataset under a given level.
- `obj.SetRefinementRatio (int level, int refRatio)` - Sets the refinement of a given level. The spacing at level level+1 is defined as spacing(level+1) = spacing(level)/refRatio(level). Note that currently, this is not enforced by this class however some algorithms might not function properly if the spacing in the blocks (vtkUniformGrid) does not match the one described by the refinement ratio.
- `int = obj.GetRefinementRatio (int level)` - Returns the refinement of a given level.
- `int = obj.GetRefinementRatio (vtkCompositeDataIterator iter)` - Returns the refinement ratio for the position pointed by the iterator.
- `obj.GenerateVisibilityArrays ()` - Blank lower level cells if they are overlapped by higher level ones.
- `obj.GetScalarRange (double range[])` - Copy the cached scalar range into range.
- `vtkDataObject = obj.GetDataSet (vtkCompositeDataIterator iter)` - Unhiding superclass method.
- `vtkInformation = obj.GetMetaData (vtkCompositeDataIterator iter)` - Unhiding superclass method.
- `int = obj.HasMetaData (vtkCompositeDataIterator iter)` - Given the level and dataset index, returns the flat index provided level and dataset index are valid.
- `int = obj.GetFlatIndex (int level, int index)` - Given the level and dataset index, returns the flat index provided level and dataset index are valid.

31.92 vtkHierarchicalBoxDataSetAlgorithm

31.92.1 Usage

Algorithms that take any type of data object (including composite dataset) and produce a vtkHierarchicalBoxDataSet in the output can subclass from this class.

To create an instance of class vtkHierarchicalBoxDataSetAlgorithm, simply invoke its constructor as follows

```
obj = vtkHierarchicalBoxDataSetAlgorithm
```

31.92.2 Methods

The class vtkHierarchicalBoxDataSetAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalBoxDataSetAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHierarchicalBoxDataSetAlgorithm = obj.NewInstance ()`
- `vtkHierarchicalBoxDataSetAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkHierarchicalBoxDataSet = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkHierarchicalBoxDataSet = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.93 vtkHyperOctree

31.93.1 Usage

An hyperoctree is a dataset where each node has either exactly 2n̂ children or no child at all if the node is a leaf. 'n' is the dimension of the dataset (1 (binary tree), 2 (quadtree) or 3 (octree)). The class name comes from the following paper:

```
@ARTICLE{yau-srihari-1983,
 author={Mann-May Yau and Sargur N. Srihari},
 title={A Hierarchical Data Structure for Multidimensional Digital Images},
 journal={Communications of the ACM},
 month={July},
 year={1983},
```

```
volume={26},
number={7},
pages={504--515}
}
```

Each node is a cell. Attributes are associated with cells, not with points. The geometry is implicitly given by the size of the root node on each axis and position of the center and the orientation. (TODO: review center position and orientation). The geometry is then not limited to an hybercube but can have a rectangular shape. Attributes are associated with leaves. For LOD (Level-Of-Detail) purpose, attributes can be computed on none-leaf nodes by computing the average values from its children (which can be leaves or not).

By construction, an hyperoctree is efficient in memory usage when the geometry is sparse. The LOD feature allows to cull quickly part of the dataset.

A couple of filters can be applied on this dataset: contour, outline, geometry.

* 3D case (octree) for each node, each child index (from 0 to 7) is encoded in the following orientation. It is easy to access each child as a cell of a grid. Note also that the binary representation is relevant, each bit code a side: bit 0 encodes -x side (0) or +x side (1) bit 1 encodes -y side (0) or +y side (1) bit 2 encodes -z side (0) or +z side (2) - the -z side first - 0: -y -x sides - 1: -y +x sides - 2: +y -x sides - 3: +y +x sides

```
             +y
+-+-+        \^
|2|3|        |
+-+-+  O +z  +-> +x
|0|1|
+-+-+
```

- then the +z side, in counter-clockwise - 4: -y -x sides - 5: -y +x sides - 6: +y -x sides - 7: +y +x sides

```
             +y
+-+-+        \^
|6|7|        |
+-+-+  O +z  +-> +x
|4|5|
+-+-+
```

The cases with fewer dimensions are consistent with the octree case:

* Quadtree: in counter-clockwise - 0: -y -x edges - 1: -y +x edges - 2: +y -x edges - 3: +y +x edges

```
        +y
+-+-+   \^
|2|3|   |
+-+-+  O+-> +x
|0|1|
+-+-+
```

* Binary tree:

```
+0+1+  O+-> +x
```

To create an instance of class vtkHyperOctree, simply invoke its constructor as follows

```
obj = vtkHyperOctree
```

31.93.2 Methods

The class vtkHyperOctree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctree class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctree = obj.NewInstance ()`
- `vtkHyperOctree = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Return what type of dataset this is.
- `obj.CopyStructure (vtkDataSet ds)` - Copy the geometric and topological structure of an input rectilinear grid object.
- `int = obj.GetDimension ()` - Return the dimension of the tree (1D:binary tree(2 children), 2D:quadtree(4 children), 3D:octree (8 children))
- `obj.SetDimension (int dim)` - Set the dimension of the tree with 'dim'. See GetDimension() for details.
- `vtkIdType = obj.GetNumberOfCells ()` - Return the number of cells in the dual grid.
- `vtkIdType = obj.GetNumberOfLeaves ()` - Get the number of leaves in the tree.
- `vtkIdType = obj.GetNumberOfPoints ()` - Return the number of points in the dual grid.
- `vtkIdType = obj.GetMaxNumberOfPoints (int level)` - Return the number of points corresponding to an hyperoctree starting at level 'level' where all the leaves at at the last level. In this case, the hyperoctree is like a uniform grid. So this number is the number of points of the uniform grid.
- `vtkIdType = obj.GetMaxNumberOfPointsOnBoundary (int level)` - Return the number of points corresponding to the boundary of an hyperoctree starting at level 'level' where all the leaves at at the last level. In this case, the hyperoctree is like a uniform grid. So this number is the number of points of on the boundary of the uniform grid. For an octree, the boundary are the faces. For a quadtree, the boundary are the edges.
- `vtkIdType = obj.GetMaxNumberOfCellsOnBoundary (int level)` - Return the number of cells corresponding to the boundary of a cell of level 'level' where all the leaves at at the last level.
- `vtkIdType = obj.GetNumberOfLevels ()` - Return the number of levels.
- `obj.SetSize (double , double , double )` - Set the size on each axis.
- `obj.SetSize (double  a[3])` - Set the size on each axis.
- `double = obj. GetSize ()` - Return the size on each axis.
- `obj.SetOrigin (double , double , double )` - Set the origin (position of corner (0,0,0) of the root.
- `obj.SetOrigin (double  a[3])` - Set the origin (position of corner (0,0,0) of the root.
- `double = obj. GetOrigin ()` - Set the origin (position of corner (0,0,0) of the root. Return the origin (position of corner (0,0,0)) of the root.
- `vtkHyperOctreeCursor = obj.NewCellCursor ()` - Create a new cursor: an object that can traverse the cell of an hyperoctree.

- `obj.SubdivideLeaf (vtkHyperOctreeCursor leaf)` - Subdivide node pointed by cursor, only if its a leaf. At the end, cursor points on the node that used to be leaf.
- `obj.CollapseTerminalNode (vtkHyperOctreeCursor node)` - Collapse a node for which all children are leaves. At the end, cursor points on the leaf that used to be a node.
- `double = obj.GetPoint (vtkIdType ptId)` - Get point coordinates with ptId such that: 0 ¡= ptId ¡ NumberOfPoints. THIS METHOD IS NOT THREAD SAFE.
- `obj.GetPoint (vtkIdType id, double x[3])` - Copy point coordinates into user provided array x[3] for specified point id. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `vtkCell = obj.GetCell (vtkIdType cellId)` - Get cell with cellId such that: 0 ¡= cellId ¡ NumberOfCells. THIS METHOD IS NOT THREAD SAFE.
- `obj.GetCell (vtkIdType cellId, vtkGenericCell cell)` - Get cell with cellId such that: 0 ¡= cellId ¡ NumberOfCells. This is a thread-safe alternative to the previous GetCell() method. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `int = obj.GetCellType (vtkIdType cellId)` - Get type of cell with cellId such that: 0 ¡= cellId ¡ NumberOfCells. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `obj.GetCellPoints (vtkIdType cellId, vtkIdList ptIds)` - Topological inquiry to get points defining cell. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `obj.GetPointCells (vtkIdType ptId, vtkIdList cellIds)` - Topological inquiry to get cells using point. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `obj.GetCellNeighbors (vtkIdType cellId, vtkIdList ptIds, vtkIdList cellIds)` - Topological inquiry to get all cells using list of points exclusive of cell specified (e.g., cellId). Note that the list consists of only cells that use ALL the points provided. THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `vtkIdType = obj.FindPoint (double x[3])`
- `obj.Initialize ()` - Restore data object to initial state, THIS METHOD IS NOT THREAD SAFE.
- `int = obj.GetMaxCellSize ()` - Convenience method returns largest cell size in dataset. This is generally used to allocate memory for supporting data structures. This is the number of points of a cell. THIS METHOD IS THREAD SAFE
- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.GetPointsOnFace (vtkHyperOctreeCursor sibling, int face, int level, vtkHyperOctreePointsGrabber` - Get the points of node 'sibling' on its face 'face'.
- `obj.GetPointsOnParentFaces (int faces[3], int level, vtkHyperOctreeCursor cursor, vtkHyperOctreePoi` - Get the points of the parent node of 'cursor' on its faces 'faces' at level 'level' or deeper.
- `obj.GetPointsOnEdge (vtkHyperOctreeCursor sibling, int level, int axis, int k, int j, vtkHyperOctre` - Get the points of node 'sibling' on its edge 'axis','k','j'. If axis==0, the edge is X-aligned and k gives the z coordinate and j the y-coordinate. If axis==1, the edge is Y-aligned and k gives the x coordinate and j the z coordinate. If axis==2, the edge is Z-aligned and k gives the y coordinate and j the x coordinate.

- `obj.GetPointsOnParentEdge (vtkHyperOctreeCursor cursor, int level, int axis, int k, int j, vtkHyper`
  - Get the points of the parent node of 'cursor' on its edge 'axis','k','j' at level 'level' or deeper. If axis==0, the edge is X-aligned and k gives the z coordinate and j the y-coordinate. If axis==1, the edge is Y-aligned and k gives the x coordinate and j the z coordinate. If axis==2, the edge is Z-aligned and k gives the y coordinate and j the x coordinate.
- `obj.GetPointsOnEdge2D (vtkHyperOctreeCursor sibling, int edge, int level, vtkHyperOctreePointsGrabb`
  - Get the points of node 'sibling' on its edge 'edge'.
- `obj.GetPointsOnParentEdge2D (vtkHyperOctreeCursor cursor, int edge, int level, vtkHyperOctreePoints`
  - Get the points of the parent node of 'cursor' on its edge 'edge' at level 'level' or deeper. (edge=0 for -X, 1 for +X, 2 for -Y, 3 for +Y)
- `vtkDataSetAttributes = obj.GetLeafData ()` - A generic way to set the leaf data attributes. This can be either point data for dual or cell data for normal grid.
- `obj.SetDualGridFlag (int flag)` - Switch between returning leaves as cells, or the dual grid.
- `int = obj.GetDualGridFlag ()` - Switch between returning leaves as cells, or the dual grid.
- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value). THIS METHOD IS THREAD SAFE.

31.94 vtkHyperOctreeAlgorithm

31.94.1 Usage

To create an instance of class vtkHyperOctreeAlgorithm, simply invoke its constructor as follows

```
obj = vtkHyperOctreeAlgorithm
```

31.94.2 Methods

The class vtkHyperOctreeAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeAlgorithm = obj.NewInstance ()`
- `vtkHyperOctreeAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkHyperOctree = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkHyperOctree = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput ()`
- `vtkDataObject = obj.GetInput (int port)`
- `vtkHyperOctree = obj.GetHyperOctreeInput (int port)`

- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm.
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm.

31.95 vtkHyperOctreeCursor

31.95.1 Usage

Objects that can traverse hyperoctree nodes. It is an abstract class. Cursors are created by the hyperoctree.
To create an instance of class vtkHyperOctreeCursor, simply invoke its constructor as follows

```
obj = vtkHyperOctreeCursor
```

31.95.2 Methods

The class vtkHyperOctreeCursor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeCursor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeCursor = obj.NewInstance ()`
- `vtkHyperOctreeCursor = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetLeafId ()` - Return the index of the current leaf in the data arrays.
- `int = obj.CurrentIsLeaf ()` - Is the node pointed by the cursor a leaf?
- `int = obj.CurrentIsRoot ()` - Is the node pointed by the cursor the root?
- `int = obj.GetCurrentLevel ()` - Return the level of the node pointed by the cursor.
- `int = obj.GetChildIndex ()` - Return the child number of the current node relative to its parent.
- `int = obj.CurrentIsTerminalNode ()`
- `obj.ToRoot ()` - Move the cursor the root node.
- `obj.ToParent ()` - Move the cursor to the parent of the current node.
- `obj.ToChild (int child)` - Move the cursor to child ‘child’ of the current node.
- `obj.ToSameNode (vtkHyperOctreeCursor other)` - Move the cursor to the same node pointed by ‘other’.
- `int = obj.IsEqual (vtkHyperOctreeCursor other)` - Is ‘this’ equal to ‘other’?
- `vtkHyperOctreeCursor = obj.Clone ()` - Create a copy of ‘this’.
- `int = obj.SameTree (vtkHyperOctreeCursor other)` - Are ‘this’ and ‘other’ pointing on the same hyperoctree?
- `int = obj.GetIndex (int d)` - Return the index in dimension ‘d’, as if the node was a cell of a uniform grid of 1¡¡GetCurrentLevel() cells in each dimension.

- `int = obj.GetNumberOfChildren ()` - Return the number of children for each node of the tree.
- `int = obj.GetDimension ()` - Return the dimension of the tree.
- `obj.MoveToNode (int indices, int level)` - Move to the node described by its indices in each dimension and at a given level. If there is actually a node or a leaf at this location, Found() returns true. Otherwise, Found() returns false and the cursor moves to the closest parent of the query. It can be the root in the worst case.
- `int = obj.Found ()`

31.96 vtkImageAlgorithm

31.96.1 Usage

vtkImageAlgorithm is a filter superclass that hides much of the pipeline complexity. It handles breaking the pipeline execution into smaller extents so that the vtkImageData limits are observed. It also provides support for multithreading. If you don't need any of this functionality, consider using vtkSimpleImageToImageFilter instead. .SECTION See also vtkSimpleImageToImageFilter

To create an instance of class vtkImageAlgorithm, simply invoke its constructor as follows

```
obj = vtkImageAlgorithm
```

31.96.2 Methods

The class vtkImageAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageAlgorithm = obj.NewInstance ()`
- `vtkImageAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkImageData = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkImageData = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `vtkDataObject = obj.GetInput (int port)`
- `vtkDataObject = obj.GetInput ()`

- `vtkImageData = obj.GetImageDataInput (int port)`
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.97 vtkImageData

31.97.1 Usage

vtkImageData is a data object that is a concrete implementation of vtkDataSet. vtkImageData represents a geometric structure that is a topological and geometrical regular array of points. Examples include volumes (voxel data) and pixmaps.

To create an instance of class vtkImageData, simply invoke its constructor as follows

```
obj = vtkImageData
```

31.97.2 Methods

The class vtkImageData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageData = obj.NewInstance ()`
- `vtkImageData = obj.SafeDownCast (vtkObject o)`
- `obj.CopyStructure (vtkDataSet ds)` - Copy the geometric and topological structure of an input image data object.
- `int = obj.GetDataObjectType ()` - Return what type of dataset this is.
- `vtkIdType = obj.GetNumberOfCells ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkIdType = obj.GetNumberOfPoints ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `double = obj.GetPoint (vtkIdType ptId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetPoint (vtkIdType id, double x[3])` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkCell = obj.GetCell (vtkIdType cellId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCell (vtkIdType cellId, vtkGenericCell cell)` - Standard vtkDataSet API methods. See vtkDataSet for more information.

- `obj.GetCellBounds (vtkIdType cellId, double bounds[6])` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkIdType = obj.FindPoint (double x, double y, double z)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkIdType = obj.FindPoint (double x[3])` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `int = obj.GetCellType (vtkIdType cellId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCellPoints (vtkIdType cellId, vtkIdList ptIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetPointCells (vtkIdType ptId, vtkIdList cellIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.ComputeBounds ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `int = obj.GetMaxCellSize ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.Initialize ()` - Restore data object to initial state,
- `obj.SetDimensions (int i, int j, int k)` - Pass your way. This is for backward compatibility only. Use SetExtent() instead. Same as SetExtent(0, i-1, 0, j-1, 0, k-1)
- `obj.SetDimensions (int dims[3])` - Pass your way. This is for backward compatibility only. Use SetExtent() instead. Same as SetExtent(0, dims[0]-1, 0, dims[1]-1, 0, dims[2]-1)
- `int = obj.GetDimensions ()` - Get dimensions of this structured points dataset. It is the number of points on each axis. Dimensions are computed from Extents during this call.
- `obj.GetDimensions (int dims[3])` - Get dimensions of this structured points dataset. It is the number of points on each axis. Dimensions are computed from Extents during this call.
- `int = obj.ComputeStructuredCoordinates (double x[3], int ijk[3], double pcoords[3])` - Convenience function computes the structured coordinates for a point x[3]. The voxel is specified by the array ijk[3], and the parametric coordinates in the cell are specified with pcoords[3]. The function returns a 0 if the point x is outside of the volume, and a 1 if inside the volume.
- `obj.GetVoxelGradient (int i, int j, int k, vtkDataArray s, vtkDataArray g)` - Given structured coordinates (i,j,k) for a voxel cell, compute the eight gradient values for the voxel corners. The order in which the gradient vectors are arranged corresponds to the ordering of the voxel points. Gradient vector is computed by central differences (except on edges of volume where forward difference is used). The scalars s are the scalars from which the gradient is to be computed. This method will treat only 3D structured point datasets (i.e., volumes).
- `obj.GetPointGradient (int i, int j, int k, vtkDataArray s, double g[3])` - Given structured coordinates (i,j,k) for a point in a structured point dataset, compute the gradient vector from the scalar data at that point. The scalars s are the scalars from which the gradient is to be computed. This method will treat structured point datasets of any dimension.
- `int = obj.GetDataDimension ()` - Return the dimensionality of the data.
- `vtkIdType = obj.ComputePointId (int ijk[3])` - Given a location in structured coordinates (i-j-k), return the point id.
- `vtkIdType = obj.ComputeCellId (int ijk[3])` - Given a location in structured coordinates (i-j-k), return the cell id.

- `obj.SetAxisUpdateExtent (int axis, int min, int max)` - Set / Get the extent on just one axis
- `obj.UpdateInformation ()` - Override to copy information from pipeline information to data information for backward compatibility. See vtkDataObject::UpdateInformation for details.
- `obj.SetExtent (int extent[6])` - Set/Get the extent. On each axis, the extent is defined by the index of the first point and the index of the last point. The extent should be set before the "Scalars" are set or allocated. The Extent is stored in the order (X, Y, Z). The dataset extent does not have to start at (0,0,0). (0,0,0) is just the extent of the origin. The first point (the one with Id=0) is at extent (Extent[0],Extent[2],Extent[4]). As for any dataset, a data array on point data starts at Id=0.
- `obj.SetExtent (int x1, int x2, int y1, int y2, int z1, int z2)` - Set/Get the extent. On each axis, the extent is defined by the index of the first point and the index of the last point. The extent should be set before the "Scalars" are set or allocated. The Extent is stored in the order (X, Y, Z). The dataset extent does not have to start at (0,0,0). (0,0,0) is just the extent of the origin. The first point (the one with Id=0) is at extent (Extent[0],Extent[2],Extent[4]). As for any dataset, a data array on point data starts at Id=0.
- `int = obj. GetExtent ()` - Set/Get the extent. On each axis, the extent is defined by the index of the first point and the index of the last point. The extent should be set before the "Scalars" are set or allocated. The Extent is stored in the order (X, Y, Z). The dataset extent does not have to start at (0,0,0). (0,0,0) is just the extent of the origin. The first point (the one with Id=0) is at extent (Extent[0],Extent[2],Extent[4]). As for any dataset, a data array on point data starts at Id=0.
- `long = obj.GetEstimatedMemorySize ()` - Get the estimated size of this data object itself. Should be called after UpdateInformation() and PropagateUpdateExtent() have both been called. This estimate should be fairly accurate since this is structured data.
- `double = obj.GetScalarTypeMin ()` - These returns the minimum and maximum values the ScalarType can hold without overflowing.
- `double = obj.GetScalarTypeMax ()` - These returns the minimum and maximum values the ScalarType can hold without overflowing.
- `int = obj.GetScalarSize ()` - Set the size of the scalar type in bytes.
- `vtkIdType = obj.GetIncrements ()` - Different ways to get the increments for moving around the data. GetIncrements() calls ComputeIncrements() to ensure the increments are up to date.
- `obj.GetIncrements (vtkIdType inc[3])` - Different ways to get the increments for moving around the data. GetIncrements() calls ComputeIncrements() to ensure the increments are up to date.
- `float = obj.GetScalarComponentAsFloat (int x, int y, int z, int component)` - For access to data from tcl
- `obj.SetScalarComponentFromFloat (int x, int y, int z, int component, float v)` - For access to data from tcl
- `double = obj.GetScalarComponentAsDouble (int x, int y, int z, int component)` - For access to data from tcl
- `obj.SetScalarComponentFromDouble (int x, int y, int z, int component, double v)` - For access to data from tcl
- `obj.AllocateScalars ()` - Allocate the vtkScalars object associated with this object.
- `obj.CopyAndCastFrom (vtkImageData inData, int extent[6])` - This method is passed a input and output region, and executes the filter algorithm to fill the output from the input. It just executes a switch statement to call the correct function for the regions data types.

- `obj.CopyAndCastFrom (vtkImageData inData, int x0, int x1, int y0, int y1, int z0, int z1)` - Reallocates and copies to set the Extent to the UpdateExtent. This is used internally when the exact extent is requested, and the source generated more than the update extent.

- `obj.Crop ()` - Reallocates and copies to set the Extent to the UpdateExtent. This is used internally when the exact extent is requested, and the source generated more than the update extent.

- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value). THIS METHOD IS THREAD SAFE.

- `obj.SetSpacing (double , double , double )` - Set the spacing (width,height,length) of the cubical cells that compose the data set.

- `obj.SetSpacing (double  a[3])` - Set the spacing (width,height,length) of the cubical cells that compose the data set.

- `double = obj. GetSpacing ()` - Set the spacing (width,height,length) of the cubical cells that compose the data set.

- `obj.SetOrigin (double , double , double )` - Set/Get the origin of the dataset. The origin is the position in world coordinates of the point of extent (0,0,0). This point does not have to be part of the dataset, in other words, the dataset extent does not have to start at (0,0,0) and the origin can be outside of the dataset bounding box. The origin plus spacing determine the position in space of the points.

- `obj.SetOrigin (double  a[3])` - Set/Get the origin of the dataset. The origin is the position in world coordinates of the point of extent (0,0,0). This point does not have to be part of the dataset, in other words, the dataset extent does not have to start at (0,0,0) and the origin can be outside of the dataset bounding box. The origin plus spacing determine the position in space of the points.

- `double = obj. GetOrigin ()` - Set/Get the origin of the dataset. The origin is the position in world coordinates of the point of extent (0,0,0). This point does not have to be part of the dataset, in other words, the dataset extent does not have to start at (0,0,0) and the origin can be outside of the dataset bounding box. The origin plus spacing determine the position in space of the points.

- `obj.SetScalarTypeToFloat ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToDouble ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToInt ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToUnsignedInt ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToLong ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToUnsignedLong ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToShort ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToUnsignedShort ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToUnsignedChar ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToSignedChar ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToChar ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarType (int )` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `int = obj.GetScalarType ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `string = obj.GetScalarTypeAsString ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetNumberOfScalarComponents (int n)` - Set/Get the number of scalar components for points. As with the SetScalarType method this is setting pipeline info.

- `int = obj.GetNumberOfScalarComponents ()` - Set/Get the number of scalar components for points. As with the SetScalarType method this is setting pipeline info.

- `obj.CopyTypeSpecificInformation (vtkDataObject image)`

- `obj.CopyInformationToPipeline (vtkInformation request, vtkInformation input, vtkInformation output,` - Override these to handle origin, spacing, scalar type, and scalar number of components. See vtkDataObject for details.
- `obj.CopyInformationFromPipeline (vtkInformation request)` - Override these to handle origin, spacing, scalar type, and scalar number of components. See vtkDataObject for details.
- `obj.PrepareForNewData ()` - make the output data ready for new data to be inserted. For most objects we just call Initialize. But for image data we leave the old data in case the memory can be reused.
- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.ComputeInternalExtent (int intExt, int tgtExt, int bnds)` - Given how many pixel are required on a side for bounrary conditions (in bnds), the target extent to traverse, compute the internal extent (the extent for this ImageData that does nto suffer from any boundary conditions) and place it in intExt
- `int = obj.GetExtentType ()` - The extent type is a 3D extent

31.98 vtkImageInPlaceFilter

31.98.1 Usage

vtkImageInPlaceFilter is a filter super class that operates directly on the input region. The data is copied if the requested region has different extent than the input region or some other object is referencing the input region.

To create an instance of class vtkImageInPlaceFilter, simply invoke its constructor as follows

```
obj = vtkImageInPlaceFilter
```

31.98.2 Methods

The class vtkImageInPlaceFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageInPlaceFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageInPlaceFilter = obj.NewInstance ()`
- `vtkImageInPlaceFilter = obj.SafeDownCast (vtkObject o)`

31.99 vtkImageMultipleInputFilter

31.99.1 Usage

vtkImageMultipleInputFilter is a super class for filters that have any number of inputs. Streaming is not available in this class yet.

To create an instance of class vtkImageMultipleInputFilter, simply invoke its constructor as follows

```
obj = vtkImageMultipleInputFilter
```

31.99.2 Methods

The class vtkImageMultipleInputFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMultipleInputFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMultipleInputFilter = obj.NewInstance ()`
- `vtkImageMultipleInputFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (int num, vtkImageData input)` - Set an Input of this filter.
- `obj.AddInput (vtkImageData input)` - Adds an input to the first null position in the input list. Expands the list memory if necessary
- `obj.RemoveInput (vtkImageData input)` - Adds an input to the first null position in the input list. Expands the list memory if necessary
- `vtkImageData = obj.GetInput (int num)` - Get one input to this filter.
- `vtkImageData = obj.GetInput ()` - Get one input to this filter.
- `obj.SetBypass (int )` - Turning bypass on will cause the filter to turn off and simply pass the data from the first input (input0) through. It is implemented for consistency with vtkImageToImageFilter.
- `int = obj.GetBypass ()` - Turning bypass on will cause the filter to turn off and simply pass the data from the first input (input0) through. It is implemented for consistency with vtkImageToImageFilter.
- `obj.BypassOn ()` - Turning bypass on will cause the filter to turn off and simply pass the data from the first input (input0) through. It is implemented for consistency with vtkImageToImageFilter.
- `obj.BypassOff ()` - Turning bypass on will cause the filter to turn off and simply pass the data from the first input (input0) through. It is implemented for consistency with vtkImageToImageFilter.
- `obj.SetNumberOfThreads (int )` - Get/Set the number of threads to create when rendering
- `int = obj.GetNumberOfThreadsMinValue ()` - Get/Set the number of threads to create when rendering
- `int = obj.GetNumberOfThreadsMaxValue ()` - Get/Set the number of threads to create when rendering
- `int = obj.GetNumberOfThreads ()` - Get/Set the number of threads to create when rendering
- `int = obj.SplitExtent (int splitExt[6], int startExt[6], int num, int total)` - Putting this here until I merge graphics and imaging streaming.

31.100 vtkImageMultipleInputOutputFilter

31.100.1 Usage

vtkImageMultipleInputOutputFilter is a super class for filters that have any number of inputs. Streaming is not available in this class yet.

To create an instance of class vtkImageMultipleInputOutputFilter, simply invoke its constructor as follows

```
obj = vtkImageMultipleInputOutputFilter
```

31.100.2 Methods

The class vtkImageMultipleInputOutputFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMultipleInputOutputFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMultipleInputOutputFilter = obj.NewInstance ()`
- `vtkImageMultipleInputOutputFilter = obj.SafeDownCast (vtkObject o)`
- `vtkImageData = obj.GetOutput (int num)` - Get one input to this filter.
- `vtkImageData = obj.GetOutput ()` - Get one input to this filter.

31.101 vtkImageSource

31.101.1 Usage

vtkImageSource is the superclass for all imaging sources and filters. The method Update(), called by the cache, is the major interface to the source.

To create an instance of class vtkImageSource, simply invoke its constructor as follows

```
obj = vtkImageSource
```

31.101.2 Methods

The class vtkImageSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageSource = obj.NewInstance ()`
- `vtkImageSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutput (vtkImageData output)` - Get the output of this source.
- `vtkImageData = obj.GetOutput ()` - Get the output of this source.
- `vtkImageData = obj.GetOutput (int idx)` - Get the output of this source.

31.102 vtkImageToImageFilter

31.102.1 Usage

vtkImageToImageFilter is a filter superclass that hides much of the pipeline complexity. It handles breaking the pipeline execution into smaller extents so that the vtkImageData limits are observed. It also provides support for multithreading. If you don't need any of this functionality, consider using vtkSimpleImageToImageFilter instead. .SECTION Warning This used to be the parent class for most imaging filter in VTK4.x,

now this role has been replaced by vtkImageAlgorithm. You should consider using vtkImageAlgorithm instead, when writing filter for VTK5 and above. This class was kept to ensure full backward compatibility. .SECTION See also vtkSimpleImageToImageFilter vtkImageSpatialFilter vtkImageAlgorithm

To create an instance of class vtkImageToImageFilter, simply invoke its constructor as follows

```
obj = vtkImageToImageFilter
```

31.102.2 Methods

The class vtkImageToImageFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageToImageFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageToImageFilter = obj.NewInstance ()`
- `vtkImageToImageFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImageData input)` - Set the Input of a filter.
- `vtkImageData = obj.GetInput ()` - Set the Input of a filter.
- `obj.SetBypass (int )` - Obsolete feature - do not use.
- `obj.BypassOn ()` - Obsolete feature - do not use.
- `obj.BypassOff ()` - Obsolete feature - do not use.
- `int = obj.GetBypass ()` - Obsolete feature - do not use.
- `obj.ThreadedExecute (vtkImageData inData, vtkImageData outData, int extent[6], int threadId)` - If the subclass does not define an Execute method, then the task will be broken up, multiple threads will be spawned, and each thread will call this method. It is public so that the thread functions can call this method.
- `obj.SetNumberOfThreads (int )` - Get/Set the number of threads to create when rendering
- `int = obj.GetNumberOfThreadsMinValue ()` - Get/Set the number of threads to create when rendering
- `int = obj.GetNumberOfThreadsMaxValue ()` - Get/Set the number of threads to create when rendering
- `int = obj.GetNumberOfThreads ()` - Get/Set the number of threads to create when rendering
- `obj.SetInputMemoryLimit (int )`
- `long = obj.GetInputMemoryLimit ()`
- `int = obj.SplitExtent (int splitExt[6], int startExt[6], int num, int total)` - Putting this here until I merge graphics and imaging streaming.

31.103 vtkImageToStructuredPoints

31.103.1 Usage

vtkImageToStructuredPoints changes an image cache format to a structured points dataset. It takes an Input plus an optional VectorInput. The VectorInput converts the RGB scalar components of the VectorInput to vector pointdata attributes. This filter will try to reference count the data but in some cases it must make a copy.

To create an instance of class vtkImageToStructuredPoints, simply invoke its constructor as follows

```
obj = vtkImageToStructuredPoints
```

31.103.2 Methods

The class vtkImageToStructuredPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageToStructuredPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageToStructuredPoints = obj.NewInstance ()`
- `vtkImageToStructuredPoints = obj.SafeDownCast (vtkObject o)`
- `obj.SetVectorInput (vtkImageData input)` - Set/Get the input object from the image pipeline.
- `vtkImageData = obj.GetVectorInput ()` - Set/Get the input object from the image pipeline.
- `vtkStructuredPoints = obj.GetStructuredPointsOutput ()` - Get the output of the filter.

31.104 vtkImageTwoInputFilter

31.104.1 Usage

vtkImageTwoInputFilter handles two inputs. It is just a subclass of vtkImageMultipleInputFilter with some methods that are specific to two inputs. Although the inputs are labeled input1 and input2, they are stored in an array indexed starting at 0.

To create an instance of class vtkImageTwoInputFilter, simply invoke its constructor as follows

```
obj = vtkImageTwoInputFilter
```

31.104.2 Methods

The class vtkImageTwoInputFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageTwoInputFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageTwoInputFilter = obj.NewInstance ()`
- `vtkImageTwoInputFilter = obj.SafeDownCast (vtkObject o)`

- `obj.SetInput1 (vtkImageData input)` - Set the Input1 of this filter. If a ScalarType has not been set, then the ScalarType of the input is used.
- `obj.SetInput2 (vtkImageData input)` - Set the Input2 of this filter. If a ScalarType has not been set, then the ScalarType of the input is used.
- `vtkImageData = obj.GetInput1 ()` - Get the inputs to this filter.
- `vtkImageData = obj.GetInput2 ()` - Get the inputs to this filter.

31.105 vtkImplicitBoolean

31.105.1 Usage

vtkImplicitBoolean is an implicit function consisting of boolean combinations of implicit functions. The class has a list of functions (FunctionList) that are combined according to a specified operator (VTK_UNION or VTK_INTERSECTION or VTK_DIFFERENCE). You can use nested combinations of vtkImplicitFunction's (and/or vtkImplicitBoolean) to create elaborate implicit functions. vtkImplicitBoolean is a concrete implementation of vtkImplicitFunction.

The operators work as follows. The VTK_UNION operator takes the minimum value of all implicit functions. The VTK_INTERSECTION operator takes the maximum value of all implicit functions. The VTK_DIFFERENCE operator subtracts the 2nd through last implicit functions from the first. The VTK_UNION_OF_MAGNI takes the minimum absolute value of the implicit functions.

To create an instance of class vtkImplicitBoolean, simply invoke its constructor as follows

```
obj = vtkImplicitBoolean
```

31.105.2 Methods

The class vtkImplicitBoolean has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitBoolean class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitBoolean = obj.NewInstance ()`
- `vtkImplicitBoolean = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])` - Evaluate boolean combinations of implicit function using current operator.
- `double = obj.EvaluateFunction (double x, double y, double z)` - Evaluate boolean combinations of implicit function using current operator.
- `obj.EvaluateGradient (double x[3], double g[3])` - Evaluate gradient of boolean combination.
- `long = obj.GetMTime ()` - Override modified time retrieval because of object dependencies.
- `obj.AddFunction (vtkImplicitFunction in)` - Add another implicit function to the list of functions.
- `obj.RemoveFunction (vtkImplicitFunction in)` - Remove a function from the list of implicit functions to boolean.
- `vtkImplicitFunctionCollection = obj.GetFunction ()` - Return the collection of implicit functions.

- `obj.SetOperationType (int )` - Specify the type of boolean operation.
- `int = obj.GetOperationTypeMinValue ()` - Specify the type of boolean operation.
- `int = obj.GetOperationTypeMaxValue ()` - Specify the type of boolean operation.
- `int = obj.GetOperationType ()` - Specify the type of boolean operation.
- `obj.SetOperationTypeToUnion ()` - Specify the type of boolean operation.
- `obj.SetOperationTypeToIntersection ()` - Specify the type of boolean operation.
- `obj.SetOperationTypeToDifference ()` - Specify the type of boolean operation.
- `obj.SetOperationTypeToUnionOfMagnitudes ()` - Specify the type of boolean operation.
- `string = obj.GetOperationTypeAsString ()` - Specify the type of boolean operation.

31.106 vtkImplicitDataSet

31.106.1 Usage

vtkImplicitDataSet treats any type of dataset as if it were an implicit function. This means it computes a function value and gradient. vtkImplicitDataSet is a concrete implementation of vtkImplicitFunction.

vtkImplicitDataSet computes the function (at the point x) by performing cell interpolation. That is, it finds the cell containing x, and then uses the cell's interpolation functions to compute an interpolated scalar value at x. (A similar approach is used to find the gradient, if requested.) Points outside of the dataset are assigned the value of the ivar OutValue, and the gradient value OutGradient.

To create an instance of class vtkImplicitDataSet, simply invoke its constructor as follows

```
obj = vtkImplicitDataSet
```

31.106.2 Methods

The class vtkImplicitDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitDataSet = obj.NewInstance ()`
- `vtkImplicitDataSet = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Return the MTime also considering the DataSet dependency.
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double n[3])`
- `obj.SetDataSet (vtkDataSet )` - Set / get the dataset used for the implicit function evaluation.
- `vtkDataSet = obj.GetDataSet ()` - Set / get the dataset used for the implicit function evaluation.
- `obj.SetOutValue (double )` - Set / get the function value to use for points outside of the dataset.

- `double = obj.GetOutValue ()` - Set / get the function value to use for points outside of the dataset.
- `obj.SetOutGradient (double , double , double )` - Set / get the function gradient to use for points outside of the dataset.
- `obj.SetOutGradient (double a[3])` - Set / get the function gradient to use for points outside of the dataset.
- `double = obj. GetOutGradient ()` - Set / get the function gradient to use for points outside of the dataset.

31.107 vtkImplicitHalo

31.107.1 Usage

vtkImplicitHalo evaluates to 1.0 for each position in the sphere of a given center and radius Radius*(1-FadeOut). It evaluates to 0.0 for each position out the sphere of a given Center and radius Radius. It fades out linearly from 1.0 to 0.0 for points in a radius from Radius*(1-FadeOut) to Radius. vtkImplicitHalo is a concrete implementation of vtkImplicitFunction. It is useful as an input to vtkSampleFunction to generate an 2D image of an halo. It is used this way by vtkShadowMapPass.

To create an instance of class vtkImplicitHalo, simply invoke its constructor as follows

```
obj = vtkImplicitHalo
```

31.107.2 Methods

The class vtkImplicitHalo has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitHalo class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitHalo = obj.NewInstance ()`
- `vtkImplicitHalo = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double g[3])`
- `obj.SetRadius (double )` - Radius of the sphere.
- `double = obj.GetRadius ()` - Radius of the sphere.
- `obj.SetCenter (double , double , double )` - Center of the sphere.
- `obj.SetCenter (double a[3])` - Center of the sphere.
- `double = obj. GetCenter ()` - Center of the sphere.
- `obj.SetFadeOut (double )` - FadeOut ratio. Valid values are between 0.0 and 1.0.
- `double = obj.GetFadeOut ()` - FadeOut ratio. Valid values are between 0.0 and 1.0.

31.108 vtkImplicitSelectionLoop

31.108.1 Usage

vtkImplicitSelectionLoop computes the implicit function value and function gradient for a irregular, cylinder-like object whose cross section is defined by a set of points forming a loop. The loop need not be convex nor its points coplanar. However, the loop must be non-self-intersecting when projected onto the plane defined by the accumulated cross product around the loop (i.e., the axis of the loop). (Alternatively, you can specify the normal to use.)

The following procedure is used to compute the implicit function value for a point x. Each point of the loop is first projected onto the plane defined by the loop normal. This forms a polygon. Then, to evaluate the implicit function value, inside/outside tests are used to determine if x is inside the polygon, and the distance to the loop boundary is computed (negative values are inside the loop).

One example application of this implicit function class is to draw a loop on the surface of a mesh, and use the loop to clip or extract cells from within the loop. Remember, the selection loop is "infinite" in length, you can use a plane (in boolean combination) to cap the extent of the selection loop. Another trick is to use a connectivity filter to extract the closest region to a given point (i.e., one of the points used to define the selection loop).

To create an instance of class vtkImplicitSelectionLoop, simply invoke its constructor as follows

```
obj = vtkImplicitSelectionLoop
```

31.108.2 Methods

The class vtkImplicitSelectionLoop has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitSelectionLoop class.

- `string = obj.GetClassName ()` - Standard VTK methods for printing and type information.
- `int = obj.IsA (string name)` - Standard VTK methods for printing and type information.
- `vtkImplicitSelectionLoop = obj.NewInstance ()` - Standard VTK methods for printing and type information.
- `vtkImplicitSelectionLoop = obj.SafeDownCast (vtkObject o)` - Standard VTK methods for printing and type information.
- `double = obj.EvaluateFunction (double x[3])` - Evaluate selection loop returning a signed distance.
- `double = obj.EvaluateFunction (double x, double y, double z)` - Evaluate selection loop returning a signed distance.
- `obj.EvaluateGradient (double x[3], double n[3])` - Evaluate selection loop returning the gradient.
- `obj.SetLoop (vtkPoints )` - Set/Get the array of point coordinates defining the loop. There must be at least three points used to define a loop.
- `vtkPoints = obj.GetLoop ()` - Set/Get the array of point coordinates defining the loop. There must be at least three points used to define a loop.
- `obj.SetAutomaticNormalGeneration (int )` - Turn on/off automatic normal generation. By default, the normal is computed from the accumulated cross product of the edges. You can also specify the normal to use.

- `int = obj.GetAutomaticNormalGeneration ()` - Turn on/off automatic normal generation. By default, the normal is computed from the accumulated cross product of the edges. You can also specify the normal to use.
- `obj.AutomaticNormalGenerationOn ()` - Turn on/off automatic normal generation. By default, the normal is computed from the accumulated cross product of the edges. You can also specify the normal to use.
- `obj.AutomaticNormalGenerationOff ()` - Turn on/off automatic normal generation. By default, the normal is computed from the accumulated cross product of the edges. You can also specify the normal to use.
- `obj.SetNormal (double , double , double )` - Set / get the normal used to determine whether a point is inside or outside the selection loop.
- `obj.SetNormal (double a[3])` - Set / get the normal used to determine whether a point is inside or outside the selection loop.
- `double = obj. GetNormal ()` - Set / get the normal used to determine whether a point is inside or outside the selection loop.
- `long = obj.GetMTime ()` - Overload GetMTime() because we depend on the Loop

31.109 vtkImplicitSum

31.109.1 Usage

vtkImplicitSum produces a linear combination of other implicit functions. The contribution of each function is weighted by a scalar coefficient. The NormalizeByWeight option normalizes the output so that the scalar weights add up to 1. Note that this function gives accurate sums and gradients only if the input functions are linear.

To create an instance of class vtkImplicitSum, simply invoke its constructor as follows

```
obj = vtkImplicitSum
```

31.109.2 Methods

The class vtkImplicitSum has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitSum class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitSum = obj.NewInstance ()`
- `vtkImplicitSum = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])` - Evaluate implicit function using current functions and weights.
- `double = obj.EvaluateFunction (double x, double y, double z)` - Evaluate implicit function using current functions and weights.
- `obj.EvaluateGradient (double x[3], double g[3])` - Evaluate gradient of the weighted sum of functions. Input functions should be linear.

- `long = obj.GetMTime ()` - Override modified time retrieval because of object dependencies.
- `obj.AddFunction (vtkImplicitFunction in, double weight)` - Add another implicit function to the list of functions, along with a weighting factor.
- `obj.AddFunction (vtkImplicitFunction in)` - Remove all functions from the list.
- `obj.RemoveAllFunctions ()` - Remove all functions from the list.
- `obj.SetFunctionWeight (vtkImplicitFunction f, double weight)` - Set the weight (coefficient) of the given function to be weight.
- `obj.SetNormalizeByWeight (int )` - When calculating the function and gradient values of the composite function, setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the gradient vector. By default, NormalizeByWeight is off.
- `int = obj.GetNormalizeByWeight ()` - When calculating the function and gradient values of the composite function, setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the gradient vector. By default, NormalizeByWeight is off.
- `obj.NormalizeByWeightOn ()` - When calculating the function and gradient values of the composite function, setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the gradient vector. By default, NormalizeByWeight is off.
- `obj.NormalizeByWeightOff ()` - When calculating the function and gradient values of the composite function, setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the gradient vector. By default, NormalizeByWeight is off.

31.110 vtkImplicitVolume

31.110.1 Usage

vtkImplicitVolume treats a volume (e.g., structured point dataset) as if it were an implicit function. This means it computes a function value and gradient. vtkImplicitVolume is a concrete implementation of vtkImplicitFunction.

vtkImplicitDataSet computes the function (at the point x) by performing cell interpolation. That is, it finds the cell containing x, and then uses the cell's interpolation functions to compute an interpolated scalar value at x. (A similar approach is used to find the gradient, if requested.) Points outside of the dataset are assigned the value of the ivar OutValue, and the gradient value OutGradient.

To create an instance of class vtkImplicitVolume, simply invoke its constructor as follows

```
obj = vtkImplicitVolume
```

31.110.2 Methods

The class vtkImplicitVolume has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitVolume class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkImplicitVolume = obj.NewInstance ()`
- `vtkImplicitVolume = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Returns the mtime also considering the volume. This also calls Update on the volume, and it therefore must be called before the function is evaluated.
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double n[3])`
- `obj.SetVolume (vtkImageData )` - Specify the volume for the implicit function.
- `vtkImageData = obj.GetVolume ()` - Specify the volume for the implicit function.
- `obj.SetOutValue (double )` - Set the function value to use for points outside of the dataset.
- `double = obj.GetOutValue ()` - Set the function value to use for points outside of the dataset.
- `obj.SetOutGradient (double , double , double )` - Set the function gradient to use for points outside of the dataset.
- `obj.SetOutGradient (double  a[3])` - Set the function gradient to use for points outside of the dataset.
- `double = obj. GetOutGradient ()` - Set the function gradient to use for points outside of the dataset.

31.111 vtkImplicitWindowFunction

31.111.1 Usage

vtkImplicitWindowFunction is used to modify the output of another implicit function to lie within a specified "window", or function range. This can be used to add "thickness" to cutting or clipping functions.

This class works as follows. First, it evaluates the function value of the user-specified implicit function. Then, based on the window range specified, it maps the function value into the window values specified.

To create an instance of class vtkImplicitWindowFunction, simply invoke its constructor as follows

```
obj = vtkImplicitWindowFunction
```

31.111.2 Methods

The class vtkImplicitWindowFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitWindowFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitWindowFunction = obj.NewInstance ()`
- `vtkImplicitWindowFunction = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double n[3])`

- `obj.SetImplicitFunction (vtkImplicitFunction )` - Specify an implicit function to operate on.
- `vtkImplicitFunction = obj.GetImplicitFunction ()` - Specify an implicit function to operate on.
- `obj.SetWindowRange (double , double )` - Specify the range of function values which are considered to lie within the window. WindowRange[0] is assumed to be less than WindowRange[1].
- `obj.SetWindowRange (double a[2])` - Specify the range of function values which are considered to lie within the window. WindowRange[0] is assumed to be less than WindowRange[1].
- `double = obj. GetWindowRange ()` - Specify the range of function values which are considered to lie within the window. WindowRange[0] is assumed to be less than WindowRange[1].
- `obj.SetWindowValues (double , double )` - Specify the range of output values that the window range is mapped into. This is effectively a scaling and shifting of the original function values.
- `obj.SetWindowValues (double a[2])` - Specify the range of output values that the window range is mapped into. This is effectively a scaling and shifting of the original function values.
- `double = obj. GetWindowValues ()` - Specify the range of output values that the window range is mapped into. This is effectively a scaling and shifting of the original function values.
- `long = obj.GetMTime ()` - Override modified time retrieval because of object dependencies.
- `obj.Register (vtkObjectBase o)` - Participate in garbage collection.
- `obj.UnRegister (vtkObjectBase o)` - Participate in garbage collection.

31.112 vtkIncrementalOctreeNode

31.112.1 Usage

Octree nodes serve as spatial sub-division primitives to build the search structure of an incremental octree in a recursive top-down manner. The hierarchy takes the form of a tree-like representation by which a parent node contains eight mutually non-overlapping child nodes. Each child is assigned with an axis-aligned rectangular volume (Spatial Bounding Box) and the eight children together cover exactly the same region as governed by their parent. The eight child nodes / octants are ordered as

(xBBoxMin, xBBoxMid] & (yBBoxMin, yBBoxMid] & (zBBoxMin, zBBoxMid] , (xBBoxMid, xBBoxMax] & (yBBoxMin, yBBoxMid] & (zBBoxMin, zBBoxMid] , (xBBoxMin, xBBoxMid] & (yBBoxMid, yBBoxMax] & (zBBoxMin, zBBoxMid] , (xBBoxMid, xBBoxMax] & (yBBoxMid, yBBoxMax] & (zBBoxMin, zBBoxMid] , (xBBoxMin, xBBoxMid] & (yBBoxMin, yBBoxMid] & (zBBoxMid, zBBoxMax] , (xBBoxMid, xBBoxMax] & (yBBoxMin, yBBoxMid] & (zBBoxMid, zBBoxMax] , (xBBoxMin, xBBoxMid] & (yBBoxMid, yBBoxMax] & (zBBoxMid, zBBoxMax] , (xBBoxMid, xBBoxMax] & (yBBoxMid, yBBoxMax] & (zBBoxMid, zBBoxMax] ,

where xrange & yRange & zRange defines the region of each 3D octant. In addition, the points falling within and registered, by means of point indices, in the parent node are distributed to the child nodes for delegated maintenance. In fact, only leaf nodes, i.e., those without any descendants, actually store point indices while each node, regardless of a leaf or non- leaf node, keeps a dynamically updated Data Bounding Box of the inhabitant points, if any. Given a maximum number of points per leaf node, an octree is initialized with an empty leaf node that is then recursively sub-divided, but only on demand as points are incrementally inserted, to construct a populated tree.

Please note that this octree node class is able to handle a large number of EXACTLY duplicate points that is greater than the specified maximum number of points per leaf node. In other words, as an exception, a leaf node may maintain an arbitrary number of exactly duplicate points to deal with possible extreme cases.

To create an instance of class vtkIncrementalOctreeNode, simply invoke its constructor as follows

```
obj = vtkIncrementalOctreeNode
```

31.112.2 Methods

The class vtkIncrementalOctreeNode has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIncrementalOctreeNode class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIncrementalOctreeNode = obj.NewInstance ()`
- `vtkIncrementalOctreeNode = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfPoints ()` - Get the number of points inside or under this node.
- `vtkIdList = obj.GetPointIdSet ()` - Get the list of point indices, NULL for a non-leaf node.
- `obj.DeleteChildNodes ()` - Delete the eight child nodes.
- `obj.SetBounds (double x1, double x2, double y1, double y2, double z1, double z2)` - Set the spatial bounding box of the node. This function sets a default data bounding box.
- `obj.GetBounds (double bounds[6]) const` - Get the spatial bounding box of the node. The values are returned via an array in order of: x_min, x_max, y_min, y_max, z_min, z_max.
- `double = obj. GetMinBounds ()` - Get access to MinBounds. Do not free this pointer.
- `double = obj. GetMaxBounds ()` - Get access to MaxBounds. Do not free this pointer.
- `int = obj.IsLeaf ()` - Determine which specific child / octant contains a given point. Note that the point is assumed to be inside this node and no checking is performed on the inside issue.
- `int = obj.GetChildIndex (double point[3])` - Determine which specific child / octant contains a given point. Note that the point is assumed to be inside this node and no checking is performed on the inside issue.
- `vtkIncrementalOctreeNode = obj.GetChild (int i)` - A point is in a node if and only if MinBounds[i] ¡ p[i] ¡= MaxBounds[i], which allows a node to be divided into eight non-overlapping children.
- `int = obj.ContainsPoint (double pnt[3])` - A point is in a node if and only if MinBounds[i] ¡ p[i] ¡= MaxBounds[i], which allows a node to be divided into eight non-overlapping children.
- `int = obj.ContainsPointByData (double pnt[3])` - A point is in a node, in terms of data, if and only if MinDataBounds[i] ¡= p[i] ¡= MaxDataBounds[i].
- `double = obj.GetDistance2ToInnerBoundary (double point[3], vtkIncrementalOctreeNode rootNode)` - Given a point inside this node, get the minimum squared distance to all inner boundaries. An inner boundary is a node's face that is shared by another non-root node.
- `double = obj.GetDistance2ToBoundary (double point[3], vtkIncrementalOctreeNode rootNode, int checkI` - Compute the minimum squared distance from a point to this node, with all six boundaries considered. The data bounding box is checked if checkData is non-zero.
- `double = obj.GetDistance2ToBoundary (double point[3], double closest[3], vtkIncrementalOctreeNode r` - Compute the minimum squared distance from a point to this node, with all six boundaries considered. The data bounding box is checked if checkData is non-zero. The closest on-boundary point is returned via closest.
- `obj.ExportAllPointIdsByInsertion (vtkIdList idList)` - Export all the indices of the points (contained in or under this node) by inserting them to an allocated vtkIdList via vtkIdList::InsertNextId().

31.113 vtkIncrementalOctreePointLocator

31.113.1 Usage

As opposed to the uniform bin-based search structure (adopted in class vtkPointLocator) with a fixed spatial resolution, an octree mechanism employs a hierarchy of tree-like sub-division of the 3D data domain. Thus it enables data-aware multi-resolution and accordingly accelerated point location as well as insertion, particularly when handling a radically imbalanced layout of points as not uncommon in datasets defined on adaptive meshes. Compared to a static point locator supporting pure location functionalities through some search structure established from a fixed set of points, an incremental point locator allows for, in addition, point insertion capabilities, with the search structure maintaining a dynamically increasing number of points. Class vtkIncrementalOctreePointLocator is an octree-based accelerated implementation of the functionalities of the uniform bin-based incremental point locator vtkPointLocator. For point location, an octree is built by accessing a vtkDataSet, specifically a vtkPointSet. For point insertion, an empty octree is inited and then incrementally populated as points are inserted. Three increasingly complex point insertion modes, i.e., direct check-free insertion, zero tolerance insertion, and non-zero tolerance insertion, are supported. In fact, the octree used in the point location mode is actually constructed via direct check-free point insertion. This class also provides a polygonal representation of the octree boundary.

To create an instance of class vtkIncrementalOctreePointLocator, simply invoke its constructor as follows

```
obj = vtkIncrementalOctreePointLocator
```

31.113.2 Methods

The class vtkIncrementalOctreePointLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIncrementalOctreePointLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIncrementalOctreePointLocator = obj.NewInstance ()`
- `vtkIncrementalOctreePointLocator = obj.SafeDownCast (vtkObject o)`
- `obj.SetMaxPointsPerLeaf (int )` - Set/Get the maximum number of points that a leaf node may maintain. Note that the actual number of points maintained by a leaf node might exceed this threshold if there is a large number (equal to or greater than the threshold) of exactly duplicate points (with zero distance) to be inserted (e.g., to construct an octree for subsequent point location) in extreme cases. Respecting this threshold in such scenarios would cause endless node sub-division. Thus this threshold is broken, but only in case of such situations.
- `int = obj.GetMaxPointsPerLeafMinValue ()` - Set/Get the maximum number of points that a leaf node may maintain. Note that the actual number of points maintained by a leaf node might exceed this threshold if there is a large number (equal to or greater than the threshold) of exactly duplicate points (with zero distance) to be inserted (e.g., to construct an octree for subsequent point location) in extreme cases. Respecting this threshold in such scenarios would cause endless node sub-division. Thus this threshold is broken, but only in case of such situations.
- `int = obj.GetMaxPointsPerLeafMaxValue ()` - Set/Get the maximum number of points that a leaf node may maintain. Note that the actual number of points maintained by a leaf node might exceed this threshold if there is a large number (equal to or greater than the threshold) of exactly duplicate points (with zero distance) to be inserted (e.g., to construct an octree for subsequent point location) in extreme cases. Respecting this threshold in such scenarios would cause endless node sub-division. Thus this threshold is broken, but only in case of such situations.

- `int = obj.GetMaxPointsPerLeaf ()` - Set/Get the maximum number of points that a leaf node may maintain. Note that the actual number of points maintained by a leaf node might exceed this threshold if there is a large number (equal to or greater than the threshold) of exactly duplicate points (with zero distance) to be inserted (e.g., to construct an octree for subsequent point location) in extreme cases. Respecting this threshold in such scenarios would cause endless node sub-division. Thus this threshold is broken, but only in case of such situations.
- `obj.SetBuildCubicOctree (int )` - Set/Get whether the search octree is built as a cubic shape or not.
- `int = obj.GetBuildCubicOctree ()` - Set/Get whether the search octree is built as a cubic shape or not.
- `obj.BuildCubicOctreeOn ()` - Set/Get whether the search octree is built as a cubic shape or not.
- `obj.BuildCubicOctreeOff ()` - Set/Get whether the search octree is built as a cubic shape or not.
- `vtkPoints = obj.GetLocatorPoints ()` - Get access to the vtkPoints object in which point coordinates are stored for either point location or point insertion.
- `obj.Initialize ()` - Delete the octree search structure.
- `obj.FreeSearchStructure ()` - Delete the octree search structure.
- `obj.GetBounds (double bounds)` - Get the spatial bounding box of the octree.
- `int = obj.GetNumberOfPoints ()` - Get the number of points maintained by the octree.
- `vtkIdType = obj.FindClosestInsertedPoint (double x[3])` - Given a point x assumed to be covered by the octree, return the index of the closest in-octree point regardless of the associated minimum squared distance relative to the squared insertion-tolerance distance. This method is used when performing incremental point insertion. Note -1 indicates that no point is found. InitPointInsertion() should have been called in advance.
- `obj.GenerateRepresentation (int nodeLevel, vtkPolyData polysData)` - Create a polygonal representation of the octree boundary (from the root node to a specified level).
- `obj.BuildLocator ()` - Load points from a dataset to construct an octree for point location. This function resorts to InitPointInsertion() to fulfill some of the work.
- `vtkIdType = obj.FindClosestPoint (double x[3])` - Given a point x, return the id of the closest point. BuildLocator() should have been called prior to this function. This method is thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `vtkIdType = obj.FindClosestPoint (double x, double y, double z)` - Given a point (x, y, z), return the id of the closest point. Note that BuildLocator() should have been called prior to this function. This method is thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `vtkIdType = obj.FindClosestPoint (double x[3], double miniDist2)` - Given a point x, return the id of the closest point and the associated minimum squared distance (via miniDist2). Note BuildLocator() should have been called prior to this function. This method is thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `vtkIdType = obj.FindClosestPoint (double x, double y, double z, double miniDist2)` - Given a point (x, y, z), return the id of the closest point and the associated minimum squared distance (via miniDist2). BuildLocator() should have been called prior to this function. This method is thread safe if BuildLocator() is directly or indirectly called from a single thread first.

- `obj.FindPointsWithinRadius (double R, double x[3], vtkIdList result)` - Find all points within a radius R relative to a given point x. The returned point ids (stored in result) are not sorted in any way. BuildLocator() should have been called prior to this function. This method is thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.FindPointsWithinSquaredRadius (double R2, double x[3], vtkIdList result)` - Find all points within a squared radius R2 relative to a given point x. The returned point ids (stored in result) are not sorted in any way. BuildLocator() should have been called prior to this function. This method is thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.FindClosestNPoints (int N, double x[3], vtkIdList result)` - Find the closest N points to a given point. The returned point ids (via result) are sorted from closest to farthest. BuildLocator() should have been called prior to this function. This method is thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `int = obj.InitPointInsertion (vtkPoints points, double bounds[6])` - Initialize the point insertion process. points is an object, storing 3D point coordinates, to which incremental point insertion put coordinates. It is created and provided by an external VTK class. Argument bounds represents the spatial bounding box, into which the points fall. In fact, an adjusted version of the bounding box is used to build the octree to make sure no any point (to be inserted) falls outside the octree. This function is not thread safe.
- `int = obj.InitPointInsertion (vtkPoints points, double bounds[6], vtkIdType estSize)` - Initialize the point insertion process. points is an object, storing 3D point coordinates, to which incremental point insertion put coordinates. It is created and provided by an external VTK class. Argument bounds represents the spatial bounding box, into which the points fall. In fact, an adjusted version of the bounding box is used to build the octree to make sure no any point (to be inserted) falls outside the octree. Argument estSize specifies the initial estimated size of the vtkPoints object. This function is not thread safe.
- `vtkIdType = obj.IsInsertedPoint (double x[3])` - Determine whether or not a given point has been inserted into the octree. Return the id of the already inserted point if true, otherwise return -1. InitPointInsertion() should have been called in advance.
- `vtkIdType = obj.IsInsertedPoint (double x, double y, double z)` - Determine whether or not a given point has been inserted into the octree. Return the id of the already inserted point if true, otherwise return -1. InitPointInsertion() should have been called in advance.
- `obj.InsertPoint (vtkIdType ptId, double x[3])` - Insert a given point into the octree with a specified point index ptId. InitPointInsertion() should have been called prior to this function. In addition, IsInsertedPoint() should have been called in advance to ensure that the given point has not been inserted unless point duplication is allowed (Note that in this case, this function involves a repeated leaf container location). vtkPoints::InsertPoint() is invoked.
- `vtkIdType = obj.InsertNextPoint (double x[3])` - Insert a given point into the octree and return the point index. Note that InitPointInsertion() should have been called prior to this function. In addition, IsInsertedPoint() should have been called in advance to ensure that the given point has not been inserted unless point duplication is allowed (in this case, this function invovles a repeated leaf container location). vtkPoints::InsertNextPoint() is invoked.

31.114 vtkIncrementalPointLocator

31.114.1 Usage

Compared to a static point locator for pure location functionalities through some search structure established from a fixed set of points, an incremental point locator allows for, in addition, point insertion capabilities, with the search structure maintaining a dynamically increasing number of points. There are two incremental

point locators, i.e., vtkPointLocator and vtkIncrementalOctreePointLocator. As opposed to the uniform bin-based search structure (adopted in vtkPointLocator) with a fixed spatial resolution, an octree mechanism (employed in vtkIncrementalOctreePointlocator) resorts to a hierarchy of tree-like sub-division of the 3D data domain. Thus it enables data-aware multi- resolution and accordingly accelerated point location as well as point insertion, particularly when handling a radically imbalanced layout of points as not uncommon in datasets defined on adaptive meshes. In other words, vtkIncrementalOctreePointLocator is an octree-based accelerated implementation of all functionalities of vtkPointLocator.

To create an instance of class vtkIncrementalPointLocator, simply invoke its constructor as follows

```
obj = vtkIncrementalPointLocator
```

31.114.2 Methods

The class vtkIncrementalPointLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIncrementalPointLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIncrementalPointLocator = obj.NewInstance ()`
- `vtkIncrementalPointLocator = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Delete the search structure.
- `vtkIdType = obj.FindClosestInsertedPoint (double x[3])` - Given a point x assumed to be covered by the search structure, return the index of the closest point (already inserted to the search structure) regardless of the associated minimum squared distance relative to the squared insertion-tolerance distance. This method is used when performing incremental point insertion. Note -1 indicates that no point is found. InitPointInsertion() should have been called in advance.
- `int = obj.InitPointInsertion (vtkPoints newPts, double bounds[6])` - Initialize the point insertion process. newPts is an object, storing 3D point coordinates, to which incremental point insertion puts coordinates. It is created and provided by an external VTK class. Argument bounds represents the spatial bounding box, into which the points fall.
- `int = obj.InitPointInsertion (vtkPoints newPts, double bounds[6], vtkIdType estSize)` - Initialize the point insertion process. newPts is an object, storing 3D point coordinates, to which incremental point insertion puts coordinates. It is created and provided by an external VTK class. Argument bounds represents the spatial bounding box, into which the points fall.
- `vtkIdType = obj.IsInsertedPoint (double x, double y, double z)` - Determine whether or not a given point has been inserted. Return the id of the already inserted point if true, else return -1. InitPointInsertion() should have been called in advance.
- `vtkIdType = obj.IsInsertedPoint (double x[3])` - Determine whether or not a given point has been inserted. Return the id of the already inserted point if true, else return -1. InitPointInsertion() should have been called in advance.
- `obj.InsertPoint (vtkIdType ptId, double x[3])` - Insert a given point with a specified point index ptId. InitPointInsertion() should have been called prior to this function. Also, IsInsertedPoint() should have been called in advance to ensure that the given point has not been inserted unless point duplication is allowed.
- `vtkIdType = obj.InsertNextPoint (double x[3])` - Insert a given point and return the point index. InitPointInsertion() should have been called prior to this function. Also, IsInsertedPoint() should have been called in advance to ensure that the given point has not been inserted unless point duplication is allowed.

31.115 vtkInEdgeIterator

31.115.1 Usage

vtkInEdgeIterator iterates through all edges whose target is a particular vertex. Instantiate this class directly and call Initialize() to traverse the vertex of a graph. Alternately, use GetInEdges() on the graph to initialize the iterator. it-¿Next() returns a vtkInEdgeType structure, which contains Id, the edge's id, and Source, the edge's source vertex.

To create an instance of class vtkInEdgeIterator, simply invoke its constructor as follows

```
obj = vtkInEdgeIterator
```

31.115.2 Methods

The class vtkInEdgeIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInEdgeIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInEdgeIterator = obj.NewInstance ()`
- `vtkInEdgeIterator = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (vtkGraph g, vtkIdType v)` - Initialize the iterator with a graph and vertex.
- `vtkGraph = obj.GetGraph ()` - Get the graph and vertex associated with this iterator.
- `vtkIdType = obj.GetVertex ()` - Get the graph and vertex associated with this iterator.
- `vtkGraphEdge = obj.NextGraphEdge ()` - Just like Next(), but returns heavy-weight vtkGraphEdge object instead of the vtkEdgeType struct, for use with wrappers. The graph edge is owned by this iterator, and changes after each call to NextGraphEdge().
- `bool = obj.HasNext ()`

31.116 vtkInformationExecutivePortKey

31.116.1 Usage

vtkInformationExecutivePortKey is used to represent keys in vtkInformation for values that are vtkExecutive instances paired with port numbers.

To create an instance of class vtkInformationExecutivePortKey, simply invoke its constructor as follows

```
obj = vtkInformationExecutivePortKey
```

31.116.2 Methods

The class vtkInformationExecutivePortKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationExecutivePortKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkInformationExecutivePortKey = obj.NewInstance ()`
- `vtkInformationExecutivePortKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationExecutivePortKey = obj.(string name, string location)`
- `~vtkInformationExecutivePortKey = obj.()`
- `obj.Set (vtkInformation info, vtkExecutive , int )` - Get/Set the value associated with this key in the given information object.
- `vtkExecutive = obj.GetExecutive (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `int = obj.GetPort (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.
- `obj.Report (vtkInformation info, vtkGarbageCollector collector)` - Report a reference this key has in the given information object.

31.117 vtkInformationExecutivePortVectorKey

31.117.1 Usage

vtkInformationExecutivePortVectorKey is used to represent keys in vtkInformation for values that are vectors of vtkExecutive instances paired with port numbers.

To create an instance of class vtkInformationExecutivePortVectorKey, simply invoke its constructor as follows

```
obj = vtkInformationExecutivePortVectorKey
```

31.117.2 Methods

The class vtkInformationExecutivePortVectorKey has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInformationExecutivePortVectorKey class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInformationExecutivePortVectorKey = obj.NewInstance ()`
- `vtkInformationExecutivePortVectorKey = obj.SafeDownCast (vtkObject o)`
- `vtkInformationExecutivePortVectorKey = obj.(string name, string location)`
- `~vtkInformationExecutivePortVectorKey = obj.()`
- `obj.Append (vtkInformation info, vtkExecutive executive, int port)` - Get/Set the value associated with this key in the given information object.
- `obj.Remove (vtkInformation info, vtkExecutive executive, int port)` - Get/Set the value associated with this key in the given information object.

- `int = obj.Length (vtkInformation info)` - Get/Set the value associated with this key in the given information object.
- `obj.ShallowCopy (vtkInformation from, vtkInformation to)` - Copy the entry associated with this key from one information object to another. If there is no entry in the first information object for this key, the value is removed from the second.
- `obj.Remove (vtkInformation info)` - Remove this key from the given information object.
- `obj.Report (vtkInformation info, vtkGarbageCollector collector)` - Report a reference this key has in the given information object.

31.118 vtkInterpolatedVelocityField

31.118.1 Usage

vtkInterpolatedVelocityField acts as a continuous velocity field via cell interpolation on a vtkDataSet, NumberOfIndependentVariables = 4 (x,y,z,t) and NumberOfFunctions = 3 (u,v,w). As a concrete sub-class of vtkAbstractInterpolatedVelocityField, this class adopts two levels of cell caching for faster though less robust cell location than its sibling class vtkCellLocatorInterpolatedVelocityField. Level #0 begins with intra-cell caching. Specifically, if the previous cell is valid and the nex point is still within it, (vtkCell::EvaluatePosition() returns 1, coupled with the new parametric coordinates and weights), the function values are interpolated and vtkCell::EvaluatePosition() is invoked only. If it fails, level #1 follows by inter-cell location of the target cell (that contains the next point). By inter-cell, the previous cell gives an important clue / guess or serves as an immediate neighbor to aid in the location of the target cell (as is typically the case with integrating a streamline across cells) by means of vtkDataSet:: FindCell(). If this still fails, a global cell search is invoked via vtkDataSet::FindCell().

Regardless of inter-cell or global search, vtkPointLocator is employed as a crucial tool underlying the cell locator. The use of vtkPointLocator casues vtkInterpolatedVelocityField to return false target cells for datasets defined on complex grids.

To create an instance of class vtkInterpolatedVelocityField, simply invoke its constructor as follows

```
obj = vtkInterpolatedVelocityField
```

31.118.2 Methods

The class vtkInterpolatedVelocityField has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInterpolatedVelocityField class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInterpolatedVelocityField = obj.NewInstance ()`
- `vtkInterpolatedVelocityField = obj.SafeDownCast (vtkObject o)`
- `obj.AddDataSet (vtkDataSet dataset)` - Add a dataset used for the implicit function evaluation. If more than one dataset is added, the evaluation point is searched in all until a match is found. THIS FUNCTION DOES NOT CHANGE THE REFERENCE COUNT OF DATASET FOR THREAD SAFETY REASONS.
- `int = obj.FunctionValues (double x, double f)` - Evaluate the velocity field f at point (x, y, z).
- `obj.SetLastCellId (vtkIdType c, int dataindex)` - Set the cell id cached by the last evaluation within a specified dataset.
- `obj.SetLastCellId (vtkIdType c)`

31.119 vtkKdNode

31.119.1 Usage

To create an instance of class vtkKdNode, simply invoke its constructor as follows

```
obj = vtkKdNode
```

31.119.2 Methods

The class vtkKdNode has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkKdNode class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkKdNode = obj.NewInstance ()`
- `vtkKdNode = obj.SafeDownCast (vtkObject o)`
- `obj.SetDim (int )` - Set/Get the dimension along which this region is divided. (0 - x, 1 - y, 2 - z, 3 - leaf node (default)).
- `int = obj.GetDim ()` - Set/Get the dimension along which this region is divided. (0 - x, 1 - y, 2 - z, 3 - leaf node (default)).
- `double = obj.GetDivisionPosition ()` - Get the location of the division plane along the axis the region is divided. See also GetDim(). The result is undertermined if this node is not divided (a leaf node).
- `obj.SetNumberOfPoints (int )` - Set/Get the number of points contained in this region.
- `int = obj.GetNumberOfPoints ()` - Set/Get the number of points contained in this region.
- `obj.SetBounds (double x1, double x2, double y1, double y2, double z1, double z2)` - Set/Get the bounds of the spatial region represented by this node. Caller allocates storage for 6-vector in GetBounds.
- `obj.SetBounds (double b[6])` - Set/Get the bounds of the spatial region represented by this node. Caller allocates storage for 6-vector in GetBounds.
- `obj.GetBounds (double b) const` - Set/Get the bounds of the spatial region represented by this node. Caller allocates storage for 6-vector in GetBounds.
- `obj.SetDataBounds (double x1, double x2, double y1, double y2, double z1, double z2)` - Set/Get the bounds of the points contained in this spatial region. This may be smaller than the bounds of the region itself. Caller allocates storage for 6-vector in GetDataBounds.
- `obj.GetDataBounds (double b) const` - Set/Get the bounds of the points contained in this spatial region. This may be smaller than the bounds of the region itself. Caller allocates storage for 6-vector in GetDataBounds.
- `obj.SetDataBounds (float v)` - Given a pointer to NumberOfPoints points, set the DataBounds of this node to the bounds of these points.
- `double = obj.GetMinBounds ()` - Get a pointer to the 3 bound minima (xmin, ymin and zmin) or the 3 bound maxima (xmax, ymax, zmax). Don't free this pointer.
- `double = obj.GetMaxBounds ()` - Set the xmin, ymin and zmin value of the bounds of this region

- `obj.SetMinBounds (double mb)` - Set the xmin, ymin and zmin value of the bounds of this region
- `obj.SetMaxBounds (double mb)` - Set the xmax, ymax and zmax value of the bounds of this region
- `double = obj.GetMinDataBounds ()` - Get a pointer to the 3 data bound minima (xmin, ymin and zmin) or the 3 data bound maxima (xmax, ymax, zmax). Don't free this pointer.
- `double = obj.GetMaxDataBounds ()` - Set the xmin, ymin and zmin value of the bounds of this data within this region
- `obj.SetMinDataBounds (double mb)` - Set the xmin, ymin and zmin value of the bounds of this data within this region
- `obj.SetMaxDataBounds (double mb)` - Set the xmax, ymax and zmax value of the bounds of this data within this region
- `obj.SetID (int )` - Set/Get the ID associated with the region described by this node. If this is not a leaf node, this value should be -1.
- `int = obj.GetID ()` - Set/Get the ID associated with the region described by this node. If this is not a leaf node, this value should be -1.
- `int = obj.GetMinID ()` - If this node is not a leaf node, there are leaf nodes below it whose regions represent a partitioning of this region. The IDs of these leaf nodes form a contigous set. Set/Get the range of the IDs of the leaf nodes below this node. If this is already a leaf node, these values should be the same as the ID.
- `int = obj.GetMaxID ()` - If this node is not a leaf node, there are leaf nodes below it whose regions represent a partitioning of this region. The IDs of these leaf nodes form a contigous set. Set/Get the range of the IDs of the leaf nodes below this node. If this is already a leaf node, these values should be the same as the ID.
- `obj.SetMinID (int )` - If this node is not a leaf node, there are leaf nodes below it whose regions represent a partitioning of this region. The IDs of these leaf nodes form a contigous set. Set/Get the range of the IDs of the leaf nodes below this node. If this is already a leaf node, these values should be the same as the ID.
- `obj.SetMaxID (int )` - If this node is not a leaf node, there are leaf nodes below it whose regions represent a partitioning of this region. The IDs of these leaf nodes form a contigous set. Set/Get the range of the IDs of the leaf nodes below this node. If this is already a leaf node, these values should be the same as the ID.
- `obj.AddChildNodes (vtkKdNode left, vtkKdNode right)` - Add the left and right children.
- `obj.DeleteChildNodes ()` - Delete the left and right children.
- `vtkKdNode = obj.GetLeft ()` - Set/Get a pointer to the left child of this node.
- `obj.SetLeft (vtkKdNode left)` - Set/Get a pointer to the left child of this node.
- `vtkKdNode = obj.GetRight ()` - Set/Get a pointer to the right child of this node.
- `obj.SetRight (vtkKdNode right)` - Set/Get a pointer to the right child of this node.
- `vtkKdNode = obj.GetUp ()` - Set/Get a pointer to the parent of this node.
- `obj.SetUp (vtkKdNode up)` - Set/Get a pointer to the parent of this node.
- `int = obj.IntersectsBox (double x1, double x2, double y1, double y2, double z1, double z2, int useD` - Return 1 if this spatial region intersects the axis-aligned box given by the bounds passed in. Use the possibly smaller bounds of the points within the region if useDataBounds is non-zero.

- `int = obj.IntersectsSphere2 (double x, double y, double z, double rSquared, int useDataBounds)` - Return 1 if this spatial region intersects a sphere described by it's center and the square of it's radius. Use the possibly smaller bounds of the points within the region if useDataBounds is non-zero.
- `int = obj.IntersectsRegion (vtkPlanesIntersection pi, int useDataBounds)` - A vtkPlanesIntersection object represents a convex 3D region bounded by planes, and it is capable of computing intersections of boxes with itself. Return 1 if this spatial region intersects the spatial region described by the vtkPlanesIntersection object. Use the possibly smaller bounds of the points within the region if useDataBounds is non-zero.
- `int = obj.IntersectsCell (vtkCell cell, int useDataBounds, int cellRegion, double cellBoundsNULL)` - Return 1 if the cell specified intersects this region. If you already know the ID of the region containing the cell's centroid, provide that as an argument. If you already know the bounds of the cell, provide that as well, in the form of xmin,xmax,ymin,ymax,zmin, zmax. Either of these may speed the calculation. Use the possibly smaller bounds of the points within the region if useDataBounds is non-zero.
- `int = obj.ContainsBox (double x1, double x2, double y1, double y2, double z1, double z2, int useDat` - Return 1 if this spatial region entirely contains a box specified by it's bounds. Use the possibly smaller bounds of the points within the region if useDataBounds is non-zero.
- `int = obj.ContainsPoint (double x, double y, double z, int useDataBounds)` - Return 1 if this spatial region entirely contains the given point. Use the possibly smaller bounds of the points within the region if useDataBounds is non-zero.
- `double = obj.GetDistance2ToBoundary (double x, double y, double z, int useDataBounds)` - Calculate the distance squared from any point to the boundary of this region. Use the boundary of the points within the region if useDataBounds is non-zero.
- `double = obj.GetDistance2ToBoundary (double x, double y, double z, double boundaryPt, int useDataBo` - Calculate the distance squared from any point to the boundary of this region. Use the boundary of the points within the region if useDataBounds is non-zero. Set boundaryPt to the point on the boundary.
- `double = obj.GetDistance2ToInnerBoundary (double x, double y, double z)` - Calculate the distance from the specified point (which is required to be inside this spatial region) to an interior boundary. An interior boundary is one that is not also an boundary of the entire space partitioned by the tree of vtkKdNode's.
- `obj.PrintNode (int depth)` - For debugging purposes, print out this node.
- `obj.PrintVerboseNode (int depth)` - For debugging purposes, print out this node.

31.120 vtkKdTree

31.120.1 Usage

Given one or more vtkDataSets, create a load balancing k-d tree decomposition of the points at the center of the cells. Or, create a k-d tree point locator from a list of points.

This class can also generate a PolyData representation of the boundaries of the spatial regions in the decomposition.

It can sort the regions with respect to a viewing direction, and it can decompose a list of regions into subsets, each of which represent a convex spatial region (since many algorithms require a convex region).

If the points were derived from cells, vtkKdTree can create a list of cell Ids for each region for each data set. Two lists are available - all cells with centroid in the region, and all cells that intersect the region but whose centroid lies in another region.

For the purpose of removing duplicate points quickly from large data sets, or for finding nearby points, we added another mode for building the locator. BuildLocatorFromPoints will build a k-d tree from one or more vtkPoints objects. This can be followed by BuildMapForDuplicatePoints which returns a mapping from the original ids to a subset of the ids that is unique within a supplied tolerance, or you can use FindPoint and FindClosestPoint to locate points in the original set that the tree was built from.

To create an instance of class vtkKdTree, simply invoke its constructor as follows

```
obj = vtkKdTree
```

31.120.2 Methods

The class vtkKdTree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkKdTree class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkKdTree = obj.NewInstance ()`
- `vtkKdTree = obj.SafeDownCast (vtkObject o)`
- `obj.TimingOn ()` - Turn on timing of the k-d tree build
- `obj.TimingOff ()` - Turn on timing of the k-d tree build
- `obj.SetTiming (int )` - Turn on timing of the k-d tree build
- `int = obj.GetTiming ()` - Turn on timing of the k-d tree build
- `obj.SetMinCells (int )` - Minimum number of cells per spatial region. Default is 100.
- `int = obj.GetMinCells ()` - Minimum number of cells per spatial region. Default is 100.
- `int = obj.GetNumberOfRegionsOrLess ()`
- `obj.SetNumberOfRegionsOrLess (int )`
- `int = obj.GetNumberOfRegionsOrMore ()`
- `obj.SetNumberOfRegionsOrMore (int )`
- `double = obj.GetFudgeFactor ()`
- `obj.SetFudgeFactor (double )`
- `vtkBSPCuts = obj.GetCuts ()`
- `obj.SetCuts (vtkBSPCuts cuts)`
- `obj.OmitXPartitioning ()` - Omit partitions along the X axis, yielding shafts in the X direction
- `obj.OmitYPartitioning ()` - Omit partitions along the Y axis, yielding shafts in the Y direction
- `obj.OmitZPartitioning ()` - Omit partitions along the Z axis, yielding shafts in the Z direction
- `obj.OmitXYPartitioning ()` - Omit partitions along the X and Y axes, yielding slabs along Z
- `obj.OmitYZPartitioning ()` - Omit partitions along the Y and Z axes, yielding slabs along X
- `obj.OmitZXPartitioning ()` - Omit partitions along the Z and X axes, yielding slabs along Y

- `obj.OmitNoPartitioning ()` - Partition along all three axes - this is the default
- `obj.SetDataSet (vtkDataSet set)` - Clear out all data sets and replace with single data set. For backward compatibility with superclass.
- `obj.AddDataSet (vtkDataSet set)` - This class can compute a spatial decomposition based on the cells in a list of one or more input data sets. Add them one at a time with this method.
- `obj.RemoveDataSet (int index)` - Remove the given data set.
- `obj.RemoveDataSet (vtkDataSet set)` - Remove the given data set.
- `obj.RemoveAllDataSets ()` - Remove the given data set.
- `int = obj.GetNumberOfDataSets ()` - Get the number of data sets included in spatial paritioning
- `vtkDataSet = obj.GetDataSet (int n)` - Return the n'th data set.
- `vtkDataSet = obj.GetDataSet ()` - Return a collection of all the data sets.
- `vtkDataSetCollection = obj.GetDataSets ()` - Return a collection of all the data sets.
- `int = obj.GetDataSetIndex (vtkDataSet set)` - Return the index of the given data set. Returns -1 if that data set does not exist.
- `obj.GetBounds (double bounds)` - Get the spatial bounds of the entire k-d tree space. Sets bounds array to xmin, xmax, ymin, ymax, zmin, zmax.
- `obj.SetNewBounds (double bounds)`
- `int = obj.GetNumberOfRegions ()` - The number of leaf nodes of the tree, the spatial regions
- `obj.GetRegionBounds (int regionID, double bounds[6])` - Get the spatial bounds of k-d tree region
- `obj.GetRegionDataBounds (int regionID, double bounds[6])` - Get the bounds of the data within the k-d tree region
- `obj.PrintTree ()` - Print out nodes of kd tree
- `obj.PrintVerboseTree ()` - Print out nodes of kd tree
- `obj.PrintRegion (int id)` - Print out leaf node data for given id
- `obj.CreateCellLists (int dataSetIndex, int regionReqList, int reqListSize)`
- `obj.CreateCellLists (vtkDataSet set, int regionReqList, int reqListSize)`
- `obj.CreateCellLists (int regionReqList, int listSize)`
- `obj.CreateCellLists ()`
- `obj.SetIncludeRegionBoundaryCells (int )` - If IncludeRegionBoundaryCells is ON, CreateCellLists() will also create a list of cells which intersect a given region, but are not assigned to the region. These lists are obtained with GetBoundaryCellList(). Default is OFF.
- `int = obj.GetIncludeRegionBoundaryCells ()` - If IncludeRegionBoundaryCells is ON, CreateCellLists() will also create a list of cells which intersect a given region, but are not assigned to the region. These lists are obtained with GetBoundaryCellList(). Default is OFF.
- `obj.IncludeRegionBoundaryCellsOn ()` - If IncludeRegionBoundaryCells is ON, CreateCellLists() will also create a list of cells which intersect a given region, but are not assigned to the region. These lists are obtained with GetBoundaryCellList(). Default is OFF.

- `obj.IncludeRegionBoundaryCellsOff ()` - If IncludeRegionBoundaryCells is ON, CreateCellLists() will also create a list of cells which intersect a given region, but are not assigned to the region. These lists are obtained with GetBoundaryCellList(). Default is OFF.

- `obj.DeleteCellLists ()` - Free the memory used by the cell lists.

- `vtkIdList = obj.GetCellList (int regionID)` - Get the cell list for a region. This returns a pointer to vtkKdTree's memory, so don't free it.

- `vtkIdList = obj.GetBoundaryCellList (int regionID)` - The cell list obtained with GetCellList is the list of all cells such that their centroid is contained in the spatial region. It may also be desirable to get a list of all cells intersecting a spatial region, but with centroid in some other region. This is that list. This list is computed in CreateCellLists() if and only if IncludeRegionBoundaryCells is ON. This returns a pointer to KdTree's memory, so don't free it.

- `vtkIdType = obj.GetCellLists (vtkIntArray regions, int set, vtkIdList inRegionCells, vtkIdList onBo` - For a list of regions, get two cell lists. The first lists the IDs all cells whose centroids lie in one of the regions. The second lists the IDs of all cells that intersect the regions, but whose centroid lies in a region not on the list.

  The total number of cell IDs written to both lists is returned. Either list pointer passed in can be NULL, and it will be ignored. If there are multiple data sets, you must specify which data set you wish cell IDs for.

  The caller should delete these two lists when done. This method uses the cell lists created in CreateCellLists(). If the cell list for any of the requested regions does not exist, then this method will call CreateCellLists() to create cell lists for *every* region of the k-d tree. You must remember to DeleteCellLists() when done with all calls to this method, as cell lists can require a great deal of memory.

- `vtkIdType = obj.GetCellLists (vtkIntArray regions, vtkDataSet set, vtkIdList inRegionCells, vtkIdLi` - For a list of regions, get two cell lists. The first lists the IDs all cells whose centroids lie in one of the regions. The second lists the IDs of all cells that intersect the regions, but whose centroid lies in a region not on the list.

  The total number of cell IDs written to both lists is returned. Either list pointer passed in can be NULL, and it will be ignored. If there are multiple data sets, you must specify which data set you wish cell IDs for.

  The caller should delete these two lists when done. This method uses the cell lists created in CreateCellLists(). If the cell list for any of the requested regions does not exist, then this method will call CreateCellLists() to create cell lists for *every* region of the k-d tree. You must remember to DeleteCellLists() when done with all calls to this method, as cell lists can require a great deal of memory.

- `vtkIdType = obj.GetCellLists (vtkIntArray regions, vtkIdList inRegionCells, vtkIdList onBoundaryCel` - For a list of regions, get two cell lists. The first lists the IDs all cells whose centroids lie in one of the regions. The second lists the IDs of all cells that intersect the regions, but whose centroid lies in a region not on the list.

  The total number of cell IDs written to both lists is returned. Either list pointer passed in can be NULL, and it will be ignored. If there are multiple data sets, you must specify which data set you wish cell IDs for.

  The caller should delete these two lists when done. This method uses the cell lists created in CreateCellLists(). If the cell list for any of the requested regions does not exist, then this method will call CreateCellLists() to create cell lists for *every* region of the k-d tree. You must remember to DeleteCellLists() when done with all calls to this method, as cell lists can require a great deal of memory.

- `int = obj.GetRegionContainingCell (vtkDataSet set, vtkIdType cellID)` - Get the id of the region containing the cell centroid. If no DataSet is specified, assume DataSet 0. If you need the region ID for every cell, use AllGetRegionContainingCell instead. It is more efficient.

- `int = obj.GetRegionContainingCell (int set, vtkIdType cellID)` - Get the id of the region containing the cell centroid. If no DataSet is specified, assume DataSet 0. If you need the region ID for every cell, use AllGetRegionContainingCell instead. It is more efficient.

- `int = obj.GetRegionContainingCell (vtkIdType cellID)` - Get the id of the region containing the cell centroid. If no DataSet is specified, assume DataSet 0. If you need the region ID for every cell, use AllGetRegionContainingCell instead. It is more efficient.

- `int = obj.GetRegionContainingPoint (double x, double y, double z)` - Get the id of the region containing the specified location.

- `obj.BuildLocator ()` - Create the k-d tree decomposition of the cells of the data set or data sets. Cells are assigned to k-d tree spatial regions based on the location of their centroids.

- `int = obj.DepthOrderAllRegions (double dop, vtkIntArray orderedList)` - DO NOT CALL. Depricated in VTK 5.2. Use ViewOrderAllRegionsInDirection or ViewOrderAllRegionsFromPosition.

- `int = obj.DepthOrderRegions (vtkIntArray regionIds, double dop, vtkIntArray orderedList)` - DO NOT CALL. Depricated in VTK 5.2. Use ViewOrderRegionsInDirection or ViewOrderRegionsFromPosition.

- `int = obj.ViewOrderAllRegionsInDirection (double directionOfProjection[3], vtkIntArray orderedList)` - Given a direction of projection (typically obtained with vtkCamera::GetDirectionOfProjection()), this method, creates a list of the k-d tree region IDs in order from front to back with respect to that direction. The number of ordered regions is returned. Use this method to view order regions for cameras that use parallel projection.

- `int = obj.ViewOrderRegionsInDirection (vtkIntArray regionIds, double directionOfProjection[3], vtkI` - Given a direction of projection and a list of k-d tree region IDs, this method, creates a list of the k-d tree region IDs in order from front to back with respect to that direction. The number of ordered regions is returned. Use this method to view order regions for cameras that use parallel projection.

- `int = obj.ViewOrderAllRegionsFromPosition (double directionOfProjection[3], vtkIntArray orderedList` - Given a camera position (typically obtained with vtkCamera::GetPosition()), this method, creates a list of the k-d tree region IDs in order from front to back with respect to that direction. The number of ordered regions is returned. Use this method to view order regions for cameras that use perspective projection.

- `int = obj.ViewOrderRegionsFromPosition (vtkIntArray regionIds, double directionOfProjection[3], vtk` - Given a camera position and a list of k-d tree region IDs, this method, creates a list of the k-d tree region IDs in order from front to back with respect to that direction. The number of ordered regions is returned. Use this method to view order regions for cameras that use perspective projection.

- `obj.BuildLocatorFromPoints (vtkPointSet pointset)` - This is a special purpose locator that builds a k-d tree to find duplicate and near-by points. It builds the tree from one or more vtkPoints objects instead of from the cells of a vtkDataSet. This build would normally be followed by BuildMapForDuplicatePoints, FindPoint, or FindClosestPoint. Since this will build a normal k-d tree, all the region intersection queries will still work, as will most other calls except those that have "Cell" in the name.

  This method works most efficiently when the point arrays are float arrays.

- `obj.BuildLocatorFromPoints (vtkPoints ptArray)` - This is a special purpose locator that builds a k-d tree to find duplicate and near-by points. It builds the tree from one or more vtkPoints objects

instead of from the cells of a vtkDataSet. This build would normally be followed by BuildMapForDuplicatePoints, FindPoint, or FindClosestPoint. Since this will build a normal k-d tree, all the region intersection queries will still work, as will most other calls except those that have "Cell" in the name.

This method works most efficiently when the point arrays are float arrays.

- `vtkIdTypeArray = obj.BuildMapForDuplicatePoints (float tolerance)` - This call returns a mapping from the original point IDs supplied to BuildLocatorFromPoints to a subset of those IDs that is unique within the specified tolerance. If points 2, 5, and 12 are the same, then IdMap[2] = IdMap[5] = IdMap[12] = 2 (or 5 or 12).

  "original point IDs" - For point IDs we start at 0 for the first point in the first vtkPoints object, and increase by 1 for subsequent points and subsequent vtkPoints objects.

  You must have called BuildLocatorFromPoints() before calling this. You are responsible for deleting the returned array.

- `vtkIdType = obj.FindPoint (double x)` - Find the Id of the point that was previously supplied to BuildLocatorFromPoints(). Returns -1 if the point was not in the original array.

- `vtkIdType = obj.FindPoint (double x, double y, double z)` - Find the Id of the point that was previously supplied to BuildLocatorFromPoints(). Returns -1 if the point was not in the original array.

- `obj.FindPointsWithinRadius (double R, double x[3], vtkIdList result)` - Find all points within a specified radius R of position x. The result is not sorted in any specific manner. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.

- `obj.FindClosestNPoints (int N, double x[3], vtkIdList result)` - Find the closest N points to a position. This returns the closest N points to a position. A faster method could be created that returned N close points to a position, but necessarily the exact N closest. The returned points are sorted from closest to farthest. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.

- `vtkIdTypeArray = obj.GetPointsInRegion (int regionId)` - Get a list of the original IDs of all points in a region. You must have called BuildLocatorFromPoints before calling this.

- `obj.FreeSearchStructure ()` - Delete the k-d tree data structure. Also delete any cell lists that were computed with CreateCellLists().

- `obj.GenerateRepresentation (int level, vtkPolyData pd)` - Create a polydata representation of the boundaries of the k-d tree regions. If level equals GetLevel(), the leaf nodes are represented.

- `obj.GenerateRepresentation (int regionList, int len, vtkPolyData pd)` - Generate a polygonal representation of a list of regions. Only leaf nodes have region IDs, so these will be leaf nodes.

- `obj.GenerateRepresentationUsingDataBoundsOn ()` - The polydata representation of the k-d tree shows the boundaries of the k-d tree decomposition spatial regions. The data inside the regions may not occupy the entire space. To draw just the bounds of the data in the regions, set this variable ON.

- `obj.GenerateRepresentationUsingDataBoundsOff ()` - The polydata representation of the k-d tree shows the boundaries of the k-d tree decomposition spatial regions. The data inside the regions may not occupy the entire space. To draw just the bounds of the data in the regions, set this variable ON.

- `obj.SetGenerateRepresentationUsingDataBounds (int )` - The polydata representation of the k-d tree shows the boundaries of the k-d tree decomposition spatial regions. The data inside the regions may not occupy the entire space. To draw just the bounds of the data in the regions, set this variable ON.

- `int = obj.GetGenerateRepresentationUsingDataBounds ()` - The polydata representation of the k-d tree shows the boundaries of the k-d tree decomposition spatial regions. The data inside the regions may not occupy the entire space. To draw just the bounds of the data in the regions, set this variable ON.

- `int = obj.NewGeometry ()` - Return 1 if the geometry of the input data sets has changed since the last time the k-d tree was built.
- `obj.InvalidateGeometry ()` - Forget about the last geometry used. The next call to NewGeometry will return 1. A new k-d tree will be built the next time BuildLocator is called.
- `obj.FindPointsInArea (double area, vtkIdTypeArray ids, bool clearArraytrue)` - Fill ids with points found in area. The area is a 6-tuple containing (xmin, xmax, ymin, ymax, zmin, zmax). This method will clear the array by default. To append ids to an array, set clearArray to false.

31.121 vtkKdTreePointLocator

31.121.1 Usage

vtkKdTreePointLocator is a wrapper class that derives from vtkAbstractPointLocator and calls the search functions in vtkKdTree.

To create an instance of class vtkKdTreePointLocator, simply invoke its constructor as follows

```
obj = vtkKdTreePointLocator
```

31.121.2 Methods

The class vtkKdTreePointLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkKdTreePointLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkKdTreePointLocator = obj.NewInstance ()`
- `vtkKdTreePointLocator = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.FindClosestPoint (double x[3])` - Given a position x, return the id of the point closest to it. Alternative method requires separate x-y-z values. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.FindClosestNPoints (int N, double x[3], vtkIdList result)` - Find the closest N points to a position. This returns the closest N points to a position. A faster method could be created that returned N close points to a position, but necessarily the exact N closest. The returned points are sorted from closest to farthest. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.FindPointsWithinRadius (double R, double x[3], vtkIdList result)` - Find all points within a specified radius R of position x. The result is not sorted in any specific manner. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `obj.FreeSearchStructure ()` - See vtkLocator interface documentation. These methods are not thread safe.
- `obj.BuildLocator ()` - See vtkLocator interface documentation. These methods are not thread safe.
- `obj.GenerateRepresentation (int level, vtkPolyData pd)` - See vtkLocator interface documentation. These methods are not thread safe.

31.122 vtkKochanekSpline

31.122.1 Usage

Implements the Kochenek interpolating spline described in: Kochanek, D., Bartels, R., "Interpolating Splines with Local Tension, Continuity, and Bias Control," Computer Graphics, vol. 18, no. 3, pp. 33-41, July 1984. These splines give the user more control over the shape of the curve than the cardinal splines implemented in vtkCardinalSpline. Three parameters can be specified. All have a range from -1 to 1.

Tension controls how sharply the curve bends at an input point. A value of -1 produces more slack in the curve. A value of 1 tightens the curve.

Continuity controls the continuity of the first derivative at input points.

Bias controls the direction of the curve at it passes through an input point. A value of -1 undershoots the point while a value of 1 overshoots the point.

These three parameters give the user broad control over the shape of the interpolating spline. The original Kochanek paper describes the effects nicely and is recommended reading.

To create an instance of class vtkKochanekSpline, simply invoke its constructor as follows

```
obj = vtkKochanekSpline
```

31.122.2 Methods

The class vtkKochanekSpline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkKochanekSpline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkKochanekSpline = obj.NewInstance ()`
- `vtkKochanekSpline = obj.SafeDownCast (vtkObject o)`
- `obj.Compute ()` - Compute Kochanek Spline coefficients.
- `double = obj.Evaluate (double t)` - Evaluate a 1D Kochanek spline.
- `obj.SetDefaultBias (double )` - Set the bias for all points. Default is 0.
- `double = obj.GetDefaultBias ()` - Set the bias for all points. Default is 0.
- `obj.SetDefaultTension (double )` - Set the tension for all points. Default is 0.
- `double = obj.GetDefaultTension ()` - Set the tension for all points. Default is 0.
- `obj.SetDefaultContinuity (double )` - Set the continuity for all points. Default is 0.
- `double = obj.GetDefaultContinuity ()` - Set the continuity for all points. Default is 0.
- `obj.DeepCopy (vtkSpline s)` - Deep copy of cardinal spline data.

31.123 vtkLine

31.123.1 Usage

vtkLine is a concrete implementation of vtkCell to represent a 1D line.

To create an instance of class vtkLine, simply invoke its constructor as follows

```
obj = vtkLine
```

31.123.2 Methods

The class vtkLine has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLine class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLine = obj.NewInstance ()`
- `vtkLine = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArr` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this line using scalar value provided. Like contouring, except that it cuts the line to produce other lines.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the triangle in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[2])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[2])`

31.124 vtkLocator

31.124.1 Usage

vtkLocator is an abstract base class for spatial search objects, or locators. The principle behind locators is that they divide 3-space into small pieces (or "buckets") that can be quickly found in response to queries like point location, line intersection, or object-object intersection.

The purpose of this base class is to provide ivars and methods shared by all locators. The GenerateRepresentation() is one such interesting method. This method works in conjunction with vtkLocatorFilter to create polygonal representations for the locator. For example, if the locator is an OBB tree (i.e., vtkOBBTree.h), then the representation is a set of one or more oriented bounding boxes, depending upon the specified level.

Locators typically work as follows. One or more "entities", such as points or cells, are inserted into the tree. These entities are associated with one or more buckets. Then, when performing geometric operations, the operations are performed first on the buckets, and then if the operation tests positive, then on the entities in the bucket. For example, during collision tests, the locators are collided first to identify intersecting buckets. If an intersection is found, more expensive operations are then carried out on the entities in the bucket.

To obtain good performance, locators are often organized in a tree structure. In such a structure, there are frequently multiple "levels" corresponding to different nodes in the tree. So the word level (in the context of the locator) can be used to specify a particular representation in the tree. For example, in an octree (which is a tree with 8 children), level 0 is the bounding box, or root octant, and level 1 consists of its eight children.

To create an instance of class vtkLocator, simply invoke its constructor as follows

```
obj = vtkLocator
```

31.124.2 Methods

The class vtkLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLocator = obj.NewInstance ()`
- `vtkLocator = obj.SafeDownCast (vtkObject o)`
- `obj.SetDataSet (vtkDataSet )` - Build the locator from the points/cells defining this dataset.
- `vtkDataSet = obj.GetDataSet ()` - Build the locator from the points/cells defining this dataset.
- `obj.SetMaxLevel (int )` - Set the maximum allowable level for the tree. If the Automatic ivar is off, this will be the target depth of the locator. Initial value is 8.
- `int = obj.GetMaxLevelMinValue ()` - Set the maximum allowable level for the tree. If the Automatic ivar is off, this will be the target depth of the locator. Initial value is 8.
- `int = obj.GetMaxLevelMaxValue ()` - Set the maximum allowable level for the tree. If the Automatic ivar is off, this will be the target depth of the locator. Initial value is 8.
- `int = obj.GetMaxLevel ()` - Set the maximum allowable level for the tree. If the Automatic ivar is off, this will be the target depth of the locator. Initial value is 8.
- `int = obj.GetLevel ()` - Get the level of the locator (determined automatically if Automatic is true). The value of this ivar may change each time the locator is built. Initial value is 8.
- `obj.SetAutomatic (int )` - Boolean controls whether locator depth/resolution of locator is computed automatically from average number of entities in bucket. If not set, there will be an explicit method to control the construction of the locator (found in the subclass).
- `int = obj.GetAutomatic ()` - Boolean controls whether locator depth/resolution of locator is computed automatically from average number of entities in bucket. If not set, there will be an explicit method to control the construction of the locator (found in the subclass).
- `obj.AutomaticOn ()` - Boolean controls whether locator depth/resolution of locator is computed automatically from average number of entities in bucket. If not set, there will be an explicit method to control the construction of the locator (found in the subclass).

- `obj.AutomaticOff ()` - Boolean controls whether locator depth/resolution of locator is computed automatically from average number of entities in bucket. If not set, there will be an explicit method to control the construction of the locator (found in the subclass).
- `obj.SetTolerance (double )` - Specify absolute tolerance (in world coordinates) for performing geometric operations.
- `double = obj.GetToleranceMinValue ()` - Specify absolute tolerance (in world coordinates) for performing geometric operations.
- `double = obj.GetToleranceMaxValue ()` - Specify absolute tolerance (in world coordinates) for performing geometric operations.
- `double = obj.GetTolerance ()` - Specify absolute tolerance (in world coordinates) for performing geometric operations.
- `obj.Update ()` - Cause the locator to rebuild itself if it or its input dataset has changed.
- `obj.Initialize ()` - Initialize locator. Frees memory and resets object as appropriate.
- `obj.BuildLocator ()` - Build the locator from the input dataset.
- `obj.FreeSearchStructure ()` - Free the memory required for the spatial data structure.
- `obj.GenerateRepresentation (int level, vtkPolyData pd)` - Method to build a representation at a particular level. Note that the method GetLevel() returns the maximum number of levels available for the tree. You must provide a vtkPolyData object into which to place the data.
- `long = obj.GetBuildTime ()` - Return the time of the last data structure build.
- `obj.Register (vtkObjectBase o)` - Handle the PointSet ¡-¿ Locator loop.
- `obj.UnRegister (vtkObjectBase o)` - Handle the PointSet ¡-¿ Locator loop.

31.125 vtkMapper2D

31.125.1 Usage

vtkMapper2D is an abstract class which defines the interface for objects which render two dimensional actors (vtkActor2D).

To create an instance of class vtkMapper2D, simply invoke its constructor as follows

```
obj = vtkMapper2D
```

31.125.2 Methods

The class vtkMapper2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMapper2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMapper2D = obj.NewInstance ()`
- `vtkMapper2D = obj.SafeDownCast (vtkObject o)`
- `obj.RenderOverlay (vtkViewport , vtkActor2D )`

- `obj.RenderOpaqueGeometry (vtkViewport , vtkActor2D )`
- `obj.RenderTranslucentPolygonalGeometry (vtkViewport , vtkActor2D )`
- `int = obj.HasTranslucentPolygonalGeometry ()`

31.126 vtkMergePoints

31.126.1 Usage

vtkMergePoints is a locator object to quickly locate points in 3D. The primary difference between vtkMergePoints and its superclass vtkPointLocator is that vtkMergePoints merges precisely coincident points and is therefore much faster.

To create an instance of class vtkMergePoints, simply invoke its constructor as follows

```
obj = vtkMergePoints
```

31.126.2 Methods

The class vtkMergePoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMergePoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMergePoints = obj.NewInstance ()`
- `vtkMergePoints = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.IsInsertedPoint (double x[3])` - Determine whether point given by x[3] has been inserted into points list. Return id of previously inserted point if this is true, otherwise return -1.
- `vtkIdType = obj.IsInsertedPoint (double x, double y, double z)` - Determine whether point given by x[3] has been inserted into points list. Return id of previously inserted point if this is true, otherwise return -1.

31.127 vtkModifiedBSPTree

31.127.1 Usage

vtkModifiedBSPTree creates an evenly balanced BSP tree using a top down implementation. Axis aligned split planes are found which evenly divide cells into two buckets. Generally a split plane will intersect some cells and these are usually stored in both child nodes of the current parent. (Or split into separate cells which we cannot consider in this case). Storing cells in multiple buckets creates problems associated with multiple tests against rays and increases the required storage as complex meshes will have many cells straddling a split plane (and further splits may cause multiple copies of these).

During a discussion with Arno Formella in 1998 he suggested using a third child node to store objects which straddle split planes. I've not seen this published (Yes! - see below), but thought it worth trying. This implementation of the BSP tree creates a third child node for storing cells lying across split planes, the third cell may overlap the other two, but the two 'proper' nodes otherwise conform to usual BSP rules.

The advantage of this implementation is cells only ever lie in one node and mailbox testing is avoided. All BBoxes are axis aligned and a ray cast uses an efficient search strategy based on near/far nodes and rejects all BBoxes using simple tests.

For fast raytracing, 6 copies of cell lists are stored in each leaf node each list is in axis sorted order +/- x,y,z and cells are always tested in the direction of the ray dominant axis. Once an intersection is found any cell or BBox with a closest point further than the I-point can be instantly rejected and raytracing stops as soon as no nodes can be closer than the current best intersection point.

The addition of the 'middle' node upsets the optimal balance of the tree, but is a minor overhead during the raytrace. Each child node is contracted such that it tightly fits all cells inside it, enabling further ray/box rejections.

This class is intented for persons requiring many ray tests and is optimized for this purpose. As no cell ever lies in more than one leaf node, and parent nodes do not maintain cell lists, the memory overhead of the sorted cell lists is 6*num_cells*4 for 6 lists of ints, each num_cells in length. The memory requirement of the nodes themselves is usually of minor significance.

Subdividision is controlled by MaxCellsPerNode - any node with more than this number will be subdivided providing a good split plane can be found and the max depth is not exceeded.

The average cells per leaf will usually be around half the MaxCellsPerNode, though the middle node is usually sparsely populated and lowers the average slightly. The middle node will not be created when not needed. Subdividing down to very small cells per node is not generally suggested as then the 6 stored cell lists are effectively redundant.

Values of MaxcellsPerNode of around 16-¿128 depending on dataset size will usually give good results.

Cells are only sorted into 6 lists once - before tree creation, each node segments the lists and passes them down to the new child nodes whilst maintaining sorted order. This makes for an efficient subdivision strategy.

NB. The following reference has been sent to me @Articleformella-1995-ray, author = "Arno Formella and Christian Gill", title = "Ray Tracing: A Quantitative Analysis and a New Practical Algorithm", journal = "The Visual Computer", year = "1995", month = dec, pages = "465–476", volume = "11", number = "9", publisher = "Springer", keywords = "ray tracing, space subdivision, plane traversal, octree, clustering, benchmark scenes", annote = "We present a new method to accelerate the process of finding nearest ray–object intersections in ray tracing. The algorithm consumes an amount of memory more or less linear in the number of objects. The basic ideas can be characterized with a modified BSP–tree and plane traversal. Plane traversal is a fast linear time algorithm to find the closest intersection point in a list of bounding volumes hit by a ray. We use plane traversal at every node of the high outdegree BSP–tree. Our implementation is competitive to fast ray tracing programs. We present a benchmark suite which allows for an extensive comparison of ray tracing algorithms.",

.SECTION Thanks John Biddiscombe for developing and contributing this class

.SECTION ToDo ———– Implement intersection heap for testing rays against transparent objects

.SECTION Style ———– This class is currently maintained by J. Biddiscombe who has specially requested that the code style not be modified to the kitware standard. Please respect the contribution of this class by keeping the style as close as possible to the author's original.

To create an instance of class vtkModifiedBSPTree, simply invoke its constructor as follows

```
obj = vtkModifiedBSPTree
```

31.127.2 Methods

The class vtkModifiedBSPTree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkModifiedBSPTree class.

- `string = obj.GetClassName ()` - Standard Type-Macro
- `int = obj.IsA (string name)` - Standard Type-Macro
- `vtkModifiedBSPTree = obj.NewInstance ()` - Standard Type-Macro
- `vtkModifiedBSPTree = obj.SafeDownCast (vtkObject o)` - Standard Type-Macro
- `obj.FreeSearchStructure ()` - Free tree memory

- `obj.BuildLocator ()` - Build Tree

31.128 vtkMultiBlockDataSet

31.128.1 Usage

vtkMultiBlockDataSet is a vtkCompositeDataSet that stores a hierarchy of datasets. The dataset collection consists of multiple blocks. Each block can itself be a vtkMultiBlockDataSet, thus providing for a full tree structure. Sub-blocks are usually used to distribute blocks across processors. For example, a 1 block dataset can be distributed as following: @verbatim proc 0: Block 0: * ds 0 * (null)
proc 1: Block 0: * (null) * ds 1 @endverbatim
To create an instance of class vtkMultiBlockDataSet, simply invoke its constructor as follows

```
obj = vtkMultiBlockDataSet
```

31.128.2 Methods

The class vtkMultiBlockDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiBlockDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiBlockDataSet = obj.NewInstance ()`
- `vtkMultiBlockDataSet = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Set the number of blocks. This will cause allocation if the new number of blocks is greater than the current size. All new blocks are initialized to null.
- `obj.SetNumberOfBlocks (int numBlocks)` - Set the number of blocks. This will cause allocation if the new number of blocks is greater than the current size. All new blocks are initialized to null.
- `int = obj.GetNumberOfBlocks ()` - Returns the number of blocks.
- `vtkDataObject = obj.GetBlock (int blockno)` - Returns the block at the given index. It is recommended that one uses the iterators to iterate over composite datasets rather than using this API.
- `obj.SetBlock (int blockno, vtkDataObject block)` - Sets the data object as the given block. The total number of blocks will be resized to fit the requested block no.
- `obj.RemoveBlock (int blockno)` - Remove the given block from the dataset.
- `int = obj.HasMetaData (int blockno)` - Returns the meta-data for the block. If none is already present, a new vtkInformation object will be allocated. Use HasMetaData to avoid allocating vtkInformation objects.
- `vtkInformation = obj.GetMetaData (int blockno)` - Unhiding superclass method.
- `vtkInformation = obj.GetMetaData (vtkCompositeDataIterator iter)` - Unhiding superclass method.
- `int = obj.HasMetaData (vtkCompositeDataIterator iter)`

31.129 vtkMultiBlockDataSetAlgorithm

31.129.1 Usage

Algorithms that take any type of data object (including composite dataset) and produce a vtkMultiBlockDataSet in the output can subclass from this class.

To create an instance of class vtkMultiBlockDataSetAlgorithm, simply invoke its constructor as follows

```
obj = vtkMultiBlockDataSetAlgorithm
```

31.129.2 Methods

The class vtkMultiBlockDataSetAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiBlockDataSetAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiBlockDataSetAlgorithm = obj.NewInstance ()`
- `vtkMultiBlockDataSetAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkMultiBlockDataSet = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkMultiBlockDataSet = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.130 vtkMultiPieceDataSet

31.130.1 Usage

A vtkMultiPieceDataSet dataset groups multiple data pieces together. For example, say that a simulation broke a volume into 16 piece so that each piece can be processed with 1 process in parallel. We want to load this volume in a visualization cluster of 4 nodes. Each node will get 4 pieces, not necessarily forming a whole rectangular piece. In this case, it is not possible to append the 4 pieces together into a vtkImageData. In this case, these 4 pieces can be collected together using a vtkMultiPieceDataSet. Note that vtkMultiPieceDataSet is intended to be included in other composite datasets eg. vtkMultiBlockDataSet, vtkHierarchicalBoxDataSet. Hence the lack of algorithms producting vtkMultiPieceDataSet.

To create an instance of class vtkMultiPieceDataSet, simply invoke its constructor as follows

```
obj = vtkMultiPieceDataSet
```

31.130.2 Methods

The class vtkMultiPieceDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiPieceDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiPieceDataSet = obj.NewInstance ()`
- `vtkMultiPieceDataSet = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Set the number of pieces. This will cause allocation if the new number of pieces is greater than the current size. All new pieces are initialized to null.
- `obj.SetNumberOfPieces (int numpieces)` - Set the number of pieces. This will cause allocation if the new number of pieces is greater than the current size. All new pieces are initialized to null.
- `int = obj.GetNumberOfPieces ()` - Returns the number of pieces.
- `vtkDataSet = obj.GetPiece (int pieceno)` - Returns the piece at the given index.
- `vtkDataObject = obj.GetPieceAsDataObject (int pieceno)` - Returns the piece at the given index.
- `obj.SetPiece (int pieceno, vtkDataObject piece)` - Sets the data object as the given piece. The total number of pieces will be resized to fit the requested piece no.
- `int = obj.HasMetaData (int piece)` - Returns the meta-data for the piece. If none is already present, a new vtkInformation object will be allocated. Use HasMetaData to avoid allocating vtkInformation objects.
- `vtkInformation = obj.GetMetaData (int pieceno)` - Unhiding superclass method.
- `vtkInformation = obj.GetMetaData (vtkCompositeDataIterator iter)` - Unhiding superclass method.
- `int = obj.HasMetaData (vtkCompositeDataIterator iter)`

31.131 vtkMutableDirectedGraph

31.131.1 Usage

vtkMutableDirectedGraph is a directed graph which has additional methods for adding edges and vertices. AddChild() is a convenience method for constructing trees. ShallowCopy(), DeepCopy(), CheckedShallowCopy() and CheckedDeepCopy() will succeed for instances of vtkDirectedGraph, vtkMutableDirectedGraph and vtkTree.

To create an instance of class vtkMutableDirectedGraph, simply invoke its constructor as follows

```
obj = vtkMutableDirectedGraph
```

31.131.2 Methods

The class vtkMutableDirectedGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMutableDirectedGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMutableDirectedGraph = obj.NewInstance ()`
- `vtkMutableDirectedGraph = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.AddVertex ()` - Adds a vertex to the graph and returns the index of the new vertex.

  Note: In a distributed graph (i.e. a graph whose DistributedHelper is non-null), this routine cannot be used to add a vertex if the vertices in the graph have pedigree IDs, because this routine will always add the vertex locally, which may conflict with the proper location of the vertex based on the distribution of the pedigree IDs.
- `vtkIdType = obj.AddVertex (vtkVariantArray propertyArr)` - Adds a vertex to the graph with associated properties defined in propertyArr and returns the index of the new vertex. The number and order of values in propertyArr must match up with the arrays in the vertex data retrieved by GetVertexData().

  If a vertex with the given pedigree ID already exists, its properties will be overwritten with the properties in propertyArr and the existing vertex index will be returned.

  Note: In a distributed graph (i.e. a graph whose DistributedHelper is non-null) the vertex added or found might not be local. In this case, AddVertex will wait until the vertex can be added or found remotely, so that the proper vertex index can be returned. If you don't actually need to use the vertex index, consider calling LazyAddVertex, which provides better performance by eliminating the delays associated with returning the vertex index.
- `obj.LazyAddVertex ()` - Adds a vertex to the graph.

  This method is lazily evaluated for distributed graphs (i.e. graphs whose DistributedHelper is non-null) the next time Synchronize is called on the helper.
- `obj.LazyAddVertex (vtkVariantArray propertyArr)` - Adds a vertex to the graph with associated properties defined in propertyArr. The number and order of values in propertyArr must match up with the arrays in the vertex data retrieved by GetVertexData().

  If a vertex with the given pedigree ID already exists, its properties will be overwritten with the properties in propertyArr.

  This method is lazily evaluated for distributed graphs (i.e. graphs whose DistributedHelper is non-null) the next time Synchronize is called on the helper.
- `vtkGraphEdge = obj.AddGraphEdge (vtkIdType u, vtkIdType v)` - Variant of AddEdge() that returns a heavyweight vtkGraphEdge object. The graph owns the reference of the edge and will replace its contents on the next call to AddGraphEdge().

  Note: This is a less efficient method for use with wrappers. In C++ you should use the faster AddEdge().
- `vtkIdType = obj.AddChild (vtkIdType parent, vtkVariantArray propertyArr)` - Convenience method for creating trees. Returns the newly created vertex id. Shortcut for

```
vtkIdType v = g->AddVertex();
g->AddEdge(parent, v);
```

If non-null, propertyArr provides edge properties for the newly-created edge. The values in propertyArr must match up with the arrays in the edge data returned by GetEdgeData().

- `vtkIdType = obj.AddChild (vtkIdType parent)` - Removes the vertex from the graph along with any connected edges. Note: This invalidates the last vertex index, which is reassigned to v.
- `obj.RemoveVertex (vtkIdType v)` - Removes the vertex from the graph along with any connected edges. Note: This invalidates the last vertex index, which is reassigned to v.
- `obj.RemoveEdge (vtkIdType e)` - Removes the edge from the graph. Note: This invalidates the last edge index, which is reassigned to e.
- `obj.RemoveVertices (vtkIdTypeArray arr)` - Removes a collection of vertices from the graph along with any connected edges.
- `obj.RemoveEdges (vtkIdTypeArray arr)` - Removes a collection of edges from the graph.

31.132 vtkMutableUndirectedGraph

31.132.1 Usage

vtkMutableUndirectedGraph is an undirected graph with additional functions for adding vertices and edges. ShallowCopy(), DeepCopy(), CheckedShallowCopy(), and CheckedDeepCopy() will succeed when the argument is a vtkUndirectedGraph or vtkMutableUndirectedGraph.

To create an instance of class vtkMutableUndirectedGraph, simply invoke its constructor as follows

```
obj = vtkMutableUndirectedGraph
```

31.132.2 Methods

The class vtkMutableUndirectedGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMutableUndirectedGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMutableUndirectedGraph = obj.NewInstance ()`
- `vtkMutableUndirectedGraph = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.AddVertex ()` - Adds a vertex to the graph and returns the index of the new vertex.

  Note: In a distributed graph (i.e. a graph whose DistributedHelper is non-null), this routine cannot be used to add a vertex if the vertices in the graph have pedigree IDs, because this routine will always add the vertex locally, which may conflict with the proper location of the vertex based on the distribution of the pedigree IDs.
- `vtkIdType = obj.AddVertex (vtkVariantArray propertyArr)` - Adds a vertex to the graph with associated properties defined in propertyArr and returns the index of the new vertex. The number and order of values in propertyArr must match up with the arrays in the vertex data retrieved by GetVertexData().

  If a vertex with the given pedigree ID already exists, its properties will be overwritten with the properties in propertyArr and the existing vertex index will be returned.

  Note: In a distributed graph (i.e. a graph whose DistributedHelper is non-null) the vertex added or found might not be local. In this case, AddVertex will wait until the vertex can be added or found

remotely, so that the proper vertex index can be returned. If you don't actually need to use the vertex index, consider calling LazyAddVertex, which provides better performance by eliminating the delays associated with returning the vertex index.

- `obj.LazyAddVertex ()` - Adds a vertex to the graph.

  This method is lazily evaluated for distributed graphs (i.e. graphs whose DistributedHelper is non-null) the next time Synchronize is called on the helper.

- `obj.LazyAddVertex (vtkVariantArray propertyArr)` - Adds a vertex to the graph with associated properties defined in propertyArr. The number and order of values in propertyArr must match up with the arrays in the vertex data retrieved by GetVertexData().

  If a vertex with the given pedigree ID already exists, its properties will be overwritten with the properties in propertyArr.

  This method is lazily evaluated for distributed graphs (i.e. graphs whose DistributedHelper is non-null) the next time Synchronize is called on the helper.

- `obj.LazyAddEdge (vtkIdType u, vtkIdType v)` - Adds an undirected edge from u to v, where u and v are vertex indices.

  This method is lazily evaluated for distributed graphs (i.e. graphs whose DistributedHelper is non-null) the next time Synchronize is called on the helper.

- `obj.LazyAddEdge (vtkIdType u, vtkIdType v, vtkVariantArray propertyArr)` - Adds an undirected edge from u to v, where u and v are vertex indices.

  The number and order of values in propertyArr must match up with the arrays in the edge data retrieved by GetEdgeData().

  This method is lazily evaluated for distributed graphs (i.e. graphs whose DistributedHelper is non-null) the next time Synchronize is called on the helper.

- `vtkGraphEdge = obj.AddGraphEdge (vtkIdType u, vtkIdType v)` - Variant of AddEdge() that returns a heavyweight vtkGraphEdge object. The graph owns the reference of the edge and will replace its contents on the next call to AddGraphEdge().

  Note: This is a less efficient method for use with wrappers. In C++ you should use the faster AddEdge().

- `obj.RemoveVertex (vtkIdType v)` - Removes the vertex from the graph along with any connected edges. Note: This invalidates the last vertex index, which is reassigned to v.

- `obj.RemoveEdge (vtkIdType e)` - Removes the edge from the graph. Note: This invalidates the last edge index, which is reassigned to e.

- `obj.RemoveVertices (vtkIdTypeArray arr)` - Removes a collection of vertices from the graph along with any connected edges.

- `obj.RemoveEdges (vtkIdTypeArray arr)` - Removes a collection of edges from the graph.

31.133 vtkNonLinearCell

31.133.1 Usage

vtkNonLinearCell is an abstract superclass for non-linear cell types. Cells that are a direct subclass of vtkCell or vtkCell3D are linear; cells that are a subclass of vtkNonLinearCell have non-linear interpolation functions. Non-linear cells require special treatment when tessellating or converting to graphics primitives. Note that the linearity of the cell is a function of whether the cell needs tessellation, which does not strictly correlate with interpolation order (e.g., vtkHexahedron has non-linear interpolation functions (a product of three linear functions in r-s-t) even thought vtkHexahedron is considered linear.)

To create an instance of class vtkNonLinearCell, simply invoke its constructor as follows

```
obj = vtkNonLinearCell
```

31.133.2 Methods

The class vtkNonLinearCell has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkNonLinearCell class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkNonLinearCell = obj.NewInstance ()`
- `vtkNonLinearCell = obj.SafeDownCast (vtkObject o)`
- `int = obj.IsLinear ()`

31.134 vtkNonMergingPointLocator

31.134.1 Usage

As a special sub-class of vtkPointLocator, vtkNonMergingPointLocator is intended for direct / check-free insertion of points into a vtkPoints object. In other words, any given point is always directly inserted. The name emphasizes the difference between this class and its sibling class vtkMergePoints in that the latter class performs check-based zero tolerance point insertion (or to 'merge' exactly duplicate / coincident points) by exploiting the uniform bin mechanism employed by the parent class vtkPointLocator. vtkPointLocator allows for generic (zero and non- zero) tolerance point insertion as well as point location.

To create an instance of class vtkNonMergingPointLocator, simply invoke its constructor as follows

```
obj = vtkNonMergingPointLocator
```

31.134.2 Methods

The class vtkNonMergingPointLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkNonMergingPointLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkNonMergingPointLocator = obj.NewInstance ()`
- `vtkNonMergingPointLocator = obj.SafeDownCast (vtkObject o)`

31.135 vtkOctreePointLocator

31.135.1 Usage

Given a vtkDataSetxs, create an octree that is locally refined such that all leaf octants contain less than a certain amount of points. Note that there is no size constraint that a leaf octant in relation to any of its neighbors.

This class can also generate a PolyData representation of the boundaries of the spatial regions in the decomposition.

To create an instance of class vtkOctreePointLocator, simply invoke its constructor as follows

```
obj = vtkOctreePointLocator
```

31.135.2 Methods

The class vtkOctreePointLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOctreePointLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOctreePointLocator = obj.NewInstance ()`
- `vtkOctreePointLocator = obj.SafeDownCast (vtkObject o)`
- `obj.SetMaximumPointsPerRegion (int )` - Maximum number of points per spatial region. Default is 100.
- `int = obj.GetMaximumPointsPerRegion ()` - Maximum number of points per spatial region. Default is 100.
- `obj.SetCreateCubicOctants (int )` - Get/Set macro for CreateCubicOctants.
- `int = obj.GetCreateCubicOctants ()` - Get/Set macro for CreateCubicOctants.
- `double = obj.GetFudgeFactor ()` - Some algorithms on octrees require a value that is a very small distance relative to the diameter of the entire space divided by the octree. This factor is the maximum axis-aligned width of the space multipled by 10e-6.
- `obj.SetFudgeFactor (double )` - Some algorithms on octrees require a value that is a very small distance relative to the diameter of the entire space divided by the octree. This factor is the maximum axis-aligned width of the space multipled by 10e-6.
- `obj.GetBounds (double bounds)` - Get the spatial bounds of the entire octree space. Sets bounds array to xmin, xmax, ymin, ymax, zmin, zmax.
- `int = obj.GetNumberOfLeafNodes ()` - The number of leaf nodes of the tree, the spatial regions
- `obj.GetRegionBounds (int regionID, double bounds[6])` - Get the spatial bounds of octree region
- `obj.GetRegionDataBounds (int leafNodeID, double bounds[6])` - Get the bounds of the data within the leaf node
- `int = obj.GetRegionContainingPoint (double x, double y, double z)` - Get the id of the leaf region containing the specified location.
- `obj.BuildLocator ()` - Create the octree decomposition of the cells of the data set or data sets. Cells are assigned to octree spatial regions based on the location of their centroids.
- `vtkIdType = obj.FindClosestPoint (double x[3])` - Return the Id of the point that is closest to the given point. Set the square of the distance between the two points.
- `obj.FindPointsWithinRadius (double radius, double x[3], vtkIdList result)` - Find all points within a specified radius of position x. The result is not sorted in any specific manner.
- `obj.FindClosestNPoints (int N, double x[3], vtkIdList result)` - Find the closest N points to a position. This returns the closest N points to a position. A faster method could be created that returned N close points to a position, but not necessarily the exact N closest. The returned points are sorted from closest to farthest. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.

- `vtkIdTypeArray = obj.GetPointsInRegion (int leafNodeId)` - Get a list of the original IDs of all points in a leaf node.
- `obj.FreeSearchStructure ()` - Delete the octree data structure.
- `obj.GenerateRepresentation (int level, vtkPolyData pd)` - Create a polydata representation of the boundaries of the octree regions.
- `obj.FindPointsInArea (double area, vtkIdTypeArray ids, bool clearArraytrue)` - Fill ids with points found in area. The area is a 6-tuple containing (xmin, xmax, ymin, ymax, zmin, zmax). This method will clear the array by default. To append ids to an array, set clearArray to false.

31.136 vtkOctreePointLocatorNode

31.136.1 Usage

This class represents a single spatial region in a 3D axis octant partitioning. It is intended to work efficiently with the vtkOctreePointLocator and is not meant for general use. It is assumed the region bounds some set of points. The ordering of the children is (-x,-y,-z),(+x,-y,-z),(-x,+y,-z),(+x,+y,-z),(-x,-y,+z),(+x,-y,+z), (-x,+y,+z),(+x,+y,+z). The portion of the domain assigned to an octant is Min ¡ x ¡= Max.

To create an instance of class vtkOctreePointLocatorNode, simply invoke its constructor as follows

```
obj = vtkOctreePointLocatorNode
```

31.136.2 Methods

The class vtkOctreePointLocatorNode has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOctreePointLocatorNode class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOctreePointLocatorNode = obj.NewInstance ()`
- `vtkOctreePointLocatorNode = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfPoints (int numberOfPoints)` - Set/Get the number of points contained in this region.
- `int = obj.GetNumberOfPoints ()` - Set/Get the number of points contained in this region.
- `obj.SetBounds (double xMin, double xMax, double yMin, double yMax, double zMin, double zMax)` - Set/Get the bounds of the spatial region represented by this node. Caller allocates storage for 6-vector in GetBounds.
- `obj.SetBounds (double b[6])` - Set/Get the bounds of the spatial region represented by this node. Caller allocates storage for 6-vector in GetBounds.
- `obj.GetBounds (double b) const` - Set/Get the bounds of the spatial region represented by this node. Caller allocates storage for 6-vector in GetBounds.
- `obj.SetDataBounds (double xMin, double xMax, double yMin, double yMax, double zMin, double zMax)` - Set/Get the bounds of the points contained in this spatial region. This may be smaller than the bounds of the region itself. Caller allocates storage for 6-vector in GetDataBounds.

- `obj.GetDataBounds (double b) const` - Set/Get the bounds of the points contained in this spatial region. This may be smaller than the bounds of the region itself. Caller allocates storage for 6-vector in GetDataBounds.

- `obj.SetMinBounds (double minBounds[3])` - Set the xmax, ymax and zmax value of the bounds of this region

- `obj.SetMaxBounds (double maxBounds[3])` - Set the xmin, ymin and zmin value of the bounds of this data within this region.

- `obj.SetMinDataBounds (double minDataBounds[3])` - Set the xmax, ymax and zmax value of the bounds of this data within this region.

- `obj.SetMaxDataBounds (double maxDataBounds[3])` - Get the ID associated with the region described by this node. If this is not a leaf node, this value should be -1.

- `int = obj.GetID ()` - Get the ID associated with the region described by this node. If this is not a leaf node, this value should be -1.

- `int = obj.GetMinID ()` - If this node is not a leaf node, there are leaf nodes below it whose regions represent a partitioning of this region. The IDs of these leaf nodes form a contigous set. Get the first of the first point's ID that is contained in this node.

- `obj.CreateChildNodes ()` - Add the 8 children.

- `obj.DeleteChildNodes ()` - Delete the 8 children.

- `vtkOctreePointLocatorNode = obj.GetChild (int i)` - Get a pointer to the ith child of this node.

- `int = obj.IntersectsRegion (vtkPlanesIntersection pi, int useDataBounds)` - A vtkPlanesIntersection object represents a convex 3D region bounded by planes, and it is capable of computing intersections of boxes with itself. Return 1 if this spatial region intersects the spatial region described by the vtkPlanesIntersection object. Use the possibly smaller bounds of the points within the region if useDataBounds is non-zero.

- `int = obj.ContainsPoint (double x, double y, double z, int useDataBounds)` - Return 1 if this spatial region entirely contains the given point. Use the possibly smaller bounds of the points within the region if useDataBounds is non-zero.

- `double = obj.GetDistance2ToBoundary (double x, double y, double z, vtkOctreePointLocatorNode top, i` - Calculate the distance squared from any point to the boundary of this region. Use the boundary of the points within the region if useDataBounds is non-zero.

- `double = obj.GetDistance2ToBoundary (double x, double y, double z, double boundaryPt, vtkOctreePoir` - Calculate the distance squared from any point to the boundary of this region. Use the boundary of the points within the region if useDataBounds is non-zero. Set boundaryPt to the point on the boundary.

- `double = obj.GetDistance2ToInnerBoundary (double x, double y, double z, vtkOctreePointLocatorNode t` - Calculate the distance from the specified point (which is required to be inside this spatial region) to an interior boundary. An interior boundary is one that is not also an boundary of the entire space partitioned by the tree of vtkOctreePointLocatorNode's.

- `int = obj.GetSubOctantIndex (double point, int CheckContainment)` - Return the id of the suboctant that a given point is in. If CheckContainment is non-zero then it checks whether the point is in the actual bounding box of the suboctant, otherwise it only checks which octant the point is in that is created from the axis-aligned partitioning of the domain at this octant's center.

31.137 vtkOrderedTriangulator

31.137.1 Usage

This class is used to generate unique triangulations of points. The uniqueness of the triangulation is controlled by the id of the inserted points in combination with a Delaunay criterion. The class is designed to be as fast as possible (since the algorithm can be slow) and uses block memory allocations to support rapid triangulation generation. Also, the assumption behind the class is that a maximum of hundreds of points are to be triangulated. If you desire more robust triangulation methods use vtkPolygon::Triangulate(), vtkDelaunay2D, or vtkDelaunay3D.

.SECTION Background This work is documented in the technical paper: W.J. Schroeder, B. Geveci, M. Malaterre. Compatible Triangulations of Spatial Decompositions. In Proceedings of Visualization 2004, IEEE Press October 2004.

Delaunay triangulations are unique assuming a random distribution of input points. The 3D Delaunay criterion is as follows: the circumsphere of each tetrahedron contains no other points of the triangulation except for the four points defining the tetrahedron. In application this property is hard to satisfy because objects like cubes are defined by eight points all sharing the same circumsphere (center and radius); hence the Delaunay triangulation is not unique. These so-called degenerate situations are typically resolved by arbitrary selecting a triangulation. This code does something different: it resolves degenerate triangulations by modifying the "InCircumsphere" method to use a slightly smaller radius. Hence, degenerate points are always considered "out" of the circumsphere. This, in combination with an ordering (based on id) of the input points, guarantees a unique triangulation.

There is another related characteristic of Delaunay triangulations. Given a N-dimensional Delaunay triangulation, points lying on a (N-1) dimensional plane also form a (N-1) Delaunay triangulation. This means for example, that if a 3D cell is defined by a set of (2D) planar faces, then the face triangulations are Delaunay. Combining this with the method to generate unique triangulations described previously, the triangulations on the face are guaranteed unique. This fact can be used to triangulate 3D objects in such a way to guarantee compatible face triangulations. This is a very useful fact for parallel processing, or performing operations like clipping that require compatible triangulations across 3D cell faces. (See vtkClipVolume for an example.)

A special feature of this class is that it can generate triangulation templates on the fly. If template triangulation is enabled, then the ordered triangulator will first triangulate the cell using the slower ordered Delaunay approach, and then store the result as a template. Later, if the same cell type and cell configuration is encountered, then the template is reused which greatly speeds the triangulation.

To create an instance of class vtkOrderedTriangulator, simply invoke its constructor as follows

```
obj = vtkOrderedTriangulator
```

31.137.2 Methods

The class vtkOrderedTriangulator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOrderedTriangulator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOrderedTriangulator = obj.NewInstance ()`
- `vtkOrderedTriangulator = obj.SafeDownCast (vtkObject o)`
- `obj.InitTriangulation (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax`
  - Initialize the triangulation process. Provide a bounding box and the maximum number of points to be inserted. Note that since the triangulation is performed using parametric coordinates (see Insert-Point()) the bounds should be represent the range of the parametric coordinates inserted.

- `obj.InitTriangulation (double bounds[6], int numPts)` - Initialize the triangulation process. Provide a bounding box and the maximum number of points to be inserted. Note that since the triangulation is performed using parametric coordinates (see InsertPoint()) the bounds should be represent the range of the parametric coordinates inserted.

- `vtkIdType = obj.InsertPoint (vtkIdType id, double x[3], double p[3], int type)` - For each point to be inserted, provide an id, a position x, parametric coordinate p, and whether the point is inside (type=0), outside (type=1), or on the boundary (type=2). You must call InitTriangulation() prior to invoking this method. Make sure that the number of points inserted does not exceed the numPts specified in InitTriangulation(). Also note that the "id" can be any integer and can be greater than numPts. It is used to create tetras (in AddTetras()) with the appropriate connectivity ids. The method returns an internal id that can be used prior to the Triangulate() method to update the type of the point with UpdatePointType(). (Note: the algorithm triangulated with the parametric coordinate p[3] and creates tetras with the global coordinate x[3]. The parametric coordinates and global coordinates may be the same.)

- `vtkIdType = obj.InsertPoint (vtkIdType id, vtkIdType sortid, double x[3], double p[3], int type)` - For each point to be inserted, provide an id, a position x, parametric coordinate p, and whether the point is inside (type=0), outside (type=1), or on the boundary (type=2). You must call InitTriangulation() prior to invoking this method. Make sure that the number of points inserted does not exceed the numPts specified in InitTriangulation(). Also note that the "id" can be any integer and can be greater than numPts. It is used to create tetras (in AddTetras()) with the appropriate connectivity ids. The method returns an internal id that can be used prior to the Triangulate() method to update the type of the point with UpdatePointType(). (Note: the algorithm triangulated with the parametric coordinate p[3] and creates tetras with the global coordinate x[3]. The parametric coordinates and global coordinates may be the same.)

- `vtkIdType = obj.InsertPoint (vtkIdType id, vtkIdType sortid, vtkIdType sortid2, double x[3], double` - For each point to be inserted, provide an id, a position x, parametric coordinate p, and whether the point is inside (type=0), outside (type=1), or on the boundary (type=2). You must call InitTriangulation() prior to invoking this method. Make sure that the number of points inserted does not exceed the numPts specified in InitTriangulation(). Also note that the "id" can be any integer and can be greater than numPts. It is used to create tetras (in AddTetras()) with the appropriate connectivity ids. The method returns an internal id that can be used prior to the Triangulate() method to update the type of the point with UpdatePointType(). (Note: the algorithm triangulated with the parametric coordinate p[3] and creates tetras with the global coordinate x[3]. The parametric coordinates and global coordinates may be the same.)

- `obj.Triangulate ()` - Perform the triangulation. (Complete all calls to InsertPoint() prior to invoking this method.) A special version is available when templates should be used.

- `obj.TemplateTriangulate (int cellType, int numPts, int numEdges)` - Perform the triangulation. (Complete all calls to InsertPoint() prior to invoking this method.) A special version is available when templates should be used.

- `obj.UpdatePointType (vtkIdType internalId, int type)` - Update the point type. This is useful when the merging of nearly coincident points is performed. The id is the internal id returned from InsertPoint(). The method should be invoked prior to the Triangulate method. The type is specified as inside (type=0), outside (type=1), or on the boundary (type=2).

- `vtkIdType = obj.GetPointId (vtkIdType internalId)` - Return the Id of point 'internalId'. This id is the one passed in argument of InsertPoint. It assumes that the point has already been inserted. The method should be invoked prior to the Triangulate method.

- `int = obj.GetNumberOfPoints ()` - Return the number of inserted points.

- `obj.SetUseTemplates (int )` - If this flag is set, then the ordered triangulator will create and use templates for the triangulation. To use templates, the TemplateTriangulate() method should be called when appropriate. (Note: the TemplateTriangulate() method works for complete (interior) cells without extra points due to intersection, etc.)

- `int = obj.GetUseTemplates ()` - If this flag is set, then the ordered triangulator will create and use templates for the triangulation. To use templates, the TemplateTriangulate() method should be called when appropriate. (Note: the TemplateTriangulate() method works for complete (interior) cells without extra points due to intersection, etc.)

- `obj.UseTemplatesOn ()` - If this flag is set, then the ordered triangulator will create and use templates for the triangulation. To use templates, the TemplateTriangulate() method should be called when appropriate. (Note: the TemplateTriangulate() method works for complete (interior) cells without extra points due to intersection, etc.)

- `obj.UseTemplatesOff ()` - If this flag is set, then the ordered triangulator will create and use templates for the triangulation. To use templates, the TemplateTriangulate() method should be called when appropriate. (Note: the TemplateTriangulate() method works for complete (interior) cells without extra points due to intersection, etc.)

- `obj.SetPreSorted (int )` - Boolean indicates whether the points have been pre-sorted. If pre-sorted is enabled, the points are not sorted on point id. By default, presorted is off. (The point id is defined in InsertPoint().)

- `int = obj.GetPreSorted ()` - Boolean indicates whether the points have been pre-sorted. If pre-sorted is enabled, the points are not sorted on point id. By default, presorted is off. (The point id is defined in InsertPoint().)

- `obj.PreSortedOn ()` - Boolean indicates whether the points have been pre-sorted. If pre-sorted is enabled, the points are not sorted on point id. By default, presorted is off. (The point id is defined in InsertPoint().)

- `obj.PreSortedOff ()` - Boolean indicates whether the points have been pre-sorted. If pre-sorted is enabled, the points are not sorted on point id. By default, presorted is off. (The point id is defined in InsertPoint().)

- `obj.SetUseTwoSortIds (int )` - Tells the triangulator that a second sort id is provided for each point and should also be considered when sorting.

- `int = obj.GetUseTwoSortIds ()` - Tells the triangulator that a second sort id is provided for each point and should also be considered when sorting.

- `obj.UseTwoSortIdsOn ()` - Tells the triangulator that a second sort id is provided for each point and should also be considered when sorting.

- `obj.UseTwoSortIdsOff ()` - Tells the triangulator that a second sort id is provided for each point and should also be considered when sorting.

- `vtkIdType = obj.GetTetras (int classification, vtkUnstructuredGrid ugrid)` - Initialize and add the tetras and points from the triangulation to the unstructured grid provided. New points are created and the mesh is allocated. (This method differs from AddTetras() in that it inserts points and cells; AddTetras only adds the tetra cells.) The tetrahdera added are of the type specified (0=inside,1=outside,2=all). Inside tetrahedron are those whose points are classified "inside" or on the "boundary." Outside tetrahedron have at least one point classified "outside." The method returns the number of tetrahedrahedron of the type requested.

- `vtkIdType = obj.AddTetras (int classification, vtkUnstructuredGrid ugrid)` - Add the tetras to the unstructured grid provided. The unstructured grid is assumed to have been initialized (with

Allocate()) and points set (with SetPoints()). The tetrahdera added are of the type specified (0=inside,1=outside,2=all). Inside tetrahedron are those whose points are classified "inside" or on the "boundary." Outside tetrahedron have at least one point classified "outside." The method returns the number of tetrahedrahedron of the type requested.

- `vtkIdType = obj.AddTetras (int classification, vtkCellArray connectivity)` - Add the tetrahedra classified (0=inside,1=outside) to the connectivity list provided. Inside tetrahedron are those whose points are all classified "inside." Outside tetrahedron have at least one point classified "outside." The method returns the number of tetrahedron of the type requested.
- `vtkIdType = obj.AddTetras (int classification, vtkIncrementalPointLocator locator, vtkCellArray out` - Assuming that all the inserted points come from a cell 'cellId' to triangulate, get the tetrahedra in outConnectivity, the points in locator and copy point data and cell data. Return the number of added tetras.
- `vtkIdType = obj.AddTetras (int classification, vtkIdList ptIds, vtkPoints pts)` - Add the tetrahedra classified (0=inside,1=outside) to the list of ids and coordinates provided. These assume that the first four points form a tetrahedron, the next four the next, and so on.
- `vtkIdType = obj.AddTriangles (vtkCellArray connectivity)` - Add the triangle faces classified (2=boundary) to the connectivity list provided. The method returns the number of triangles.
- `vtkIdType = obj.AddTriangles (vtkIdType id, vtkCellArray connectivity)` - Add the triangle faces classified (2=boundary) and attached to the specified point id to the connectivity list provided. (The id is the same as that specified in InsertPoint().)
- `obj.InitTetraTraversal ()` - Methods to get one tetra at a time. Start with InitTetraTraversal() and then invoke GetNextTetra() until the method returns 0.
- `int = obj.GetNextTetra (int classification, vtkTetra tet, vtkDataArray cellScalars, vtkDoubleArray` - Methods to get one tetra at a time. Start with InitTetraTraversal() and then invoke GetNextTetra() until the method returns 0. cellScalars are point-centered scalars on the original cell. tetScalars are point-centered scalars on the tetra: the values will be copied from cellScalars.

31.138 vtkOutEdgeIterator

31.138.1 Usage

vtkOutEdgeIterator iterates through all edges whose source is a particular vertex. Instantiate this class directly and call Initialize() to traverse the vertex of a graph. Alternately, use GetInEdges() on the graph to initialize the iterator. it-¿Next() returns a vtkOutEdgeType structure, which contains Id, the edge's id, and Target, the edge's target vertex.

To create an instance of class vtkOutEdgeIterator, simply invoke its constructor as follows

```
obj = vtkOutEdgeIterator
```

31.138.2 Methods

The class vtkOutEdgeIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOutEdgeIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOutEdgeIterator = obj.NewInstance ()`

- `vtkOutEdgeIterator = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (vtkGraph g, vtkIdType v)` - Initialize the iterator with a graph and vertex.
- `vtkGraph = obj.GetGraph ()` - Get the graph and vertex associated with this iterator.
- `vtkIdType = obj.GetVertex ()` - Get the graph and vertex associated with this iterator.
- `vtkGraphEdge = obj.NextGraphEdge ()` - Just like Next(), but returns heavy-weight vtkGraphEdge object instead of the vtkEdgeType struct, for use with wrappers. The graph edge is owned by this iterator, and changes after each call to NextGraphEdge().
- `bool = obj.HasNext ()`

31.139 vtkParametricSpline

31.139.1 Usage

vtkParametricSpline is a parametric function for 1D interpolating splines. vtkParametricSpline maps the single parameter u into a 3D point (x,y,z) using three instances of interpolating splines. This family of 1D splines is quaranteed to be parameterized in the interval [0,1]. Attempting to evaluate outside this interval will cause the parameter u to be clamped in the range [0,1].

When constructed, this class creates instances of vtkCardinalSpline for each of the x-y-z coordinates. The user may choose to replace these with their own instances of subclasses of vtkSpline.

To create an instance of class vtkParametricSpline, simply invoke its constructor as follows

```
obj = vtkParametricSpline
```

31.139.2 Methods

The class vtkParametricSpline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricSpline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricSpline = obj.NewInstance ()`
- `vtkParametricSpline = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Evaluate the spline at parametric coordinate u[0] returning the point coordinate Pt[3].
- `obj.Evaluate (double u[3], double Pt[3], double Du[9])` - Evaluate the spline at parametric coordinate u[0] returning the point coordinate Pt[3].
- `double = obj.EvaluateScalar (double u[3], double Pt[3], double Du[9])` - Evaluate a scalar value at parametric coordinate u[0] and Pt[3]. The scalar value is just the parameter u[0].
- `obj.SetXSpline (vtkSpline )` - By default, this class is constructed with three instances of vtkCardinalSpline (for each of the x-y-z coordinate axes). The user may choose to create and assign their own instances of vtkSpline.
- `obj.SetYSpline (vtkSpline )` - By default, this class is constructed with three instances of vtkCardinalSpline (for each of the x-y-z coordinate axes). The user may choose to create and assign their own instances of vtkSpline.

- `obj.SetZSpline (vtkSpline )` - By default, this class is constructed with three instances of vtkCardinalSpline (for each of the x-y-z coordinate axes). The user may choose to create and assign their own instances of vtkSpline.
- `vtkSpline = obj.GetXSpline ()` - By default, this class is constructed with three instances of vtkCardinalSpline (for each of the x-y-z coordinate axes). The user may choose to create and assign their own instances of vtkSpline.
- `vtkSpline = obj.GetYSpline ()` - By default, this class is constructed with three instances of vtkCardinalSpline (for each of the x-y-z coordinate axes). The user may choose to create and assign their own instances of vtkSpline.
- `vtkSpline = obj.GetZSpline ()` - By default, this class is constructed with three instances of vtkCardinalSpline (for each of the x-y-z coordinate axes). The user may choose to create and assign their own instances of vtkSpline.
- `obj.SetPoints (vtkPoints )` - Specify the list of points defining the spline. Do this by specifying a vtkPoints array containing the points. Note that the order of the points in vtkPoints is the order that the splines will be fit.
- `vtkPoints = obj.GetPoints ()` - Specify the list of points defining the spline. Do this by specifying a vtkPoints array containing the points. Note that the order of the points in vtkPoints is the order that the splines will be fit.
- `obj.SetNumberOfPoints (vtkIdType numPts)` - Another API to set the points. Set the number of points and then set the individual point coordinates.
- `obj.SetPoint (vtkIdType index, double x, double y, double z)` - Another API to set the points. Set the number of points and then set the individual point coordinates.
- `obj.SetClosed (int )` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous.
- `int = obj.GetClosed ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous.
- `obj.ClosedOn ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous.
- `obj.ClosedOff ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous.
- `obj.SetParameterizeByLength (int )` - Control whether the spline is parameterized by length or by point index. Default is by length.
- `int = obj.GetParameterizeByLength ()` - Control whether the spline is parameterized by length or by point index. Default is by length.
- `obj.ParameterizeByLengthOn ()` - Control whether the spline is parameterized by length or by point index. Default is by length.
- `obj.ParameterizeByLengthOff ()` - Control whether the spline is parameterized by length or by point index. Default is by length.
- `obj.SetLeftConstraint (int )` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

2: the second derivative at left(right) most point is set to Left(Right)Value.

3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetLeftConstraintMinValue ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetLeftConstraintMaxValue ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetLeftConstraint ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `obj.SetRightConstraint (int )` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetRightConstraintMinValue ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetRightConstraintMaxValue ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetRightConstraint ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `obj.SetLeftValue (double )` - The values of the derivative on the left and right sides. The value is used only if the left(right) constraint is type 1-3.

- `double = obj.GetLeftValue ()` - The values of the derivative on the left and right sides. The value is used only if the left(right) constraint is type 1-3.

- `obj.SetRightValue (double )` - The values of the derivative on the left and right sides. The value is used only if the left(right) constraint is type 1-3.

- `double = obj.GetRightValue ()` - The values of the derivative on the left and right sides. The value is used only if the left(right) constraint is type 1-3.

31.140 vtkPassInputTypeAlgorithm

31.140.1 Usage

vtkPassInputTypeAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline architecture. Ther are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this classes contstructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be DataObject. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into seperate functions such as RequestDataObject RequestData and RequestInformation. The default implementation of RequestDataObject will create an output data of the same type as the input.

To create an instance of class vtkPassInputTypeAlgorithm, simply invoke its constructor as follows

```
obj = vtkPassInputTypeAlgorithm
```

31.140.2 Methods

The class vtkPassInputTypeAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPassInputTypeAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPassInputTypeAlgorithm = obj.NewInstance ()`
- `vtkPassInputTypeAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkDataObject = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `vtkPolyData = obj.GetPolyDataOutput ()` - Get the output as vtkPolyData.
- `vtkStructuredPoints = obj.GetStructuredPointsOutput ()` - Get the output as vtkStructuredPoints.
- `vtkImageData = obj.GetImageDataOutput ()` - Get the output as vtkStructuredPoints.
- `vtkStructuredGrid = obj.GetStructuredGridOutput ()` - Get the output as vtkStructuredGrid.
- `vtkUnstructuredGrid = obj.GetUnstructuredGridOutput ()` - Get the output as vtkUnstructuredGrid.
- `vtkRectilinearGrid = obj.GetRectilinearGridOutput ()` - Get the output as vtkRectilinearGrid.
- `vtkTable = obj.GetTableOutput ()` - Get the output as vtkTable.
- `vtkGraph = obj.GetGraphOutput ()` - Get the output as vtkGraph.
- `vtkDataObject = obj.GetInput ()` - Get the input data object. This method is not recommended for use, but lots of old style filters use it.
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.141 vtkPentagonalPrism

31.141.1 Usage

vtkPentagonalPrism is a concrete implementation of vtkCell to represent a linear 3D prism with pentagonal base. Such prism is defined by the ten points (0-9) where (0,1,2,3,4) is the base of the prism which, using the right hand rule, forms a pentagon whose normal points is in the direction of the opposite face (5,6,7,8,9).

To create an instance of class vtkPentagonalPrism, simply invoke its constructor as follows

```
obj = vtkPentagonalPrism
```

31.141.2 Methods

The class vtkPentagonalPrism has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPentagonalPrism class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPentagonalPrism = obj.NewInstance ()`
- `vtkPentagonalPrism = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell3D API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell3D API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell3D API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell3D API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell3D API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - See the vtkCell3D API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell3D API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the wedge in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[10])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[30])` - Return the ids of the vertices defining edge/face (‘edgeId‘/‘faceId’). Ids are related to the cell, not to the dataset.

31.142 vtkPerlinNoise

31.142.1 Usage

vtkPerlinNoise computes a Perlin noise field as an implicit function. vtkPerlinNoise is a concrete implementation of vtkImplicitFunction. Perlin noise, originally described by Ken Perlin, is a non-periodic and continuous noise function useful for modeling real-world objects.

The amplitude and frequency of the noise pattern are adjustable. This implementation of Perlin noise is derived closely from Greg Ward’s version in Graphics Gems II.

To create an instance of class vtkPerlinNoise, simply invoke its constructor as follows

```
obj = vtkPerlinNoise
```

31.142.2 Methods

The class vtkPerlinNoise has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPerlinNoise class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPerlinNoise = obj.NewInstance ()`
- `vtkPerlinNoise = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])` - Evaluate PerlinNoise function.
- `double = obj.EvaluateFunction (double x, double y, double z)` - Evaluate PerlinNoise function.
- `obj.EvaluateGradient (double x[3], double n[3])` - Evaluate PerlinNoise gradient. Currently, the method returns a 0 gradient.
- `obj.SetFrequency (double , double , double )` - Set/get the frequency, or physical scale, of the noise function (higher is finer scale). The frequency can be adjusted per axis, or the same for all axes.
- `obj.SetFrequency (double  a[3])` - Set/get the frequency, or physical scale, of the noise function (higher is finer scale). The frequency can be adjusted per axis, or the same for all axes.
- `double = obj. GetFrequency ()` - Set/get the frequency, or physical scale, of the noise function (higher is finer scale). The frequency can be adjusted per axis, or the same for all axes.
- `obj.SetPhase (double , double , double )` - Set/get the phase of the noise function. This parameter can be used to shift the noise function within space (perhaps to avoid a beat with a noise pattern at another scale). Phase tends to repeat about every unit, so a phase of 0.5 is a half-cycle shift.
- `obj.SetPhase (double  a[3])` - Set/get the phase of the noise function. This parameter can be used to shift the noise function within space (perhaps to avoid a beat with a noise pattern at another scale). Phase tends to repeat about every unit, so a phase of 0.5 is a half-cycle shift.
- `double = obj. GetPhase ()` - Set/get the phase of the noise function. This parameter can be used to shift the noise function within space (perhaps to avoid a beat with a noise pattern at another scale). Phase tends to repeat about every unit, so a phase of 0.5 is a half-cycle shift.
- `obj.SetAmplitude (double )` - Set/get the amplitude of the noise function. Amplitude can be negative. The noise function varies randomly between -—Amplitude— and —Amplitude—. Therefore the range of values is 2*—Amplitude— large. The initial amplitude is 1.
- `double = obj.GetAmplitude ()` - Set/get the amplitude of the noise function. Amplitude can be negative. The noise function varies randomly between -—Amplitude— and —Amplitude—. Therefore the range of values is 2*—Amplitude— large. The initial amplitude is 1.

31.143 vtkPiecewiseFunction

31.143.1 Usage

Defines a piecewise function mapping. This mapping allows the addition of control points, and allows the user to control the function between the control points. A piecewise hermite curve is used between control points, based on the sharpness and midpoint parameters. A sharpness of 0 yields a piecewise linear function and a sharpness of 1 yields a piecewise constant function. The midpoint is the normalized distance between

control points at which the curve reaches the median Y value. The midpoint and sharpness values specified when adding a node are used to control the transition to the next node (the last node's values are ignored) Outside the range of nodes, the values are 0 if Clamping is off, or the nearest node point if Clamping is on. Using the legacy methods for adding points (which do not have Sharpness and Midpoint parameters) will default to Midpoint = 0.5 (halfway between the control points) and Sharpness = 0.0 (linear).

To create an instance of class vtkPiecewiseFunction, simply invoke its constructor as follows

```
obj = vtkPiecewiseFunction
```

31.143.2 Methods

The class vtkPiecewiseFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPiecewiseFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPiecewiseFunction = obj.NewInstance ()`
- `vtkPiecewiseFunction = obj.SafeDownCast (vtkObject o)`
- `obj.DeepCopy (vtkDataObject f)`
- `obj.ShallowCopy (vtkDataObject f)`
- `int = obj.GetDataObjectType ()` - Return what type of dataset this is.
- `int = obj.GetSize ()` - Get the number of points used to specify the function
- `int = obj.AddPoint (double x, double y)` - Add/Remove points to/from the function. If a duplicate point is added then the function value is changed at that location. Return the index of the point (0 based), or -1 on error.
- `int = obj.AddPoint (double x, double y, double midpoint, double sharpness)` - Add/Remove points to/from the function. If a duplicate point is added then the function value is changed at that location. Return the index of the point (0 based), or -1 on error.
- `int = obj.RemovePoint (double x)` - Add/Remove points to/from the function. If a duplicate point is added then the function value is changed at that location. Return the index of the point (0 based), or -1 on error.
- `obj.RemoveAllPoints ()` - Removes all points from the function.
- `obj.AddSegment (double x1, double y1, double x2, double y2)` - Add a line segment to the function. All points defined between the two points specified are removed from the function. This is a legacy method that does not allow the specification of the sharpness and midpoint values for the two nodes.
- `double = obj.GetValue (double x)` - Returns the value of the function at the specified location using the specified interpolation.
- `int = obj.GetNodeValue (int index, double val[4])` - For the node specified by index, set/get the location (X), value (Y), midpoint, and sharpness values at the node.
- `int = obj.SetNodeValue (int index, double val[4])` - For the node specified by index, set/get the location (X), value (Y), midpoint, and sharpness values at the node.

- `obj.FillFromDataPointer (int , double )` - Returns a pointer to the data stored in the table. Fills from a pointer to data stored in a similar table. These are legacy methods which will be maintained for compatibility - however, note that the vtkPiecewiseFunction no longer stores the nodes in a double array internally.

- `double = obj. GetRange ()` - Returns the min and max node locations of the function.

- `int = obj.AdjustRange (double range[2])` - Remove all points out of the new range, and make sure there is a point at each end of that range. Return 1 on success, 0 otherwise.

- `obj.GetTable (double x1, double x2, int size, float table, int stride)` - Fills in an array of function values evaluated at regular intervals. Parameter "stride" is used to step through the output "table".

- `obj.GetTable (double x1, double x2, int size, double table, int stride)` - Fills in an array of function values evaluated at regular intervals. Parameter "stride" is used to step through the output "table".

- `obj.BuildFunctionFromTable (double x1, double x2, int size, double table, int stride)` - Constructs a piecewise function from a table. Function range is is set to [x1, x2], function size is set to size, and function points are regularly spaced between x1 and x2. Parameter "stride" is is step through the input table.

- `obj.SetClamping (int )` - When zero range clamping is Off, GetValue() returns 0.0 when a value is requested outside of the points specified. When zero range clamping is On, GetValue() returns the value at the value at the lowest point for a request below all points specified and returns the value at the highest point for a request above all points specified. On is the default.

- `int = obj.GetClamping ()` - When zero range clamping is Off, GetValue() returns 0.0 when a value is requested outside of the points specified. When zero range clamping is On, GetValue() returns the value at the value at the lowest point for a request below all points specified and returns the value at the highest point for a request above all points specified. On is the default.

- `obj.ClampingOn ()` - When zero range clamping is Off, GetValue() returns 0.0 when a value is requested outside of the points specified. When zero range clamping is On, GetValue() returns the value at the value at the lowest point for a request below all points specified and returns the value at the highest point for a request above all points specified. On is the default.

- `obj.ClampingOff ()` - When zero range clamping is Off, GetValue() returns 0.0 when a value is requested outside of the points specified. When zero range clamping is On, GetValue() returns the value at the value at the lowest point for a request below all points specified and returns the value at the highest point for a request above all points specified. On is the default.

- `string = obj.GetType ()` - Return the type of function: Function Types: 0 : Constant (No change in slope between end points) 1 : NonDecreasing (Always increasing or zero slope) 2 : NonIncreasing (Always decreasing or zero slope) 3 : Varied (Contains both decreasing and increasing slopes)

- `double = obj.GetFirstNonZeroValue ()` - Returns the first point location which precedes a non-zero segment of the function. Note that the value at this point may be zero.

- `obj.Initialize ()` - Clears out the current function. A newly created vtkPiecewiseFunction is alreay initialized, so there is no need to call this method which in turn simply calls RemoveAllPoints()

- `obj.SetAllowDuplicateScalars (int )` - Toggle whether to allow duplicate scalar values in the piecewise function (off by default).

- `int = obj.GetAllowDuplicateScalars ()` - Toggle whether to allow duplicate scalar values in the piecewise function (off by default).

- `obj.AllowDuplicateScalarsOn ()` - Toggle whether to allow duplicate scalar values in the piecewise function (off by default).
- `obj.AllowDuplicateScalarsOff ()` - Toggle whether to allow duplicate scalar values in the piecewise function (off by default).

31.144 vtkPiecewiseFunctionAlgorithm

31.144.1 Usage

To create an instance of class vtkPiecewiseFunctionAlgorithm, simply invoke its constructor as follows

```
obj = vtkPiecewiseFunctionAlgorithm
```

31.144.2 Methods

The class vtkPiecewiseFunctionAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPiecewiseFunctionAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPiecewiseFunctionAlgorithm = obj.NewInstance ()`
- `vtkPiecewiseFunctionAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkDataObject = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput ()`
- `vtkDataObject = obj.GetInput (int port)`
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.145 vtkPiecewiseFunctionShiftScale

31.145.1 Usage

To create an instance of class vtkPiecewiseFunctionShiftScale, simply invoke its constructor as follows

```
obj = vtkPiecewiseFunctionShiftScale
```

31.145.2 Methods

The class vtkPiecewiseFunctionShiftScale has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPiecewiseFunctionShiftScale class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPiecewiseFunctionShiftScale = obj.NewInstance ()`
- `vtkPiecewiseFunctionShiftScale = obj.SafeDownCast (vtkObject o)`
- `obj.SetPositionShift (double )`
- `obj.SetPositionScale (double )`
- `obj.SetValueShift (double )`
- `obj.SetValueScale (double )`
- `double = obj.GetPositionShift ()`
- `double = obj.GetPositionScale ()`
- `double = obj.GetValueShift ()`
- `double = obj.GetValueScale ()`

31.146 vtkPixel

31.146.1 Usage

vtkPixel is a concrete implementation of vtkCell to represent a 2D orthogonal quadrilateral. Unlike vtkQuad, the corners are at right angles, and aligned along x-y-z coordinate axes leading to large increases in computational efficiency.

To create an instance of class vtkPixel, simply invoke its constructor as follows

```
obj = vtkPixel
```

31.146.2 Methods

The class vtkPixel has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPixel class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPixel = obj.NewInstance ()`

- `vtkPixel = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the triangle in parametric coordinates.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.InterpolateFunctions (double pcoords[3], double weights[4])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[8])`

31.147 vtkPlanesIntersection

31.147.1 Usage

A subclass of vtkPlanes, this class determines whether it intersects an axis aligned box. This is motivated by the need to intersect the axis aligned region of a spacial decomposition of volume data with various other regions. It uses the algorithm from Graphics Gems IV, page 81.

.SECTION Caveat An instance of vtkPlanes can be redefined by changing the planes, but this subclass then will not know if the region vertices are up to date. (Region vertices can be specified in SetRegionVertices or computed by the subclass.) So Delete and recreate if you want to change the set of planes.

To create an instance of class vtkPlanesIntersection, simply invoke its constructor as follows

```
obj = vtkPlanesIntersection
```

31.147.2 Methods

The class vtkPlanesIntersection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPlanesIntersection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkPlanesIntersection = obj.NewInstance ()`
- `vtkPlanesIntersection = obj.SafeDownCast (vtkObject o)`
- `obj.SetRegionVertices (vtkPoints pts)`
- `obj.SetRegionVertices (double v, int nvertices)`
- `int = obj.GetNumRegionVertices ()`
- `int = obj.GetRegionVertices (double v, int nvertices)`
- `int = obj.IntersectsRegion (vtkPoints R)`

31.148 vtkPointData

31.148.1 Usage

vtkPointData is a class that is used to represent and manipulate point attribute data (e.g., scalars, vectors, normals, texture coordinates, etc.) Most of the functionality is handled by vtkDataSetAttributes

To create an instance of class vtkPointData, simply invoke its constructor as follows

```
obj = vtkPointData
```

31.148.2 Methods

The class vtkPointData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointData = obj.NewInstance ()`
- `vtkPointData = obj.SafeDownCast (vtkObject o)`
- `obj.NullPoint (vtkIdType ptId)`

31.149 vtkPointLocator

31.149.1 Usage

vtkPointLocator is a spatial search object to quickly locate points in 3D. vtkPointLocator works by dividing a specified region of space into a regular array of "rectangular" buckets, and then keeping a list of points that lie in each bucket. Typical operation involves giving a position in 3D and finding the closest point.

vtkPointLocator has two distinct methods of interaction. In the first method, you supply it with a dataset, and it operates on the points in the dataset. In the second method, you supply it with an array of points, and the object operates on the array.

To create an instance of class vtkPointLocator, simply invoke its constructor as follows

```
obj = vtkPointLocator
```

31.149.2 Methods

The class vtkPointLocator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointLocator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointLocator = obj.NewInstance ()`
- `vtkPointLocator = obj.SafeDownCast (vtkObject o)`
- `obj.SetDivisions (int , int , int )` - Set the number of divisions in x-y-z directions.
- `obj.SetDivisions (int  a[3])` - Set the number of divisions in x-y-z directions.
- `int = obj. GetDivisions ()` - Set the number of divisions in x-y-z directions.
- `obj.SetNumberOfPointsPerBucket (int )` - Specify the average number of points in each bucket.
- `int = obj.GetNumberOfPointsPerBucketMinValue ()` - Specify the average number of points in each bucket.
- `int = obj.GetNumberOfPointsPerBucketMaxValue ()` - Specify the average number of points in each bucket.
- `int = obj.GetNumberOfPointsPerBucket ()` - Specify the average number of points in each bucket.
- `vtkIdType = obj.FindClosestPoint (double x[3])` - Given a position x, return the id of the point closest to it. Alternative method requires separate x-y-z values. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.
- `int = obj.InitPointInsertion (vtkPoints newPts, double bounds[6])` - Initialize the point insertion process. The newPts is an object representing point coordinates into which incremental insertion methods place their data. Bounds are the box that the points lie in. Not thread safe.
- `int = obj.InitPointInsertion (vtkPoints newPts, double bounds[6], vtkIdType estSize)` - Initialize the point insertion process. The newPts is an object representing point coordinates into which incremental insertion methods place their data. Bounds are the box that the points lie in. Not thread safe.
- `obj.InsertPoint (vtkIdType ptId, double x[3])` - Incrementally insert a point into search structure with a particular index value. You should use the method IsInsertedPoint() to see whether this point has already been inserted (that is, if you desire to prevent duplicate points). Before using this method you must make sure that newPts have been supplied, the bounds has been set properly, and that divs are properly set. (See InitPointInsertion().) Not thread safe.
- `vtkIdType = obj.InsertNextPoint (double x[3])` - Incrementally insert a point into search structure. The method returns the insertion location (i.e., point id). You should use the method IsInsertedPoint() to see whether this point has already been inserted (that is, if you desire to prevent duplicate points). Before using this method you must make sure that newPts have been supplied, the bounds has been set properly, and that divs are properly set. (See InitPointInsertion().) Not thread safe.
- `vtkIdType = obj.IsInsertedPoint (double x, double y, double z)` - Determine whether point given by x[3] has been inserted into points list. Return id of previously inserted point if this is true, otherwise return -1. This method is thread safe.

- `vtkIdType = obj.IsInsertedPoint (double x[3])` - Determine whether point given by x[3] has been inserted into points list. Return id of previously inserted point if this is true, otherwise return -1. This method is thread safe.

- `vtkIdType = obj.FindClosestInsertedPoint (double x[3])` - Given a position x, return the id of the point closest to it. This method is used when performing incremental point insertion. Note that -1 indicates that no point was found. This method is thread safe if BuildLocator() is directly or indirectly called from a single thread first.

- `obj.FindClosestNPoints (int N, double x[3], vtkIdList result)` - Find the closest N points to a position. This returns the closest N points to a position. A faster method could be created that returned N close points to a position, but necessarily the exact N closest. The returned points are sorted from closest to farthest. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.

- `obj.FindDistributedPoints (int N, double x[3], vtkIdList result, int M)` - Find the closest points to a position such that each octant of space around the position contains at least N points. Loosely limit the search to a maximum number of points evaluated, M. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.

- `obj.FindDistributedPoints (int N, double x, double y, double z, vtkIdList result, int M)` - Find the closest points to a position such that each octant of space around the position contains at least N points. Loosely limit the search to a maximum number of points evaluated, M. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.

- `obj.FindPointsWithinRadius (double R, double x[3], vtkIdList result)` - Find all points within a specified radius R of position x. The result is not sorted in any specific manner. These methods are thread safe if BuildLocator() is directly or indirectly called from a single thread first.

- `vtkIdList = obj.GetPointsInBucket (double x[3], int ijk[3])` - Given a position x, return the list of points in the bucket that contains the point. It is possible that NULL is returned. The user provides an ijk array that is the indices into the locator. This method is thread safe.

- `vtkPoints = obj.GetPoints ()` - Provide an accessor to the points.

- `obj.Initialize ()` - See vtkLocator interface documentation. These methods are not thread safe.

- `obj.FreeSearchStructure ()` - See vtkLocator interface documentation. These methods are not thread safe.

- `obj.BuildLocator ()` - See vtkLocator interface documentation. These methods are not thread safe.

- `obj.GenerateRepresentation (int level, vtkPolyData pd)` - See vtkLocator interface documentation. These methods are not thread safe.

31.150 vtkPointSet

31.150.1 Usage

vtkPointSet is an abstract class that specifies the interface for datasets that explicitly use "point" arrays to represent geometry. For example, vtkPolyData and vtkUnstructuredGrid require point arrays to specify point position, while vtkStructuredPoints generates point positions implicitly.

To create an instance of class vtkPointSet, simply invoke its constructor as follows

```
obj = vtkPointSet
```

31.150.2 Methods

The class vtkPointSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointSet = obj.NewInstance ()`
- `vtkPointSet = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Reset to an empty state and free any memory.
- `obj.CopyStructure (vtkDataSet pd)` - Copy the geometric structure of an input point set object.
- `vtkIdType = obj.GetNumberOfPoints ()` - See vtkDataSet for additional information.
- `double = obj.GetPoint (vtkIdType ptId)` - See vtkDataSet for additional information.
- `obj.GetPoint (vtkIdType ptId, double x[3])` - See vtkDataSet for additional information.
- `vtkIdType = obj.FindPoint (double x[3])` - See vtkDataSet for additional information.
- `vtkIdType = obj.FindPoint (double x, double y, double z)` - See vtkDataSet for additional information.
- `long = obj.GetMTime ()` - Get MTime which also considers its vtkPoints MTime.
- `obj.ComputeBounds ()` - Compute the (X, Y, Z) bounds of the data.
- `obj.Squeeze ()` - Reclaim any unused memory.
- `obj.SetPoints (vtkPoints )` - Specify point array to define point coordinates.
- `vtkPoints = obj.GetPoints ()` - Specify point array to define point coordinates.
- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value). THIS METHOD IS THREAD SAFE.
- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.

31.151 vtkPointSetAlgorithm

31.151.1 Usage

vtkPointSetAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline architecture. Ther are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this classes contstructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be PointSet. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into seperate functions such as RequestDataObject RequestData and ExecuteInformation. The default implementation of RequestDataObject will create an output data of the same type as the input.

To create an instance of class vtkPointSetAlgorithm, simply invoke its constructor as follows

```
obj = vtkPointSetAlgorithm
```

31.151.2 Methods

The class vtkPointSetAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointSetAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointSetAlgorithm = obj.NewInstance ()`
- `vtkPointSetAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkPointSet = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkPointSet = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `vtkPolyData = obj.GetPolyDataOutput ()` - Get the output as vtkPolyData.
- `vtkStructuredGrid = obj.GetStructuredGridOutput ()` - Get the output as vtkStructuredGrid.
- `vtkUnstructuredGrid = obj.GetUnstructuredGridOutput ()` - Get the output as vtkUnstructuredGrid.
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (vtkPointSet )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkPointSet )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (vtkPointSet )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkPointSet )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

- `vtkDataObject = obj.GetInput ()`

31.152 vtkPointSetSource

31.152.1 Usage

vtkPointSetSource is an abstract class whose subclasses generate pointdata.

To create an instance of class vtkPointSetSource, simply invoke its constructor as follows

```
obj = vtkPointSetSource
```

31.152.2 Methods

The class vtkPointSetSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointSetSource class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkPointSetSource = obj.NewInstance ()`

- `vtkPointSetSource = obj.SafeDownCast (vtkObject o)`

- `vtkPointSet = obj.GetOutput ()` - Get the output of this source.

- `vtkPointSet = obj.GetOutput (int idx)` - Get the output of this source.

- `obj.SetOutput (vtkPointSet output)` - Get the output of this source.

31.153 vtkPointSetToPointSetFilter

31.153.1 Usage

vtkPointSetToPointSetFilter is an abstract filter class whose subclasses take as input a point set and generates a point set on output. At a minimum, the concrete subclasses of vtkPointSetToPointSetFilter modify their point coordinates. They never modify their topological form, however.

This is an abstract filter type. What that means is that the output of the filter is an abstract type (i.e., vtkPointSet), no matter what the input of the filter is. This can cause problems connecting together filters due to the change in dataset type. (For example, in a series of filters processing vtkPolyData, when a vtkPointSetToPointSetFilter or subclass is introduced into the pipeline, if the filter downstream of it takes vtkPolyData as input, the pipeline connection cannot be made.) To get around this problem, use one of the convenience methods to return a concrete type (e.g., vtkGetPolyDataOutput(), GetStructuredGridOutput(), etc.).

To create an instance of class vtkPointSetToPointSetFilter, simply invoke its constructor as follows

```
obj = vtkPointSetToPointSetFilter
```

31.153.2 Methods

The class vtkPointSetToPointSetFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointSetToPointSetFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointSetToPointSetFilter = obj.NewInstance ()`
- `vtkPointSetToPointSetFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkPointSet input)` - Specify the input data or filter.
- `vtkPointSet = obj.GetInput ()` - Get the input data or filter.
- `vtkPointSet = obj.GetOutput ()` - Get the output of this filter. If output is NULL, then input hasn't been set, which is necessary for abstract filter objects.
- `vtkPointSet = obj.GetOutput (int idx)` - Get the output as vtkPolyData. Performs run-time checking.
- `vtkPolyData = obj.GetPolyDataOutput ()` - Get the output as vtkPolyData. Performs run-time checking.
- `vtkStructuredGrid = obj.GetStructuredGridOutput ()` - Get the output as vtkStructuredGrid. Performs run-time checking.
- `vtkUnstructuredGrid = obj.GetUnstructuredGridOutput ()` - Get the output as vtkUnstructured-Grid. Performs run-time checking.
- `obj.ComputeInputUpdateExtents (vtkDataObject output)` - By default copy the output update extent to the input

31.154 vtkPointsProjectedHull

31.154.1 Usage

a subclass of vtkPoints, it maintains the counter clockwise convex hull of the points (projected orthogonally in the three coordinate directions) and has a method to test for intersection of that hull with an axis aligned rectangle. This is used for intersection tests of 3D volumes.

To create an instance of class vtkPointsProjectedHull, simply invoke its constructor as follows

```
obj = vtkPointsProjectedHull
```

31.154.2 Methods

The class vtkPointsProjectedHull has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointsProjectedHull class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointsProjectedHull = obj.NewInstance ()`

- vtkPointsProjectedHull = obj.SafeDownCast (vtkObject o)
- int = obj.RectangleIntersectionX (vtkPoints R)
- int = obj.RectangleIntersectionX (float ymin, float ymax, float zmin, float zmax)
- int = obj.RectangleIntersectionX (double ymin, double ymax, double zmin, double zmax)
- int = obj.RectangleIntersectionY (vtkPoints R)
- int = obj.RectangleIntersectionY (float zmin, float zmax, float xmin, float xmax)
- int = obj.RectangleIntersectionY (double zmin, double zmax, double xmin, double xmax)
- int = obj.RectangleIntersectionZ (vtkPoints R)
- int = obj.RectangleIntersectionZ (float xmin, float xmax, float ymin, float ymax)
- int = obj.RectangleIntersectionZ (double xmin, double xmax, double ymin, double ymax)
- int = obj.GetCCWHullX (float pts, int len)
- int = obj.GetCCWHullX (double pts, int len)
- int = obj.GetCCWHullY (float pts, int len)
- int = obj.GetCCWHullY (double pts, int len)
- int = obj.GetCCWHullZ (float pts, int len)
- int = obj.GetCCWHullZ (double pts, int len)
- int = obj.GetSizeCCWHullX ()
- int = obj.GetSizeCCWHullY ()
- int = obj.GetSizeCCWHullZ ()
- obj.Initialize ()
- obj.Reset ()
- obj.Update ()

31.155 vtkPolyData

31.155.1 Usage

vtkPolyData is a data object that is a concrete implementation of vtkDataSet. vtkPolyData represents a geometric structure consisting of vertices, lines, polygons, and/or triangle strips. Point and cell attribute values (e.g., scalars, vectors, etc.) also are represented.

The actual cell types (vtkCellType.h) supported by vtkPolyData are: vtkVertex, vtkPolyVertex, vtkLine, vtkPolyLine, vtkTriangle, vtkQuad, vtkPolygon, and vtkTriangleStrip.

One important feature of vtkPolyData objects is that special traversal and data manipulation methods are available to process data. These methods are generally more efficient than vtkDataSet methods and should be used whenever possible. For example, traversing the cells in a dataset we would use GetCell(). To traverse cells with vtkPolyData we would retrieve the cell array object representing polygons (for example using GetPolys()) and then use vtkCellArray's InitTraversal() and GetNextCell() methods.

To create an instance of class vtkPolyData, simply invoke its constructor as follows

```
obj = vtkPolyData
```

31.155.2 Methods

The class vtkPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyData = obj.NewInstance ()`
- `vtkPolyData = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Copy the geometric and topological structure of an input poly data object.
- `obj.CopyStructure (vtkDataSet ds)` - Copy the geometric and topological structure of an input poly data object.
- `vtkIdType = obj.GetNumberOfCells ()` - Standard vtkDataSet interface.
- `vtkCell = obj.GetCell (vtkIdType cellId)` - Standard vtkDataSet interface.
- `obj.GetCell (vtkIdType cellId, vtkGenericCell cell)` - Standard vtkDataSet interface.
- `int = obj.GetCellType (vtkIdType cellId)` - Standard vtkDataSet interface.
- `obj.GetCellBounds (vtkIdType cellId, double bounds[6])` - Standard vtkDataSet interface.
- `obj.GetCellNeighbors (vtkIdType cellId, vtkIdList ptIds, vtkIdList cellIds)` - Standard vtkDataSet interface.
- `obj.CopyCells (vtkPolyData pd, vtkIdList idList, vtkPointLocator locatorNULL)` - Copy cells listed in idList from pd, including points, point data, and cell data. This method assumes that point and cell data have been allocated. If you pass in a point locator, then the points won't be duplicated in the output.
- `obj.GetCellPoints (vtkIdType cellId, vtkIdList ptIds)` - Copy a cells point ids into list provided. (Less efficient.)
- `obj.GetPointCells (vtkIdType ptId, vtkIdList cellIds)` - Efficient method to obtain cells using a particular point. Make sure that routine BuildLinks() has been called.
- `obj.ComputeBounds ()` - Compute the (X, Y, Z) bounds of the data.
- `obj.Squeeze ()` - Recover extra allocated memory when creating data whose initial size is unknown. Examples include using the InsertNextCell() method, or when using the CellArray::EstimateSize() method to create vertices, lines, polygons, or triangle strips.
- `int = obj.GetMaxCellSize ()` - Return the maximum cell size in this poly data.
- `obj.SetVerts (vtkCellArray v)` - Set the cell array defining vertices.
- `vtkCellArray = obj.GetVerts ()` - Get the cell array defining vertices. If there are no vertices, an empty array will be returned (convenience to simplify traversal).
- `obj.SetLines (vtkCellArray l)` - Set the cell array defining lines.
- `vtkCellArray = obj.GetLines ()` - Get the cell array defining lines. If there are no lines, an empty array will be returned (convenience to simplify traversal).

- `obj.SetPolys (vtkCellArray p)` - Set the cell array defining polygons.
- `vtkCellArray = obj.GetPolys ()` - Get the cell array defining polygons. If there are no polygons, an empty array will be returned (convenience to simplify traversal).
- `obj.SetStrips (vtkCellArray s)` - Set the cell array defining triangle strips.
- `vtkCellArray = obj.GetStrips ()` - Get the cell array defining triangle strips. If there are no triangle strips, an empty array will be returned (convenience to simplify traversal).
- `vtkIdType = obj.GetNumberOfVerts ()` - Return the number of primitives of a particular type held..
- `vtkIdType = obj.GetNumberOfLines ()` - Return the number of primitives of a particular type held..
- `vtkIdType = obj.GetNumberOfPolys ()` - Return the number of primitives of a particular type held..
- `vtkIdType = obj.GetNumberOfStrips ()` - Return the number of primitives of a particular type held..
- `obj.Allocate (vtkIdType numCells, int extSize)` - Method allocates initial storage for vertex, line, polygon, and triangle strip arrays. Use this method before the method PolyData::InsertNextCell(). (Or, provide vertex, line, polygon, and triangle strip cell arrays.)
- `obj.Allocate (vtkPolyData inPolyData, vtkIdType numCells, int extSize)` - Similar to the method above, this method allocates initial storage for vertex, line, polygon, and triangle strip arrays. It does this more intelligently, examining the supplied inPolyData to determine whether to allocate the verts, lines, polys, and strips arrays. (These arrays are allocated only if there is data in the corresponding arrays in the inPolyData.) Caution: if the inPolyData has no verts, and after allocating with this method an PolyData::InsertNextCell() is invoked where a vertex is inserted, bad things will happen.
- `int = obj.InsertNextCell (int type, vtkIdList pts)` - Insert a cell of type VTK_VERTEX, VTK_POLY_VERTEX, VTK_LINE, VTK_POLY_LINE, VTK_TRIANGLE, VTK_QUAD, VTK_POLYGON, or VTK_TRIANGLE_STRIP. Make sure that the PolyData::Allocate() function has been called first or that vertex, line, polygon, and triangle strip arrays have been supplied. Note: will also insert VTK_PIXEL, but converts it to VTK_QUAD.
- `obj.Reset ()` - Begin inserting data all over again. Memory is not freed but otherwise objects are returned to their initial state.
- `obj.BuildCells ()` - Create data structure that allows random access of cells.
- `obj.BuildLinks (int initialSize)` - Create upward links from points to cells that use each point. Enables topologically complex queries. Normally the links array is allocated based on the number of points in the vtkPolyData. The optional initialSize parameter can be used to allocate a larger size initially.
- `obj.DeleteCells ()` - Release data structure that allows random access of the cells. This must be done before a 2nd call to BuildLinks(). DeleteCells implicitly deletes the links as well since they are no longer valid.
- `obj.DeleteLinks ()` - Release the upward links from point to cells that use each point.
- `obj.GetCellEdgeNeighbors (vtkIdType cellId, vtkIdType p1, vtkIdType p2, vtkIdList cellIds)` - Get the neighbors at an edge. More efficient than the general GetCellNeighbors(). Assumes links have been built (with BuildLinks()), and looks specifically for edge neighbors.
- `int = obj.IsTriangle (int v1, int v2, int v3)` - Given three vertices, determine whether it's a triangle. Make sure BuildLinks() has been called first.

- `int = obj.IsEdge (vtkIdType p1, vtkIdType p2)` - Determine whether two points form an edge. If they do, return non-zero. By definition PolyVertex and PolyLine have no edges since 1-dimensional edges are only found on cells 2D and higher. Edges are defined as 1-D boundary entities to cells. Make sure BuildLinks() has been called first.
- `int = obj.IsPointUsedByCell (vtkIdType ptId, vtkIdType cellId)` - Determine whether a point is used by a particular cell. If it is, return non-zero. Make sure BuildCells() has been called first.
- `obj.ReplaceCellPoint (vtkIdType cellId, vtkIdType oldPtId, vtkIdType newPtId)` - Replace a point in the cell connectivity list with a different point.
- `obj.ReverseCell (vtkIdType cellId)` - Reverse the order of point ids defining the cell.
- `obj.DeletePoint (vtkIdType ptId)` - Mark a point/cell as deleted from this vtkPolyData.
- `obj.DeleteCell (vtkIdType cellId)` - Mark a point/cell as deleted from this vtkPolyData.
- `obj.RemoveDeletedCells ()` - The cells marked by calls to DeleteCell are stored in the Cell Array VTK_EMPTY_CELL, but they still exist in the polys array. Calling RemoveDeletedCells will travers the poly array and remove/compact the cell array as well as any cell data thus truly removing the cells from the polydata object. WARNING. This only handles the polys at the moment
- `int = obj.InsertNextLinkedPoint (int numLinks)` - Add a point to the cell data structure (after cell pointers have been built). This method adds the point and then allocates memory for the links to the cells. (To use this method, make sure points are available and BuildLinks() has been invoked.) Of the two methods below, one inserts a point coordinate and the other just makes room for cell links.
- `int = obj.InsertNextLinkedPoint (double x[3], int numLinks)` - Add a point to the cell data structure (after cell pointers have been built). This method adds the point and then allocates memory for the links to the cells. (To use this method, make sure points are available and BuildLinks() has been invoked.) Of the two methods below, one inserts a point coordinate and the other just makes room for cell links.
- `obj.RemoveCellReference (vtkIdType cellId)` - Remove all references to cell in cell structure. This means the links from the cell's points to the cell are deleted. Memory is not reclaimed. Use the method ResizeCellList() to resize the link list from a point to its using cells. (This operator assumes BuildLinks() has been called.)
- `obj.AddCellReference (vtkIdType cellId)` - Add references to cell in cell structure. This means the links from the cell's points to the cell are modified. Memory is not extended. Use the method ResizeCellList() to resize the link list from a point to its using cells. (This operator assumes BuildLinks() has been called.)
- `obj.RemoveReferenceToCell (vtkIdType ptId, vtkIdType cellId)` - Remove a reference to a cell in a particular point's link list. You may also consider using RemoveCellReference() to remove the references from all the cell's points to the cell. This operator does not reallocate memory; use the operator ResizeCellList() to do this if necessary.
- `obj.AddReferenceToCell (vtkIdType ptId, vtkIdType cellId)` - Add a reference to a cell in a particular point's link list. (You may also consider using AddCellReference() to add the references from all the cell's points to the cell.) This operator does not realloc memory; use the operator ResizeCellList() to do this if necessary.
- `obj.ResizeCellList (vtkIdType ptId, int size)` - Resize the list of cells using a particular point. (This operator assumes that BuildLinks() has been called.)
- `obj.Initialize ()` - Restore object to initial state. Release memory back to system.
- `int = obj.GetUpdateExtent ()` - We need this here to avoid hiding superclass method

- `obj.GetUpdateExtent (int extent[6])` - We need this here to avoid hiding superclass method
- `int = obj.GetPiece ()` - Get the piece and the number of pieces. Similar to extent in 3D.
- `int = obj.GetNumberOfPieces ()` - Get the piece and the number of pieces. Similar to extent in 3D.
- `int = obj.GetGhostLevel ()` - Get the ghost level.
- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value). THIS METHOD IS THREAD SAFE.
- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.RemoveGhostCells (int level)` - This method will remove any cell that has a ghost level array value greater or equal to level. It does not remove unused points (yet).
- `int = obj.GetScalarFieldCriticalIndex (vtkIdType pointId, vtkDataArray scalarField)` - Scalar field critical point classification (for manifold 2D meshes). Reference: J. Milnor "Morse Theory", Princeton University Press, 1963.

  Given a pointId and an attribute representing a scalar field, this member returns the index of the critical point: vtkPolyData::MINIMUM (index 0): local minimum; vtkPolyData::SADDLE (index 1): local saddle; vtkPolyData::MAXIMUM (index 2): local maximum.

  Other returned values are: vtkPolyData::REGULAR_POINT: regular point (the gradient does not vanish); vtkPolyData::ERR_NON_MANIFOLD_STAR: the star of the considered vertex is not manifold (could not evaluate the index) vtkPolyData::ERR_INCORRECT_FIELD: the number of entries in the scalar field array is different form the number of vertices in the mesh. vtkPolyData::ERR_NO_SUCH_FIELD: the specified scalar field does not exist.
- `int = obj.GetScalarFieldCriticalIndex (vtkIdType pointId, int fieldId)` - Scalar field critical point classification (for manifold 2D meshes). Reference: J. Milnor "Morse Theory", Princeton University Press, 1963.

  Given a pointId and an attribute representing a scalar field, this member returns the index of the critical point: vtkPolyData::MINIMUM (index 0): local minimum; vtkPolyData::SADDLE (index 1): local saddle; vtkPolyData::MAXIMUM (index 2): local maximum.

  Other returned values are: vtkPolyData::REGULAR_POINT: regular point (the gradient does not vanish); vtkPolyData::ERR_NON_MANIFOLD_STAR: the star of the considered vertex is not manifold (could not evaluate the index) vtkPolyData::ERR_INCORRECT_FIELD: the number of entries in the scalar field array is different form the number of vertices in the mesh. vtkPolyData::ERR_NO_SUCH_FIELD: the specified scalar field does not exist.
- `int = obj.GetScalarFieldCriticalIndex (vtkIdType pointId, string fieldName)` - Scalar field critical point classification (for manifold 2D meshes). Reference: J. Milnor "Morse Theory", Princeton University Press, 1963.

  Given a pointId and an attribute representing a scalar field, this member returns the index of the critical point: vtkPolyData::MINIMUM (index 0): local minimum; vtkPolyData::SADDLE (index 1): local saddle; vtkPolyData::MAXIMUM (index 2): local maximum.

  Other returned values are: vtkPolyData::REGULAR_POINT: regular point (the gradient does not vanish); vtkPolyData::ERR_NON_MANIFOLD_STAR: the star of the considered vertex is not manifold (could not evaluate the index) vtkPolyData::ERR_INCORRECT_FIELD: the number of entries in the scalar field array is different form the number of vertices in the mesh. vtkPolyData::ERR_NO_SUCH_FIELD: the specified scalar field does not exist.

31.156 vtkPolyDataAlgorithm

31.156.1 Usage

To create an instance of class vtkPolyDataAlgorithm, simply invoke its constructor as follows

```
obj = vtkPolyDataAlgorithm
```

31.156.2 Methods

The class vtkPolyDataAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataAlgorithm = obj.NewInstance ()`
- `vtkPolyDataAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkPolyData = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkPolyData = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput ()`
- `vtkDataObject = obj.GetInput (int port)`
- `vtkPolyData = obj.GetPolyDataInput (int port)`
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.157 vtkPolyDataCollection

31.157.1 Usage

vtkPolyDataCollection is an object that creates and manipulates lists of datasets of type vtkPolyData.

To create an instance of class vtkPolyDataCollection, simply invoke its constructor as follows

```
obj = vtkPolyDataCollection
```

31.157.2 Methods

The class vtkPolyDataCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataCollection = obj.NewInstance ()`
- `vtkPolyDataCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkPolyData pd)` - Get the next poly data in the list.
- `vtkPolyData = obj.GetNextItem ()` - Get the next poly data in the list.

31.158 vtkPolyDataSource

31.158.1 Usage

vtkPolyDataSource is an abstract class whose subclasses generate polygonal data.

To create an instance of class vtkPolyDataSource, simply invoke its constructor as follows

```
obj = vtkPolyDataSource
```

31.158.2 Methods

The class vtkPolyDataSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataSource = obj.NewInstance ()`
- `vtkPolyDataSource = obj.SafeDownCast (vtkObject o)`
- `vtkPolyData = obj.GetOutput ()` - Get the output of this source.
- `vtkPolyData = obj.GetOutput (int idx)` - Get the output of this source.
- `obj.SetOutput (vtkPolyData output)` - Get the output of this source.

31.159 vtkPolyDataToPolyDataFilter

31.159.1 Usage

vtkPolyDataToPolyDataFilter is an abstract filter class whose subclasses take as input polygonal data and generate polygonal data on output. .SECTION Warning This used to be the parent class for most polydata filter in VTK4.x, now this role has been replaced by vtkPolyDataAlgorithm. You should consider using vtkPolyDataAlgorithm instead of this class, when writing filter for VTK5 and above. This class was kept to ensure full backward compatibility.

To create an instance of class vtkPolyDataToPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkPolyDataToPolyDataFilter
```

31.159.2 Methods

The class vtkPolyDataToPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataToPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataToPolyDataFilter = obj.NewInstance ()`
- `vtkPolyDataToPolyDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkPolyData input)` - Set / get the input data or filter.
- `vtkPolyData = obj.GetInput ()` - Set / get the input data or filter.

31.160 vtkPolygon

31.160.1 Usage

vtkPolygon is a concrete implementation of vtkCell to represent a 2D n-sided polygon. The polygons cannot have any internal holes, and cannot self-intersect. Define the polygon with n-points ordered in the counter-clockwise direction; do not repeat the last point.

To create an instance of class vtkPolygon, simply invoke its constructor as follows

```
obj = vtkPolygon
```

31.160.2 Methods

The class vtkPolygon has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolygon class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolygon = obj.NewInstance ()`
- `vtkPolygon = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.

- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `int = obj.IsPrimaryCell ()` - Compute the area of a polygon. This is a convenience function which simply calls static double ComputeArea(vtkPoints *p, vtkIdType numPts, vtkIdType *pts, double normal[3]); with the appropriate parameters from the instantiated vtkPolygon.
- `double = obj.ComputeArea ()` - Compute the area of a polygon. This is a convenience function which simply calls static double ComputeArea(vtkPoints *p, vtkIdType numPts, vtkIdType *pts, double normal[3]); with the appropriate parameters from the instantiated vtkPolygon.
- `obj.InterpolateFunctions (double pcoords[3], double sf)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.ComputeWeights (double x[3], double weights)` - Compute interpolation weights using 1/r**2 normalized sum. @deprecated Replaced by vtkPolygon::InterpolateFunctions as of VTK 5.2
- `int = obj.Triangulate (vtkIdList outTris)` - Triangulate this polygon. The user must provide the vtkIdList outTris. On output, the outTris list contains the ids of the points defining the triangulation. The ids are ordered into groups of three: each three-group defines one triangle.
- `int = obj.NonDegenerateTriangulate (vtkIdList outTris)` - Same as Triangulate(vtkIdList *outTris) but with a first pass to split the polygon into non-degenerate polygons.

31.161 vtkPolyLine

31.161.1 Usage

vtkPolyLine is a concrete implementation of vtkCell to represent a set of 1D lines.

To create an instance of class vtkPolyLine, simply invoke its constructor as follows

```
obj = vtkPolyLine
```

31.161.2 Methods

The class vtkPolyLine has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyLine class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyLine = obj.NewInstance ()`
- `vtkPolyLine = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.

- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `int = obj.IsPrimaryCell ()` - Return the center of the point cloud in parametric coordinates.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the point cloud in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)

31.162 vtkPolyVertex

31.162.1 Usage

vtkPolyVertex is a concrete implementation of vtkCell to represent a set of 3D vertices.

To create an instance of class vtkPolyVertex, simply invoke its constructor as follows

```
obj = vtkPolyVertex
```

31.162.2 Methods

The class vtkPolyVertex has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyVertex class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyVertex = obj.NewInstance ()`
- `vtkPolyVertex = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.

- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `int = obj.IsPrimaryCell ()` - Return the center of the point cloud in parametric coordinates.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the point cloud in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)

31.163 vtkProcessObject

31.163.1 Usage

vtkProcessObject is an abstract object that specifies behavior and interface of visualization network process objects (sources, filters, mappers). Source objects are creators of visualization data; filters input, process, and output visualization data; and mappers transform data into another form (like rendering primitives or write data to a file).

vtkProcessObject fires events for Start and End events before and after object execution (via Execute()). These events can be used for any purpose (e.g., debugging info, highlighting/notifying user interface, etc.)

Another event, Progress, can be observed. Some filters fire this event periodically during their execution. The use is similar to that of Start and End events. Filters may also check their AbortExecute flag to determine whether to prematurely end their execution.

An important feature of subclasses of vtkProcessObject is that it is possible to control the memory-management model (i.e., retain output versus delete output data). If enabled the ReleaseDataFlag enables the deletion of the output data once the downstream process object finishes processing the data (please see text).

To create an instance of class vtkProcessObject, simply invoke its constructor as follows

```
obj = vtkProcessObject
```

31.163.2 Methods

The class vtkProcessObject has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProcessObject class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProcessObject = obj.NewInstance ()`
- `vtkProcessObject = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfInputs ()` - Return an array with all the inputs of this process object. This is useful for tracing back in the pipeline to construct graphs etc.
- `obj.SqueezeInputArray ()` - This method will rearrange the input array so that all NULL entries are removed.
- `obj.RemoveAllInputs ()` - Remove all the input data.
- `obj.SetInputConnection (vtkAlgorithmOutput input)` - Reimplemented from vtkAlgorithm to maintain backward compatibility for vtkProcessObject.
- `obj.SetInputConnection (int port, vtkAlgorithmOutput input)` - Reimplemented from vtkAlgorithm to maintain backward compatibility for vtkProcessObject.
- `obj.AddInputConnection (int port, vtkAlgorithmOutput input)` - Reimplemented from vtkAlgorithm to maintain backward compatibility for vtkProcessObject.
- `obj.AddInputConnection (vtkAlgorithmOutput input)` - Reimplemented from vtkAlgorithm to maintain backward compatibility for vtkProcessObject.
- `obj.RemoveInputConnection (int port, vtkAlgorithmOutput input)` - Reimplemented from vtkAlgorithm to maintain backward compatibility for vtkProcessObject.
- `obj.SetNthInputConnection (int port, int index, vtkAlgorithmOutput input)` - Reimplemented from vtkAlgorithm to maintain backward compatibility for vtkProcessObject.
- `obj.SetNumberOfInputConnections (int port, int n)` - Reimplemented from vtkAlgorithm to maintain backward compatibility for vtkProcessObject.

31.164 vtkPropAssembly

31.164.1 Usage

vtkPropAssembly is an object that groups props and other prop assemblies into a tree-like hierarchy. The props can then be treated as a group (e.g., turning visibility on and off).

A vtkPropAssembly object can be used in place of an vtkProp since it is a subclass of vtkProp. The difference is that vtkPropAssembly maintains a list of other prop and prop assembly instances (its "parts") that form the assembly. Note that this process is recursive: you can create groups consisting of prop assemblies to arbitrary depth.

vtkPropAssembly's and vtkProp's that compose a prop assembly need not be added to a renderer's list of props, as long as the parent assembly is in the prop list. This is because they are automatically renderered during the hierarchical traversal process.

To create an instance of class vtkPropAssembly, simply invoke its constructor as follows

```
obj = vtkPropAssembly
```

31.164.2 Methods

The class vtkPropAssembly has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPropAssembly class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPropAssembly = obj.NewInstance ()`
- `vtkPropAssembly = obj.SafeDownCast (vtkObject o)`
- `obj.AddPart (vtkProp )` - Add a part to the list of parts.
- `obj.RemovePart (vtkProp )` - Remove a part from the list of parts,
- `vtkPropCollection = obj.GetParts ()` - Return the list of parts.
- `int = obj.RenderOpaqueGeometry (vtkViewport ren)` - Render this assembly and all its parts. The rendering process is recursive. The parts of each assembly are rendered only if the visibility for the prop is turned on.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport ren)` - Render this assembly and all its parts. The rendering process is recursive. The parts of each assembly are rendered only if the visibility for the prop is turned on.
- `int = obj.RenderVolumetricGeometry (vtkViewport ren)` - Render this assembly and all its parts. The rendering process is recursive. The parts of each assembly are rendered only if the visibility for the prop is turned on.
- `int = obj.RenderOverlay (vtkViewport ren)` - Render this assembly and all its parts. The rendering process is recursive. The parts of each assembly are rendered only if the visibility for the prop is turned on.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `double = obj.GetBounds ()` - Get the bounds for this prop assembly as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax). May return NULL in some cases (meaning the bounds is undefined).
- `obj.ShallowCopy (vtkProp Prop)` - Shallow copy of this vtkPropAssembly.
- `long = obj.GetMTime ()` - Override default GetMTime method to also consider all of the prop assembly's parts.
- `obj.InitPathTraversal ()` - Methods to traverse the paths (i.e., leaf nodes) of a prop assembly. These methods should be contrasted to those that traverse the list of parts using GetParts(). GetParts() returns a list of children of this assembly, not necessarily the leaf nodes of the assembly. To use the methods below - first invoke InitPathTraversal() followed by repeated calls to GetNextPath(). GetNextPath() returns a NULL pointer when the list is exhausted. (See the superclass vtkProp for more information about paths.)

- `vtkAssemblyPath = obj.GetNextPath ()` - Methods to traverse the paths (i.e., leaf nodes) of a prop assembly. These methods should be contrasted to those that traverse the list of parts using GetParts(). GetParts() returns a list of children of this assembly, not necessarily the leaf nodes of the assembly. To use the methods below - first invoke InitPathTraversal() followed by repeated calls to GetNextPath(). GetNextPath() returns a NULL pointer when the list is exhausted. (See the superclass vtkProp for more information about paths.)

- `int = obj.GetNumberOfPaths ()` - Methods to traverse the paths (i.e., leaf nodes) of a prop assembly. These methods should be contrasted to those that traverse the list of parts using GetParts(). GetParts() returns a list of children of this assembly, not necessarily the leaf nodes of the assembly. To use the methods below - first invoke InitPathTraversal() followed by repeated calls to GetNextPath(). GetNextPath() returns a NULL pointer when the list is exhausted. (See the superclass vtkProp for more information about paths.)

31.165 vtkPyramid

31.165.1 Usage

vtkPyramid is a concrete implementation of vtkCell to represent a 3D pyramid. A pyramid consists of a rectangular base with four triangular faces. vtkPyramid uses the standard isoparametric shape functions for a linear pyramid. The pyramid is defined by the five points (0-4) where (0,1,2,3) is the base of the pyramid which, using the right hand rule, forms a quadrilaterial whose normal points in the direction of the pyramid apex at vertex #4.

To create an instance of class vtkPyramid, simply invoke its constructor as follows

```
obj = vtkPyramid
```

31.165.2 Methods

The class vtkPyramid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPyramid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPyramid = obj.NewInstance ()`
- `vtkPyramid = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.

- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the pyramid in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[5])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[15])`

31.166 vtkQuad

31.166.1 Usage

vtkQuad is a concrete implementation of vtkCell to represent a 2D quadrilateral. vtkQuad is defined by the four points (0,1,2,3) in counterclockwise order. vtkQuad uses the standard isoparametric interpolation functions for a linear quadrilateral.

To create an instance of class vtkQuad, simply invoke its constructor as follows

```
obj = vtkQuad
```

31.166.2 Methods

The class vtkQuad has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuad class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuad = obj.NewInstance ()`
- `vtkQuad = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.

- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the triangle in parametric coordinates.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quad using scalar value provided. Like contouring, except that it cuts the quad to produce other quads and/or triangles.
- `obj.InterpolateFunctions (double pcoords[3], double sf[4])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[8])` - Return the ids of the vertices defining edge (‘edgeId‘). Ids are related to the cell, not to the dataset.

31.167 vtkQuadraticEdge

31.167.1 Usage

vtkQuadraticEdge is a concrete implementation of vtkNonLinearCell to represent a one-dimensional, 3-nodes, isoparametric parabolic line. The interpolation is the standard finite element, quadratic isoparametric shape function. The cell includes a mid-edge node. The ordering of the three points defining the cell is point ids (0,1,2) where id #2 is the midedge node.

To create an instance of class vtkQuadraticEdge, simply invoke its constructor as follows

```
obj = vtkQuadraticEdge
```

31.167.2 Methods

The class vtkQuadraticEdge has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraticEdge class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraticEdge = obj.NewInstance ()`
- `vtkQuadraticEdge = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )`

- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this edge using scalar value provided. Like contouring, except that it cuts the edge to produce linear line segments.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the quadratic tetra in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[3])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[3])`

31.168 vtkQuadraticHexahedron

31.168.1 Usage

vtkQuadraticHexahedron is a concrete implementation of vtkNonLinearCell to represent a three-dimensional, 20-node isoparametric parabolic hexahedron. The interpolation is the standard finite element, quadratic isoparametric shape function. The cell includes a mid-edge node. The ordering of the twenty points defining the cell is point ids (0-7,8-19) where point ids 0-7 are the eight corner vertices of the cube; followed by twelve midedge nodes (8-19). Note that these midedge nodes correspond lie on the edges defined by (0,1), (1,2), (2,3), (3,0), (4,5), (5,6), (6,7), (7,4), (0,4), (1,5), (2,6), (3,7).

To create an instance of class vtkQuadraticHexahedron, simply invoke its constructor as follows

```
obj = vtkQuadraticHexahedron
```

31.168.2 Methods

The class vtkQuadraticHexahedron has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraticHexahedron class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraticHexahedron = obj.NewInstance ()`
- `vtkQuadraticHexahedron = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.

- `vtkCell = obj.GetEdge (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quadratic hexahedron using scalar value provided. Like contouring, except that it cuts the hex to produce linear tetrahedron.
- `obj.InterpolateFunctions (double pcoords[3], double weights[20])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[60])` - Return the ids of the vertices defining edge/face ('edgeId'/'faceId'). Ids are related to the cell, not to the dataset.

31.169 vtkQuadraticLinearQuad

31.169.1 Usage

vtkQuadraticQuad is a concrete implementation of vtkNonLinearCell to represent a two-dimensional, 6-node isoparametric quadratic-linear quadrilateral element. The interpolation is the standard finite element, quadratic-linear isoparametric shape function. The cell includes a mid-edge node for two of the four edges. The ordering of the six points defining the cell are point ids (0-3,4-5) where ids 0-3 define the four corner vertices of the quad; ids 4-7 define the midedge nodes (0,1) and (2,3) .

To create an instance of class vtkQuadraticLinearQuad, simply invoke its constructor as follows

```
obj = vtkQuadraticLinearQuad
```

31.169.2 Methods

The class vtkQuadraticLinearQuad has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraticLinearQuad class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraticLinearQuad = obj.NewInstance ()`
- `vtkQuadraticLinearQuad = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.

- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )`
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quadratic linear quad using scalar value provided. Like contouring, except that it cuts the quad to produce linear triangles.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the pyramid in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[6])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[12])` - Return the ids of the vertices defining edge (‘edgeId‘). Ids are related to the cell, not to the dataset.

31.170 vtkQuadraticLinearWedge

31.170.1 Usage

vtkQuadraticLinearWedge is a concrete implementation of vtkNonLinearCell to represent a three-dimensional, 12-node isoparametric linear quadratic wedge. The interpolation is the standard finite element, quadratic isoparametric shape function in xy - layer and the linear functions in z - direction. The cell includes midedge node in the triangle edges. The ordering of the 12 points defining the cell is point ids (0-5,6-12) where point ids 0-5 are the six corner vertices of the wedge; followed by six midedge nodes (6-12). Note that these midedge nodes correspond lie on the edges defined by (0,1), (1,2), (2,0), (3,4), (4,5), (5,3). The Edges (0,3), (1,4), (2,5) dont have midedge nodes.

To create an instance of class vtkQuadraticLinearWedge, simply invoke its constructor as follows

```
obj = vtkQuadraticLinearWedge
```

31.170.2 Methods

The class vtkQuadraticLinearWedge has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraticLinearWedge class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraticLinearWedge = obj.NewInstance ()`
- `vtkQuadraticLinearWedge = obj.SafeDownCast (vtkObject o)`

- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - The quadratic linear wege is splitted into 4 linear wedges, each of them is contoured by a provided scalar value
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - The quadratic linear wege is splitted into 4 linear wedges, each of them is contoured by a provided scalar value
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - The quadratic linear wege is splitted into 4 linear wedges, each of them is contoured by a provided scalar value
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quadratic linear wedge using scalar value provided. Like contouring, except that it cuts the hex to produce linear tetrahedron.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the quadratic linear wedge in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[15])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[45])` - Return the ids of the vertices defining edge/face ('edgeId'/'faceId'). Ids are related to the cell, not to the dataset.

31.171 vtkQuadraticPyramid

31.171.1 Usage

vtkQuadraticPyramid is a concrete implementation of vtkNonLinearCell to represent a three-dimensional, 13-node isoparametric parabolic pyramid. The interpolation is the standard finite element, quadratic isoparametric shape function. The cell includes a mid-edge node. The ordering of the thirteen points defining the cell is point ids (0-4,5-12) where point ids 0-4 are the five corner vertices of the pyramid; followed by eight midedge nodes (5-12). Note that these midedge nodes correspond lie on the edges defined by (0,1), (1,2), (2,3), (3,0), (0,4), (1,4), (2,4), (3,4).

To create an instance of class vtkQuadraticPyramid, simply invoke its constructor as follows

```
obj = vtkQuadraticPyramid
```

31.171.2 Methods

The class vtkQuadraticPyramid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraticPyramid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraticPyramid = obj.NewInstance ()`
- `vtkQuadraticPyramid = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quadratic triangle using scalar value provided. Like contouring, except that it cuts the triangle to produce linear triangles.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the quadratic pyramid in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[13])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[39])` - Return the ids of the vertices defining edge/face (‘edgeId‘/‘faceId’). Ids are related to the cell, not to the dataset.

31.172 vtkQuadraticQuad

31.172.1 Usage

vtkQuadraticQuad is a concrete implementation of vtkNonLinearCell to represent a two-dimensional, 8-node isoparametric parabolic quadrilateral element. The interpolation is the standard finite element, quadratic isoparametric shape function. The cell includes a mid-edge node for each of the four edges of the cell. The ordering of the eight points defining the cell are point ids (0-3,4-7) where ids 0-3 define the four corner vertices of the quad; ids 4-7 define the midedge nodes (0,1), (1,2), (2,3), (3,0).

To create an instance of class vtkQuadraticQuad, simply invoke its constructor as follows

```
obj = vtkQuadraticQuad
```

31.172.2 Methods

The class vtkQuadraticQuad has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraticQuad class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraticQuad = obj.NewInstance ()`
- `vtkQuadraticQuad = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )`
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quadratic quad using scalar value provided. Like contouring, except that it cuts the quad to produce linear triangles.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the pyramid in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[8])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[16])`

31.173 vtkQuadraticTetra

31.173.1 Usage

vtkQuadraticTetra is a concrete implementation of vtkNonLinearCell to represent a three-dimensional, 10-node, isoparametric parabolic tetrahedron. The interpolation is the standard finite element, quadratic isoparametric shape function. The cell includes a mid-edge node on each of the size edges of the tetrahedron. The ordering of the ten points defining the cell is point ids (0-3,4-9) where ids 0-3 are the four tetra vertices; and point ids 4-9 are the midedge nodes between (0,1), (1,2), (2,0), (0,3), (1,3), and (2,3).

To create an instance of class vtkQuadraticTetra, simply invoke its constructor as follows

```
obj = vtkQuadraticTetra
```

31.173.2 Methods

The class vtkQuadraticTetra has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraticTetra class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraticTetra = obj.NewInstance ()`
- `vtkQuadraticTetra = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this edge using scalar value provided. Like contouring, except that it cuts the tetra to produce new tetras.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the quadratic tetra in parametric coordinates.

- `double = obj.GetParametricDistance (double pcoords[3])` - Return the distance of the parametric coordinate provided to the cell. If inside the cell, a distance of zero is returned.
- `obj.InterpolateFunctions (double pcoords[3], double weights[10])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[30])` - Return the ids of the vertices defining edge/face ('edgeId'/'faceId'). Ids are related to the cell, not to the dataset.

31.174 vtkQuadraticTriangle

31.174.1 Usage

vtkQuadraticTriangle is a concrete implementation of vtkNonLinearCell to represent a two-dimensional, 6-node, isoparametric parabolic triangle. The interpolation is the standard finite element, quadratic isoparametric shape function. The cell includes three mid-edge nodes besides the three triangle vertices. The ordering of the three points defining the cell is point ids (0-2,3-5) where id #3 is the midedge node between points (0,1); id #4 is the midedge node between points (1,2); and id #5 is the midedge node between points (2,0).

To create an instance of class vtkQuadraticTriangle, simply invoke its constructor as follows

```
obj = vtkQuadraticTriangle
```

31.174.2 Methods

The class vtkQuadraticTriangle has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraticTriangle class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraticTriangle = obj.NewInstance ()`
- `vtkQuadraticTriangle = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )`
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`

- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quadratic triangle using scalar value provided. Like contouring, except that it cuts the triangle to produce linear triangles.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the quadratic triangle in parametric coordinates.
- `double = obj.GetParametricDistance (double pcoords[3])` - Return the distance of the parametric coordinate provided to the cell. If inside the cell, a distance of zero is returned.
- `obj.InterpolateFunctions (double pcoords[3], double weights[6])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[12])`

31.175 vtkQuadraticWedge

31.175.1 Usage

vtkQuadraticWedge is a concrete implementation of vtkNonLinearCell to represent a three-dimensional, 15-node isoparametric parabolic wedge. The interpolation is the standard finite element, quadratic isoparametric shape function. The cell includes a mid-edge node. The ordering of the fifteen points defining the cell is point ids (0-5,6-14) where point ids 0-5 are the six corner vertices of the wedge; followed by nine midedge nodes (6-14). Note that these midedge nodes correspond lie on the edges defined by (0,1), (1,2), (2,0), (3,4), (4,5), (5,3), (0,3), (1,4), (2,5).

To create an instance of class vtkQuadraticWedge, simply invoke its constructor as follows

```
obj = vtkQuadraticWedge
```

31.175.2 Methods

The class vtkQuadraticWedge has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraticWedge class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraticWedge = obj.NewInstance ()`
- `vtkQuadraticWedge = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.

- `vtkCell = obj.GetFace (int faceId)` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this quadratic hexahedron using scalar value provided. Like contouring, except that it cuts the hex to produce linear tetrahedron.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the quadratic wedge in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[15])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[45])` - Return the ids of the vertices defining edge/face ('edgeId'/'faceId'). Ids are related to the cell, not to the dataset.

31.176 vtkRectilinearGrid

31.176.1 Usage

vtkRectilinearGrid is a data object that is a concrete implementation of vtkDataSet. vtkRectilinearGrid represents a geometric structure that is topologically regular with variable spacing in the three coordinate directions x-y-z.

To define a vtkRectilinearGrid, you must specify the dimensions of the data and provide three arrays of values specifying the coordinates along the x-y-z axes. The coordinate arrays are specified using three vtkDataArray objects (one for x, one for y, one for z).

To create an instance of class vtkRectilinearGrid, simply invoke its constructor as follows

```
obj = vtkRectilinearGrid
```

31.176.2 Methods

The class vtkRectilinearGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGrid = obj.NewInstance ()`
- `vtkRectilinearGrid = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Return what type of dataset this is.
- `obj.CopyStructure (vtkDataSet ds)` - Copy the geometric and topological structure of an input rectilinear grid object.
- `obj.Initialize ()` - Restore object to initial state. Release memory back to system.

- `vtkIdType = obj.GetNumberOfCells ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkIdType = obj.GetNumberOfPoints ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `double = obj.GetPoint (vtkIdType ptId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetPoint (vtkIdType id, double x[3])` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkCell = obj.GetCell (vtkIdType cellId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCell (vtkIdType cellId, vtkGenericCell cell)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCellBounds (vtkIdType cellId, double bounds[6])` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkIdType = obj.FindPoint (double x, double y, double z)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkIdType = obj.FindPoint (double x[3])` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `int = obj.GetCellType (vtkIdType cellId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCellPoints (vtkIdType cellId, vtkIdList ptIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetPointCells (vtkIdType ptId, vtkIdList cellIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.ComputeBounds ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `int = obj.GetMaxCellSize ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCellNeighbors (vtkIdType cellId, vtkIdList ptIds, vtkIdList cellIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.SetDimensions (int i, int j, int k)` - Set dimensions of rectilinear grid dataset. This also sets the extent.
- `obj.SetDimensions (int dim[3])` - Set dimensions of rectilinear grid dataset. This also sets the extent.
- `int = obj. GetDimensions ()` - Get dimensions of this rectilinear grid dataset.
- `int = obj.GetDataDimension ()` - Return the dimensionality of the data.
- `int = obj.ComputeStructuredCoordinates (double x[3], int ijk[3], double pcoords[3])` - Convenience function computes the structured coordinates for a point x[3]. The cell is specified by the array ijk[3], and the parametric coordinates in the cell are specified with pcoords[3]. The function returns a 0 if the point x is outside of the grid, and a 1 if inside the grid.
- `vtkIdType = obj.ComputePointId (int ijk[3])` - Given a location in structured coordinates (i-j-k), return the point id.

- `vtkIdType = obj.ComputeCellId (int ijk[3])` - Given a location in structured coordinates (i-j-k), return the cell id.
- `obj.SetXCoordinates (vtkDataArray )` - Specify the grid coordinates in the x-direction.
- `vtkDataArray = obj.GetXCoordinates ()` - Specify the grid coordinates in the x-direction.
- `obj.SetYCoordinates (vtkDataArray )` - Specify the grid coordinates in the y-direction.
- `vtkDataArray = obj.GetYCoordinates ()` - Specify the grid coordinates in the y-direction.
- `obj.SetZCoordinates (vtkDataArray )` - Specify the grid coordinates in the z-direction.
- `vtkDataArray = obj.GetZCoordinates ()` - Specify the grid coordinates in the z-direction.
- `obj.SetExtent (int extent[6])` - Different ways to set the extent of the data array. The extent should be set before the "Scalars" are set or allocated. The Extent is stored in the order (X, Y, Z).
- `obj.SetExtent (int x1, int x2, int y1, int y2, int z1, int z2)` - Different ways to set the extent of the data array. The extent should be set before the "Scalars" are set or allocated. The Extent is stored in the order (X, Y, Z).
- `int = obj. GetExtent ()` - Different ways to set the extent of the data array. The extent should be set before the "Scalars" are set or allocated. The Extent is stored in the order (X, Y, Z).
- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value). THIS METHOD IS THREAD SAFE.
- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.
- `int = obj.GetExtentType ()` - Structured extent. The extent type is a 3D extent
- `obj.Crop ()` - Reallocates and copies to set the Extent to the UpdateExtent. This is used internally when the exact extent is requested, and the source generated more than the update extent.

31.177 vtkRectilinearGridAlgorithm

31.177.1 Usage

To create an instance of class vtkRectilinearGridAlgorithm, simply invoke its constructor as follows

```
obj = vtkRectilinearGridAlgorithm
```

31.177.2 Methods

The class vtkRectilinearGridAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGridAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGridAlgorithm = obj.NewInstance ()`
- `vtkRectilinearGridAlgorithm = obj.SafeDownCast (vtkObject o)`

- `vtkRectilinearGrid = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkRectilinearGrid = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput ()`
- `vtkDataObject = obj.GetInput (int port)`
- `vtkRectilinearGrid = obj.GetRectilinearGridInput (int port)`
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.178 vtkRectilinearGridSource

31.178.1 Usage

vtkRectilinearGridSource is an abstract class whose subclasses generate rectilinear grid data.

To create an instance of class vtkRectilinearGridSource, simply invoke its constructor as follows

```
obj = vtkRectilinearGridSource
```

31.178.2 Methods

The class vtkRectilinearGridSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGridSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGridSource = obj.NewInstance ()`
- `vtkRectilinearGridSource = obj.SafeDownCast (vtkObject o)`
- `vtkRectilinearGrid = obj.GetOutput ()` - Get the output of this source.
- `vtkRectilinearGrid = obj.GetOutput (int idx)` - Get the output of this source.
- `obj.SetOutput (vtkRectilinearGrid output)` - Get the output of this source.

31.179 vtkRectilinearGridToPolyDataFilter

31.179.1 Usage

vtkRectilinearGridToPolyDataFilter is a filter whose subclasses take as input rectilinear grid datasets and generate polygonal data on output.

To create an instance of class vtkRectilinearGridToPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkRectilinearGridToPolyDataFilter
```

31.179.2 Methods

The class vtkRectilinearGridToPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGridToPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGridToPolyDataFilter = obj.NewInstance ()`
- `vtkRectilinearGridToPolyDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkRectilinearGrid input)` - Set / get the input Grid or filter.
- `vtkRectilinearGrid = obj.GetInput ()` - Set / get the input Grid or filter.

31.180 vtkScalarTree

31.180.1 Usage

vtkScalarTree is an abstract class that defines the API to concrete scalar tree subclasses. A scalar tree is a data structure that organizes data according to its scalar value. This allows rapid access to data for those algorithms that access the data based on scalar value. For example, isocontouring operates on cells based on the scalar (isocontour) value.

To use subclasses of this class, you must specify a dataset to operate on, and then specify a scalar value in the InitTraversal() method. Then calls to GetNextCell() return cells whose scalar data contains the scalar value specified.

To create an instance of class vtkScalarTree, simply invoke its constructor as follows

```
obj = vtkScalarTree
```

31.180.2 Methods

The class vtkScalarTree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkScalarTree class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkScalarTree = obj.NewInstance ()`
- `vtkScalarTree = obj.SafeDownCast (vtkObject o)`

- `obj.SetDataSet (vtkDataSet )` - Build the tree from the points/cells defining this dataset.
- `vtkDataSet = obj.GetDataSet ()` - Build the tree from the points/cells defining this dataset.
- `obj.BuildTree ()` - Construct the scalar tree from the dataset provided. Checks build times and modified time from input and reconstructs the tree if necessary.
- `obj.Initialize ()` - Initialize locator. Frees memory and resets object as appropriate.
- `obj.InitTraversal (double scalarValue)` - Begin to traverse the cells based on a scalar value. Returned cells will have scalar values that span the scalar value specified.

31.181 vtkSelection

31.181.1 Usage

To create an instance of class vtkSelection, simply invoke its constructor as follows

```
obj = vtkSelection
```

31.181.2 Methods

The class vtkSelection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSelection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSelection = obj.NewInstance ()`
- `vtkSelection = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Restore data object to initial state,
- `int = obj.GetDataObjectType ()` - Returns the number of nodes in this selection. Each node contains information about part of the selection.
- `int = obj.GetNumberOfNodes ()` - Returns the number of nodes in this selection. Each node contains information about part of the selection.
- `vtkSelectionNode = obj.GetNode (int idx)` - Returns a node given it's index. Performs bound checking and will return 0 if out-of-bounds.
- `obj.AddNode (vtkSelectionNode )` - Adds a selection node.
- `obj.RemoveNode (int idx)` - Removes a selection node.
- `obj.RemoveNode (vtkSelectionNode )` - Removes a selection node.
- `obj.RemoveAllNodes ()` - Removes a selection node.
- `obj.DeepCopy (vtkDataObject src)` - Copy selection nodes of the input.
- `obj.ShallowCopy (vtkDataObject src)` - Copy selection nodes of the input. This is a shallow copy: selection lists and pointers in the properties are passed by reference.
- `obj.Union (vtkSelection selection)` - Union this selection with the specified selection. Attempts to reuse selection nodes in this selection if properties match exactly. Otherwise, creates new selection nodes.

- `obj.Union (vtkSelectionNode node)` - Union this selection with the specified selection node. Attempts to reuse a selection node in this selection if properties match exactly. Otherwise, creates a new selection node.
- `long = obj.GetMTime ()` - Return the MTime taking into account changes to the properties
- `obj.Dump ()` - Dumps the contents of the selection, giving basic information only.

31.182 vtkSelectionAlgorithm

31.182.1 Usage

vtkSelectionAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline edgehitecture. There are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this class constructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be Selection. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into separate functions such as ExecuteData and ExecuteInformation. For new algorithms you should implement RequestData(request, inputVec, outputVec) but for older filters there is a default implementation that calls the old ExecuteData(output) signature. For even older filters that don't implement ExecuteData the default implementation calls the even older Execute() signature.

.SECTION Thanks Thanks to Patricia Crossno, Ken Moreland, Andrew Wilson and Brian Wylie from Sandia National Laboratories for their help in developing this class.

To create an instance of class vtkSelectionAlgorithm, simply invoke its constructor as follows

```
obj = vtkSelectionAlgorithm
```

31.182.2 Methods

The class vtkSelectionAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSelectionAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSelectionAlgorithm = obj.NewInstance ()`
- `vtkSelectionAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkSelection = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkSelection = obj.GetOutput (int index)` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int index, vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.183 vtkSelectionNode

31.183.1 Usage

To create an instance of class vtkSelectionNode, simply invoke its constructor as follows

```
obj = vtkSelectionNode
```

31.183.2 Methods

The class vtkSelectionNode has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSelectionNode class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSelectionNode = obj.NewInstance ()`
- `vtkSelectionNode = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Restore data object to initial state,
- `obj.SetSelectionList (vtkAbstractArray )` - Sets the selection list.
- `vtkAbstractArray = obj.GetSelectionList ()` - Sets the selection list.
- `obj.SetSelectionData (vtkDataSetAttributes data)` - Sets the selection table.
- `vtkDataSetAttributes = obj.GetSelectionData ()` - Sets the selection table.
- `vtkInformation = obj.GetProperties ()` - Returns the property map.
- `obj.DeepCopy (vtkSelectionNode src)` - Copy properties, selection list and children of the input.
- `obj.ShallowCopy (vtkSelectionNode src)` - Copy properties, selection list and children of the input. This is a shallow copy: selection lists and pointers in the properties are passed by reference.
- `long = obj.GetMTime ()` - Return the MTime taking into account changes to the properties
- `obj.SetContentType (int type)` - Get or set the content type of the selection. This is the same as setting the CONTENT_TYPE() key on the property.
- `int = obj.GetContentType ()` - Get or set the content type of the selection. This is the same as setting the CONTENT_TYPE() key on the property.
- `obj.SetFieldType (int type)` - Get or set the field type of the selection. This is the same as setting the FIELD_TYPE() key on the property.
- `int = obj.GetFieldType ()` - Get or set the field type of the selection. This is the same as setting the FIELD_TYPE() key on the property.
- `obj.SetSelectedProp (vtkProp prop)` - Get or set the prop of the selection. This is the same as setting the PROP() key on the property.
- `vtkProp = obj.GetSelectedProp ()` - Get or set the prop of the selection. This is the same as setting the PROP() key on the property.
- `obj.UnionSelectionList (vtkSelectionNode other)` - Merges the selection list between self and the other. Assumes that both has identical properties.
- `bool = obj.EqualProperties (vtkSelectionNode other, bool fullcomparetrue)` - Compares Properties of self and other to ensure that they are exactly same.

31.184 vtkSimpleCellTessellator

31.184.1 Usage

vtkSimpleCellTessellator is a helper class to perform adaptive tessellation of particular cell topologies. The major purpose for this class is to transform higher-order cell types (e.g., higher-order finite elements) into linear cells that can then be easily visualized by VTK. This class works in conjunction with the vtkGenericDataSet and vtkGenericAdaptorCell classes.

This algorithm is based on edge subdivision. An error metric along each edge is evaluated, and if the error is greater than some tolerance, the edge is subdivided (as well as all connected 2D and 3D cells). The process repeats until the error metric is satisfied. Since the algorithm is based on edge subdivision it inherently avoid T-junctions.

A significant issue addressed by this algorithm is to insure face compatibility across neigboring cells. That is, diagonals due to face triangulation must match to insure that the mesh is compatible. The algorithm employs a precomputed table to accelerate the tessellation process. The table was generated with the help of vtkOrderedTriangulator the basic idea is that the choice of diagonal is made only by considering the relative value of the point ids.

To create an instance of class vtkSimpleCellTessellator, simply invoke its constructor as follows

```
obj = vtkSimpleCellTessellator
```

31.184.2 Methods

The class vtkSimpleCellTessellator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSimpleCellTessellator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSimpleCellTessellator = obj.NewInstance ()`
- `vtkSimpleCellTessellator = obj.SafeDownCast (vtkObject o)`
- `vtkGenericAdaptorCell = obj.GetGenericCell ()` - Get the higher order cell in order to access the evaluation function.
- `obj.TessellateFace (vtkGenericAdaptorCell cell, vtkGenericAttributeCollection att, vtkIdType index,` - Tessellate a face of a 3D 'cell'. The face is specified by the index value. The result is a set of smaller linear triangles in 'cellArray' with 'points' and point data 'internalPd'.
- `obj.Tessellate (vtkGenericAdaptorCell cell, vtkGenericAttributeCollection att, vtkDoubleArray point` - Tessellate a 3D 'cell'. The result is a set of smaller linear tetrahedra in 'cellArray' with 'points' and point data 'internalPd'.
- `obj.Triangulate (vtkGenericAdaptorCell cell, vtkGenericAttributeCollection att, vtkDoubleArray poir` - Triangulate a 2D 'cell'. The result is a set of smaller linear triangles in 'cellArray' with 'points' and point data 'internalPd'.
- `obj.Reset ()` - Reset the output for repeated use of this class.
- `obj.Initialize (vtkGenericDataSet ds)` - Initialize the tessellator with a data set 'ds'.
- `int = obj.GetFixedSubdivisions ()` - Return the number of fixed subdivisions. It is used to prevent from infinite loop in degenerated cases. For order 3 or higher, if the inflection point is exactly on the mid-point, error metric will not detect that a subdivision is required. 0 means no fixed subdivision: there will be only adaptive subdivisions.

The algorithm first performs 'GetFixedSubdivisions' non adaptive subdivisions followed by at most 'GetMaxAdaptiveSubdivisions' adaptive subdivisions. Hence, there are at most 'GetMaxSubdivisionLevel' subdivisions.

- `int = obj.GetMaxSubdivisionLevel ()` - Return the maximum level of subdivision. It is used to prevent from infinite loop in degenerated cases. For order 3 or higher, if the inflection point is exactly on the mid-point, error metric will not detect that a subdivision is required. 0 means no subdivision, neither fixed nor adaptive.

- `int = obj.GetMaxAdaptiveSubdivisions ()` - Return the maximum number of adaptive subdivisions.

- `obj.SetFixedSubdivisions (int level)` - Set the number of fixed subdivisions. See GetFixedSubdivisions() for more explanations.

- `obj.SetMaxSubdivisionLevel (int level)` - Set the maximum level of subdivision. See GetMaxSubdivisionLevel() for more explanations.

- `obj.SetSubdivisionLevels (int fixed, int maxLevel)` - Set both the number of fixed subdivisions and the maximum level of subdivisions. See GetFixedSubdivisions(), GetMaxSubdivisionLevel() and GetMaxAdaptiveSubdivisions() for more explanations.

31.185 vtkSimpleImageToImageFilter

31.185.1 Usage

vtkSimpleImageToImageFilter is a filter which aims to avoid much of the complexity associated with vtkImageAlgorithm (i.e. support for pieces, multi-threaded operation). If you need to write a simple image-image filter which operates on the whole input, use this as the superclass. The subclass has to provide only an execute method which takes input and output as arguments. Memory allocation is handled in vtkSimpleImageToImageFilter. Also, you are guaranteed to have a valid input in the Execute(input, output) method. By default, this filter requests it's input's whole extent and copies the input's information (spacing, whole extent etc...) to the output. If the output's setup is different (for example, if it performs some sort of sub-sampling), ExecuteInformation has to be overwritten. As an example of how this can be done, you can look at vtkImageShrink3D::ExecuteInformation. For a complete example which uses templates to support generic data types, see vtkSimpleImageToImageFilter.

To create an instance of class vtkSimpleImageToImageFilter, simply invoke its constructor as follows

```
obj = vtkSimpleImageToImageFilter
```

31.185.2 Methods

The class vtkSimpleImageToImageFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSimpleImageToImageFilter class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkSimpleImageToImageFilter = obj.NewInstance ()`

- `vtkSimpleImageToImageFilter = obj.SafeDownCast (vtkObject o)`

31.186 vtkSimpleScalarTree

31.186.1 Usage

vtkSimpleScalarTree creates a pointerless binary tree that helps search for cells that lie within a particular scalar range. This object is used to accelerate some contouring (and other scalar-based techniques).

The tree consists of an array of (min,max) scalar range pairs per node in the tree. The (min,max) range is determined from looking at the range of the children of the tree node. If the node is a leaf, then the range is determined by scanning the range of scalar data in n cells in the dataset. The n cells are determined by arbitrary selecting cell ids from id(i) to id(i+n), and where n is specified using the BranchingFactor ivar. Note that leaf node i=0 contains the scalar range computed from cell ids (0,n-1); leaf node i=1 contains the range from cell ids (n,2n-1); and so on. The implication is that there are no direct lists of cell ids per leaf node, instead the cell ids are implicitly known.

To create an instance of class vtkSimpleScalarTree, simply invoke its constructor as follows

```
obj = vtkSimpleScalarTree
```

31.186.2 Methods

The class vtkSimpleScalarTree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSimpleScalarTree class.

- `string = obj.GetClassName ()` - Standard type related macros and PrintSelf() method.
- `int = obj.IsA (string name)` - Standard type related macros and PrintSelf() method.
- `vtkSimpleScalarTree = obj.NewInstance ()` - Standard type related macros and PrintSelf() method.
- `vtkSimpleScalarTree = obj.SafeDownCast (vtkObject o)` - Standard type related macros and PrintSelf() method.
- `obj.SetBranchingFactor (int )` - Set the branching factor for the tree. This is the number of children per tree node. Smaller values (minimum is 2) mean deeper trees and more memory overhead. Larger values mean shallower trees, less memory usage, but worse performance.
- `int = obj.GetBranchingFactorMinValue ()` - Set the branching factor for the tree. This is the number of children per tree node. Smaller values (minimum is 2) mean deeper trees and more memory overhead. Larger values mean shallower trees, less memory usage, but worse performance.
- `int = obj.GetBranchingFactorMaxValue ()` - Set the branching factor for the tree. This is the number of children per tree node. Smaller values (minimum is 2) mean deeper trees and more memory overhead. Larger values mean shallower trees, less memory usage, but worse performance.
- `int = obj.GetBranchingFactor ()` - Set the branching factor for the tree. This is the number of children per tree node. Smaller values (minimum is 2) mean deeper trees and more memory overhead. Larger values mean shallower trees, less memory usage, but worse performance.
- `int = obj.GetLevel ()` - Get the level of the scalar tree. This value may change each time the scalar tree is built and the branching factor changes.
- `obj.SetMaxLevel (int )` - Set the maximum allowable level for the tree.
- `int = obj.GetMaxLevelMinValue ()` - Set the maximum allowable level for the tree.
- `int = obj.GetMaxLevelMaxValue ()` - Set the maximum allowable level for the tree.
- `int = obj.GetMaxLevel ()` - Set the maximum allowable level for the tree.

- `obj.BuildTree ()` - Construct the scalar tree from the dataset provided. Checks build times and modified time from input and reconstructs the tree if necessary.
- `obj.Initialize ()` - Initialize locator. Frees memory and resets object as appropriate.
- `obj.InitTraversal (double scalarValue)` - Begin to traverse the cells based on a scalar value. Returned cells will have scalar values that span the scalar value specified.

31.187 vtkSmoothErrorMetric

31.187.1 Usage

It is a concrete error metric, based on a geometric criterium: a max angle between the chord passing through the midpoint and one of the endpoints and the other chord passing through the midpoint and the other endpoint of the edge. It is related to the flatness of an edge.

To create an instance of class vtkSmoothErrorMetric, simply invoke its constructor as follows

```
obj = vtkSmoothErrorMetric
```

31.187.2 Methods

The class vtkSmoothErrorMetric has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSmoothErrorMetric class.

- `string = obj.GetClassName ()` - Standard VTK type and error macros.
- `int = obj.IsA (string name)` - Standard VTK type and error macros.
- `vtkSmoothErrorMetric = obj.NewInstance ()` - Standard VTK type and error macros.
- `vtkSmoothErrorMetric = obj.SafeDownCast (vtkObject o)` - Standard VTK type and error macros.
- `double = obj.GetAngleTolerance ()` - Return the flatness threshold.
- `obj.SetAngleTolerance (double value)` - Set the flatness threshold with an angle in degrees. Internally compute the cosine. value is supposed to be in]90,180[, if not it is clamped in [90.1,179.9]. For instance 178 will give better result than 150.
- `int = obj.RequiresEdgeSubdivision (double leftPoint, double midPoint, double rightPoint, double alp` - Does the edge need to be subdivided according to the cosine between the two chords passing through the mid-point and the endpoints? The edge is defined by its 'leftPoint' and its 'rightPoint'. 'leftPoint', 'midPoint' and 'rightPoint' have to be initialized before calling RequiresEdgeSubdivision(). Their format is global coordinates, parametric coordinates and point centered attributes: xyx rst abc de... 'alpha' is the normalized abscissa of the midpoint along the edge. (close to 0 means close to the left point, close to 1 means close to the right point)

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6
- `double = obj.GetError (double leftPoint, double midPoint, double rightPoint, double alpha)` - Return the error at the mid-point. It will return an error relative to the bounding box size if GetRelative() is true, a square absolute error otherwise. See RequiresEdgeSubdivision() for a description of the arguments.

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6

31.188 vtkSource

31.188.1 Usage

vtkSource is an abstract object that specifies behavior and interface of source objects. Source objects are objects that begin visualization pipeline. Sources include readers (read data from file or communications port) and procedural sources (generate data programmatically). vtkSource objects are also objects that generate output data. In this sense vtkSource is used as a superclass to vtkFilter.

Concrete subclasses of vtkSource must define Update() and Execute() methods. The public method Update() invokes network execution and will bring the network up-to-date. The protected Execute() method actually does the work of data creation/generation. The difference between the two methods is that Update() implements input consistency checks and modified time comparisons and then invokes the Execute() which is an implementation of a particular algorithm.

An important feature of subclasses of vtkSource is that it is possible to control the memory-management model (i.e., retain output versus delete output data). If enabled the ReleaseDataFlag enables the deletion of the output data once the downstream process object finishes processing the data (please see text).

To create an instance of class vtkSource, simply invoke its constructor as follows

```
obj = vtkSource
```

31.188.2 Methods

The class vtkSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSource = obj.NewInstance ()`
- `vtkSource = obj.SafeDownCast (vtkObject o)`
- `obj.Update ()` - Bring object up-to-date before execution. Update() checks modified time against last execution time, and re-executes object if necessary.
- `obj.UpdateWholeExtent ()` - Like update, but make sure the update extent is the whole extent in the output.
- `obj.UpdateInformation ()` - Updates any global information about the data (like spacing for images)
- `obj.PropagateUpdateExtent (vtkDataObject output)` - WARNING: INTERNAL METHOD - NOT FOR GENERAL USE. THIS METHOD IS PART OF THE PIPELINE UPDATE FUNCTIONALITY. The update extent for this object is propagated up the pipeline. This propagation may early terminate based on the PipelineMTime.
- `obj.TriggerAsynchronousUpdate ()` - WARNING: INTERNAL METHOD - NOT FOR GENERAL USE. THIS METHOD IS PART OF THE PIPELINE UPDATE FUNCTIONALITY. Propagate back up the pipeline for ports and trigger the update on the other side of the port to allow for asynchronous parallel processing in the pipeline. This propagation may early terminate based on the PipelineMTime.
- `obj.UpdateData (vtkDataObject output)` - WARNING: INTERNAL METHOD - NOT FOR GENERAL USE. THIS METHOD IS PART OF THE PIPELINE UPDATE FUNCTIONALITY. Propagate the update back up the pipeline, and perform the actual work of updating on the way down. When the propagate arrives at a port, block and wait for the asynchronous update to finish on the other side. This propagation may early terminate based on the PipelineMTime.

- `obj.ComputeInputUpdateExtents (vtkDataObject output)` - What is the input update extent that is required to produce the desired output? By default, the whole input is always required but this is overridden in many subclasses.
- `obj.SetReleaseDataFlag (int )` - Turn on/off flag to control whether this object's data is released after being used by a source.
- `int = obj.GetReleaseDataFlag ()` - Turn on/off flag to control whether this object's data is released after being used by a source.
- `obj.ReleaseDataFlagOn ()` - Turn on/off flag to control whether this object's data is released after being used by a source.
- `obj.ReleaseDataFlagOff ()` - Turn on/off flag to control whether this object's data is released after being used by a source.
- `int = obj.GetNumberOfOutputs ()` - Return an array with all the inputs of this process object. This is useful for tracing back in the pipeline to construct graphs etc.
- `obj.UnRegisterAllOutputs (void )` - Release/disconnect all outputs of this source. This is intended to be called prior to Delete() if the user is concerned about outputs holding on to the filter/source.
- `int = obj.GetOutputIndex (vtkDataObject out)` - Return what index output the passed in output is, return -1 if it does not match any of the outputs
- `obj.SetExecutive (vtkExecutive executive)` - Set this algorithm's executive. This algorithm is removed from any executive to which it has previously been assigned and then assigned to the given executive.

31.189 vtkSphere

31.189.1 Usage

vtkSphere computes the implicit function and/or gradient for a sphere. vtkSphere is a concrete implementation of vtkImplicitFunction. Additional methods are available for sphere-related computations, such as computing bounding spheres for a set of points, or set of spheres.

To create an instance of class vtkSphere, simply invoke its constructor as follows

```
obj = vtkSphere
```

31.189.2 Methods

The class vtkSphere has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSphere class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSphere = obj.NewInstance ()`
- `vtkSphere = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double n[3])`

- `obj.SetRadius (double )` - Set / get the radius of the sphere.
- `double = obj.GetRadius ()` - Set / get the radius of the sphere.
- `obj.SetCenter (double , double , double )` - Set / get the center of the sphere.
- `obj.SetCenter (double  a[3])` - Set / get the center of the sphere.
- `double = obj. GetCenter ()` - Set / get the center of the sphere.

31.190 vtkSpline

31.190.1 Usage

vtkSpline interpolates a set of data points (i.e., interpolation means that the spline passes through the points). vtkSpline is an abstract class: its subclasses vtkCardinalSpline and vtkKochenekSpline do the interpolation. Note that this spline maps the 1D parametric coordinate t into a single value x. Thus if you want to use the spline to interpolate points (i.e. x[3]), you have to create three splines for each of the x-y-z coordinates. Fortunately, the vtkParametricSpline class does this for you.

Typically a spline is used by adding a sequence of parametric coordinate / data (t,x) values followed by use of an evaluation function (e.g., vtkCardinalSpline::Evaluate()). Since these splines are 1D, a point in this context is an independent / dependent variable pair.

Splines can also be set up to be closed or open. Closed splines continue from the last point to the first point with continuous function and derivative values. (You don't need to duplicate the first point to close the spline, just set ClosedOn.)

This implementation of splines does not use a normalized parametric coordinate. If the spline is open, then the parameter space is (tMin ¡= t ¡= tMax) where tMin and tMax are the minimum and maximum parametric values seen when performing AddPoint(). If the spline is closed, then the parameter space is (tMin ¡= t ¡= (tMax+1)) where tMin and tMax are the minimum and maximum parametric values seen when performing AddPoint(). Note, however, that this behavior can be changed by explicitly setting the ParametricRange(tMin,tMax). If set, the parameter space remains (tMin ¡= t ¡= tMax), except that additions of data with parametric values outside this range are clamped within this range.

To create an instance of class vtkSpline, simply invoke its constructor as follows

```
obj = vtkSpline
```

31.190.2 Methods

The class vtkSpline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSpline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSpline = obj.NewInstance ()`
- `vtkSpline = obj.SafeDownCast (vtkObject o)`
- `obj.SetParametricRange (double tMin, double tMax)` - Set/Get the parametric range. If not set, the range is determined implicitly by keeping track of the (min,max) parameter values for t. If set, the AddPoint() method will clamp the t value to lie within the specified range.
- `obj.SetParametricRange (double tRange[2])` - Set/Get the parametric range. If not set, the range is determined implicitly by keeping track of the (min,max) parameter values for t. If set, the AddPoint() method will clamp the t value to lie within the specified range.

- `obj.GetParametricRange (double tRange[2]) const` - Set/Get the parametric range. If not set, the range is determined implicitly by keeping track of the (min,max) parameter values for t. If set, the AddPoint() method will clamp the t value to lie within the specified range.
- `obj.SetClampValue (int )` - Set/Get ClampValue. If On, results of the interpolation will be clamped to the min/max of the input data.
- `int = obj.GetClampValue ()` - Set/Get ClampValue. If On, results of the interpolation will be clamped to the min/max of the input data.
- `obj.ClampValueOn ()` - Set/Get ClampValue. If On, results of the interpolation will be clamped to the min/max of the input data.
- `obj.ClampValueOff ()` - Set/Get ClampValue. If On, results of the interpolation will be clamped to the min/max of the input data.
- `obj.Compute ()` - Compute the coefficients for the spline.
- `double = obj.Evaluate (double t)` - Interpolate the value of the spline at parametric location of t.
- `int = obj.GetNumberOfPoints ()` - Return the number of points inserted thus far.
- `obj.AddPoint (double t, double x)` - Add a pair of points to be fit with the spline.
- `obj.RemovePoint (double t)` - Remove a point from the data to be fit with the spline.
- `obj.RemoveAllPoints ()` - Remove all points from the data.
- `obj.SetClosed (int )` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous.
- `int = obj.GetClosed ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous.
- `obj.ClosedOn ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous.
- `obj.ClosedOff ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous.
- `obj.SetLeftConstraint (int )` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.
- `int = obj.GetLeftConstraintMinValue ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetLeftConstraintMaxValue ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetLeftConstraint ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `obj.SetRightConstraint (int )` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetRightConstraintMinValue ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetRightConstraintMaxValue ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

  1: the first derivative at left(right) most point is set to Left(Right)Value.

  2: the second derivative at left(right) most point is set to Left(Right)Value.

  3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `int = obj.GetRightConstraint ()` - Set the type of constraint of the left(right) end points. Four constraints are available:

  0: the first derivative at left(right) most point is determined from the line defined from the first(last) two points.

1: the first derivative at left(right) most point is set to Left(Right)Value.

2: the second derivative at left(right) most point is set to Left(Right)Value.

3: the second derivative at left(right)most points is Left(Right)Value times second derivative at first interior point.

- `obj.SetLeftValue (double )` - The values of the derivative on the left and right sides. The value is used only if the left(right) constraint is type 1-3.
- `double = obj.GetLeftValue ()` - The values of the derivative on the left and right sides. The value is used only if the left(right) constraint is type 1-3.
- `obj.SetRightValue (double )` - The values of the derivative on the left and right sides. The value is used only if the left(right) constraint is type 1-3.
- `double = obj.GetRightValue ()` - The values of the derivative on the left and right sides. The value is used only if the left(right) constraint is type 1-3.
- `long = obj.GetMTime ()` - Return the MTime also considering the Piecewise function.
- `obj.DeepCopy (vtkSpline s)` - Deep copy of spline data.

31.191 vtkStreamingDemandDrivenPipeline

31.191.1 Usage

vtkStreamingDemandDrivenPipeline is an executive that supports updating only a portion of the data set in the pipeline. This is the style of pipeline update that is provided by the old-style VTK 4.x pipeline. Instead of always updating an entire data set, this executive supports asking for pieces or sub-extents.

To create an instance of class vtkStreamingDemandDrivenPipeline, simply invoke its constructor as follows

```
obj = vtkStreamingDemandDrivenPipeline
```

31.191.2 Methods

The class vtkStreamingDemandDrivenPipeline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStreamingDemandDrivenPipeline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStreamingDemandDrivenPipeline = obj.NewInstance ()`
- `vtkStreamingDemandDrivenPipeline = obj.SafeDownCast (vtkObject o)`
- `int = obj.Update ()` - Bring the outputs up-to-date.
- `int = obj.Update (int port)` - Bring the outputs up-to-date.
- `int = obj.UpdateWholeExtent ()` - Bring the outputs up-to-date.
- `int = obj.PropagateUpdateExtent (int outputPort)` - Propagate the update request from the given output port back through the pipeline. Should be called only when information is up to date.

- `int = obj.SetMaximumNumberOfPieces (int port, int n)` - Set/Get the maximum number of pieces that can be requested from the given port. The maximum number of pieces is meta data for unstructured data sets. It gets set by the source during the update information call. A value of -1 indicates that there is no maximum.

- `int = obj.SetMaximumNumberOfPieces (vtkInformation , int n)` - Set/Get the maximum number of pieces that can be requested from the given port. The maximum number of pieces is meta data for unstructured data sets. It gets set by the source during the update information call. A value of -1 indicates that there is no maximum.

- `int = obj.GetMaximumNumberOfPieces (int port)` - Set/Get the maximum number of pieces that can be requested from the given port. The maximum number of pieces is meta data for unstructured data sets. It gets set by the source during the update information call. A value of -1 indicates that there is no maximum.

- `int = obj.GetMaximumNumberOfPieces (vtkInformation )` - Set/Get the maximum number of pieces that can be requested from the given port. The maximum number of pieces is meta data for unstructured data sets. It gets set by the source during the update information call. A value of -1 indicates that there is no maximum.

- `int = obj.SetWholeExtent (vtkInformation , int extent[6])` - Set/Get the whole extent of an output port. The whole extent is meta data for structured data sets. It gets set by the algorithm during the update information pass.

- `obj.GetWholeExtent (vtkInformation , int extent[6])` - Set/Get the whole extent of an output port. The whole extent is meta data for structured data sets. It gets set by the algorithm during the update information pass.

- `int = obj.SetUpdateExtentToWholeExtent (int port)` - If the whole input extent is required to generate the requested output extent, this method can be called to set the input update extent to the whole input extent. This method assumes that the whole extent is known (that UpdateInformation has been called)

- `int = obj.SetUpdateExtentToWholeExtent (vtkInformation )` - If the whole input extent is required to generate the requested output extent, this method can be called to set the input update extent to the whole input extent. This method assumes that the whole extent is known (that UpdateInformation has been called)

- `int = obj.SetUpdateExtent (int port, int extent[6])` - Get/Set the update extent for output ports that use 3D extents.

- `int = obj.SetUpdateExtent (vtkInformation , int extent[6])` - Get/Set the update extent for output ports that use 3D extents.

- `obj.GetUpdateExtent (vtkInformation , int extent[6])` - Get/Set the update extent for output ports that use 3D extents.

- `int = obj.SetUpdateExtent (int port, int piece, int numPieces, int ghostLevel)` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.

- `int = obj.SetUpdateExtent (vtkInformation , int piece, int numPieces, int ghostLevel)` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.

- `int = obj.SetUpdatePiece (vtkInformation , int piece)` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.

- `int = obj.GetUpdatePiece (vtkInformation )` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.
- `int = obj.SetUpdateNumberOfPieces (vtkInformation , int n)` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.
- `int = obj.GetUpdateNumberOfPieces (vtkInformation )` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.
- `int = obj.SetUpdateGhostLevel (vtkInformation , int n)` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.
- `int = obj.GetUpdateGhostLevel (vtkInformation )` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.
- `int = obj.SetUpdateResolution (int port, double r)` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.
- `int = obj.SetUpdateResolution (vtkInformation , double r)` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.
- `double = obj.GetUpdateResolution (vtkInformation )` - Set/Get the update piece, update number of pieces, and update number of ghost levels for an output port. Similar to update extent in 3D.
- `int = obj.SetSplitUpdateExtent (int port, int major, int minor, int numPieces, int ghostLevel)` - Get/Set the update extent for output ports that use Temporal Extents
- `int = obj.SetUpdateTimeSteps (int port, double times, int length)` - Get/Set the update extent for output ports that use Temporal Extents
- `int = obj.SetUpdateTimeSteps (vtkInformation , double times, int length)` - Get/Set the update extent for output ports that use Temporal Extents
- `int = obj.SetUpdateTimeStep (int port, double time)` - Get/Set the update extent for output ports that use Temporal Extents
- `int = obj.SetRequestExactExtent (int port, int flag)` - This request flag indicates whether the requester can handle more data than requested for the given port. Right now it is used in vtkImageData. Image filters can return more data than requested. The the consumer cannot handle this (i.e. DataSetToDataSetFitler) the image will crop itself. This functionality used to be in ImageToStructuredPoints.
- `int = obj.GetRequestExactExtent (int port)` - This request flag indicates whether the requester can handle more data than requested for the given port. Right now it is used in vtkImageData. Image filters can return more data than requested. The the consumer cannot handle this (i.e. DataSetToDataSetFitler) the image will crop itself. This functionality used to be in ImageToStructuredPoints.
- `int = obj.SetExtentTranslator (int port, vtkExtentTranslator translator)` - Get/Set the object that will translate pieces into structured extents for an output port.
- `int = obj.SetExtentTranslator (vtkInformation , vtkExtentTranslator translator)` - Get/Set the object that will translate pieces into structured extents for an output port.
- `vtkExtentTranslator = obj.GetExtentTranslator (int port)` - Get/Set the object that will translate pieces into structured extents for an output port.

- `vtkExtentTranslator = obj.GetExtentTranslator (vtkInformation info)` - Get/Set the object that will translate pieces into structured extents for an output port.
- `int = obj.SetWholeBoundingBox (int port, double bb[6])` - Set/Get the whole bounding box of an output port data object. The whole whole bounding box is meta data for data sets. It gets set by the algorithm during the update information pass.
- `obj.GetWholeBoundingBox (int port, double bb[6])` - Set/Get the whole bounding box of an output port data object. The whole whole bounding box is meta data for data sets. It gets set by the algorithm during the update information pass.
- `int = obj.SetPieceBoundingBox (int port, double bb[6])` - Set/Get the piece bounding box of an output port data object. The piece bounding box is meta data for data sets. It gets set by the algorithm during the update extent information pass.
- `obj.GetPieceBoundingBox (int port, double bb[6])` - Set/Get the piece bounding box of an output port data object. The piece bounding box is meta data for data sets. It gets set by the algorithm during the update extent information pass.
- `double = obj.ComputePriority ()` - Issues pipeline request to determine and return the priority of the piece described by the current update extent. The priority is a number between 0.0 and 1.0 with 0 meaning skippable (REQUEST_DATA not needed) and 1.0 meaning important.
- `double = obj.ComputePriority (int port)` - Issues pipeline request to determine and return the priority of the piece described by the current update extent. The priority is a number between 0.0 and 1.0 with 0 meaning skippable (REQUEST_DATA not needed) and 1.0 meaning important.

31.192 vtkStructuredGrid

31.192.1 Usage

vtkStructuredGrid is a data object that is a concrete implementation of vtkDataSet. vtkStructuredGrid represents a geometric structure that is a topologically regular array of points. The topology is that of a cube that has been subdivided into a regular array of smaller cubes. Each point/cell can be addressed with i-j-k indices. Examples include finite difference grids.

The order and number of points must match that specified by the dimensions of the grid. The point order increases in i fastest (from 0¡=i¡dims[0]), then j (0¡=j¡dims[1]), then k (0¡=k¡dims[2]) where dims[] are the dimensions of the grid in the i-j-k topological directions. The number of points is dims[0]*dims[1]*dims[2]. The same is true for the cells of the grid. The order and number of cells must match that specified by the dimensions of the grid. The cell order increases in i fastest (from 0¡=i¡(dims[0]-1)), then j (0¡=j¡(dims[1]-1)), then k (0¡=k¡(dims[2]-1)) The number of cells is (dims[0]-1)*(dims[1]-1)*(dims[2]-1).

A unusual feature of vtkStructuredGrid is the ability to blank, or "turn-off" points and cells in the dataset. This is controlled by defining a "blanking array" whose values (0,1) specify whether a point should be blanked or not.

To create an instance of class vtkStructuredGrid, simply invoke its constructor as follows

```
obj = vtkStructuredGrid
```

31.192.2 Methods

The class vtkStructuredGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGrid class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkStructuredGrid = obj.NewInstance ()`
- `vtkStructuredGrid = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Copy the geometric and topological structure of an input poly data object.
- `obj.CopyStructure (vtkDataSet ds)` - Copy the geometric and topological structure of an input poly data object.
- `vtkIdType = obj.GetNumberOfPoints ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `double = obj.GetPoint (vtkIdType ptId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetPoint (vtkIdType ptId, double p[3])` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkCell = obj.GetCell (vtkIdType cellId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCell (vtkIdType cellId, vtkGenericCell cell)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCellBounds (vtkIdType cellId, double bounds[6])` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `int = obj.GetCellType (vtkIdType cellId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkIdType = obj.GetNumberOfCells ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCellPoints (vtkIdType cellId, vtkIdList ptIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetPointCells (vtkIdType ptId, vtkIdList cellIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.Initialize ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `int = obj.GetMaxCellSize ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCellNeighbors (vtkIdType cellId, vtkIdList ptIds, vtkIdList cellIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetScalarRange (double range[2])` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `double = obj.GetScalarRange ()` - following methods are specific to structured grid
- `obj.SetDimensions (int i, int j, int k)` - following methods are specific to structured grid
- `obj.SetDimensions (int dim[3])` - following methods are specific to structured grid
- `int = obj.GetDimensions ()` - Get dimensions of this structured points dataset.
- `obj.GetDimensions (int dim[3])` - Get dimensions of this structured points dataset.

- `int = obj.GetDataDimension ()` - Return the dimensionality of the data.
- `obj.SetExtent (int extent[6])` - Different ways to set the extent of the data array. The extent should be set before the "Scalars" are set or allocated. The Extent is stored in the order (X, Y, Z).
- `obj.SetExtent (int x1, int x2, int y1, int y2, int z1, int z2)` - Different ways to set the extent of the data array. The extent should be set before the "Scalars" are set or allocated. The Extent is stored in the order (X, Y, Z).
- `int = obj. GetExtent ()` - Different ways to set the extent of the data array. The extent should be set before the "Scalars" are set or allocated. The Extent is stored in the order (X, Y, Z).
- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value). THIS METHOD IS THREAD SAFE.
- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.
- `int = obj.GetExtentType ()` - Methods for supporting blanking of cells. Blanking turns on or off points in the structured grid, and hence the cells connected to them. These methods should be called only after the dimensions of the grid are set.
- `obj.BlankPoint (vtkIdType ptId)` - Methods for supporting blanking of cells. Blanking turns on or off points in the structured grid, and hence the cells connected to them. These methods should be called only after the dimensions of the grid are set.
- `obj.UnBlankPoint (vtkIdType ptId)` - Methods for supporting blanking of cells. Blanking turns on or off points in the structured grid, and hence the cells connected to them. These methods should be called only after the dimensions of the grid are set.
- `obj.BlankCell (vtkIdType ptId)` - Methods for supporting blanking of cells. Blanking turns on or off cells in the structured grid, and hence the cells connected to them. These methods should be called only after the dimensions of the grid are set.
- `obj.UnBlankCell (vtkIdType ptId)` - Methods for supporting blanking of cells. Blanking turns on or off cells in the structured grid, and hence the cells connected to them. These methods should be called only after the dimensions of the grid are set.
- `vtkUnsignedCharArray = obj.GetPointVisibilityArray ()` - Get the array that defines the blanking (visibility) of each point.
- `obj.SetPointVisibilityArray (vtkUnsignedCharArray pointVisibility)` - Set an array that defines the (blanking) visibility of the points in the grid. Make sure that length of the visibility array matches the number of points in the grid.
- `vtkUnsignedCharArray = obj.GetCellVisibilityArray ()` - Get the array that defines the blanking (visibility) of each cell.
- `obj.SetCellVisibilityArray (vtkUnsignedCharArray pointVisibility)` - Set an array that defines the (blanking) visibility of the cells in the grid. Make sure that length of the visibility array matches the number of points in the grid.
- `char = obj.IsPointVisible (vtkIdType ptId)` - Return non-zero value if specified point is visible. These methods should be called only after the dimensions of the grid are set.
- `char = obj.IsCellVisible (vtkIdType cellId)` - Return non-zero value if specified point is visible. These methods should be called only after the dimensions of the grid are set.

- `char = obj.GetPointBlanking ()` - Returns 1 if there is any visibility constraint on the points, 0 otherwise.
- `char = obj.GetCellBlanking ()` - Returns 1 if there is any visibility constraint on the cells, 0 otherwise.
- `obj.Crop ()` - Reallocates and copies to set the Extent to the UpdateExtent. This is used internally when the exact extent is requested, and the source generated more than the update extent.
- `obj.GetPoint (int i, int j, int k, double p[3], bool adjustForExtenttrue)` - Get a point in the grid. If adjustForExtent is true, (i,j,k) is interpreted as a position relative to the beginning of the extent. If adjustForExtent is false, (i,j,k) is interpreted literally and the (i,j,k) point of the grid is returned regardless of the extent beginning. The point coordinate is returned in 'p'. The default adjustForExtent is true.

31.193 vtkStructuredGridAlgorithm

31.193.1 Usage

To create an instance of class vtkStructuredGridAlgorithm, simply invoke its constructor as follows

```
obj = vtkStructuredGridAlgorithm
```

31.193.2 Methods

The class vtkStructuredGridAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGridAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredGridAlgorithm = obj.NewInstance ()`
- `vtkStructuredGridAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkStructuredGrid = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkStructuredGrid = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput ()`
- `vtkDataObject = obj.GetInput (int port)`
- `vtkStructuredGrid = obj.GetStructuredGridInput (int port)`
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.194 vtkStructuredGridSource

31.194.1 Usage

vtkStructuredGridSource is an abstract class whose subclasses generate structured grid data.

To create an instance of class vtkStructuredGridSource, simply invoke its constructor as follows

```
obj = vtkStructuredGridSource
```

31.194.2 Methods

The class vtkStructuredGridSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGridSource class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkStructuredGridSource = obj.NewInstance ()`

- `vtkStructuredGridSource = obj.SafeDownCast (vtkObject o)`

- `vtkStructuredGrid = obj.GetOutput ()` - Get the output of this source.

- `vtkStructuredGrid = obj.GetOutput (int idx)` - Get the output of this source.

- `obj.SetOutput (vtkStructuredGrid output)` - Get the output of this source.

31.195 vtkStructuredGridToPolyDataFilter

31.195.1 Usage

vtkStructuredGridToPolyDataFilter is a filter whose subclasses take as input structured grid datasets and generate polygonal data on output.

To create an instance of class vtkStructuredGridToPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkStructuredGridToPolyDataFilter
```

31.195.2 Methods

The class vtkStructuredGridToPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGridToPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredGridToPolyDataFilter = obj.NewInstance ()`
- `vtkStructuredGridToPolyDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkStructuredGrid input)` - Set / get the input Grid or filter.
- `vtkStructuredGrid = obj.GetInput ()` - Set / get the input Grid or filter.

31.196 vtkStructuredGridToStructuredGridFilter

31.196.1 Usage

vtkStructuredPointsToStructuredPointsFilter is an abstract filter class whose subclasses take on input a structured grid and generate a structured grid on output.

To create an instance of class vtkStructuredGridToStructuredGridFilter, simply invoke its constructor as follows

```
obj = vtkStructuredGridToStructuredGridFilter
```

31.196.2 Methods

The class vtkStructuredGridToStructuredGridFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGridToStructuredGridFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredGridToStructuredGridFilter = obj.NewInstance ()`
- `vtkStructuredGridToStructuredGridFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkStructuredGrid input)` - Set / get the input Grid or filter.
- `vtkStructuredGrid = obj.GetInput ()` - Set / get the input Grid or filter.

31.197 vtkStructuredPoints

31.197.1 Usage

StructuredPoints is a subclass of ImageData that requires the data extent to exactly match the update extent. Normall image data allows that the data extent may be larger than the update extent. StructuredPoints also defines the origin differently that vtkImageData. For structured points the origin is the location of first point. Whereas images define the origin as the location of point 0, 0, 0. Image Origin is stored in ivar, and structured points have special methods for setting/getting the origin/extents.

To create an instance of class vtkStructuredPoints, simply invoke its constructor as follows

```
obj = vtkStructuredPoints
```

31.197.2 Methods

The class vtkStructuredPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredPoints = obj.NewInstance ()`
- `vtkStructuredPoints = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()`

31.198 vtkStructuredPointsCollection

31.198.1 Usage

vtkStructuredPointsCollection is an object that creates and manipulates lists of structured points datasets. See also vtkCollection and subclasses.

To create an instance of class vtkStructuredPointsCollection, simply invoke its constructor as follows

```
obj = vtkStructuredPointsCollection
```

31.198.2 Methods

The class vtkStructuredPointsCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredPointsCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredPointsCollection = obj.NewInstance ()`
- `vtkStructuredPointsCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkStructuredPoints ds)` - Get the next item in the collection. NULL is returned if the collection is exhausted.
- `vtkStructuredPoints = obj.GetNextItem ()` - Get the next item in the collection. NULL is returned if the collection is exhausted.

31.199 vtkStructuredPointsSource

31.199.1 Usage

vtkStructuredPointsSource is an abstract class whose subclasses generate structured Points data.

To create an instance of class vtkStructuredPointsSource, simply invoke its constructor as follows

```
obj = vtkStructuredPointsSource
```

31.199.2 Methods

The class vtkStructuredPointsSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredPointsSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredPointsSource = obj.NewInstance ()`
- `vtkStructuredPointsSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutput (vtkStructuredPoints output)` - Set/Get the output of this source.
- `vtkStructuredPoints = obj.GetOutput ()` - Set/Get the output of this source.
- `vtkStructuredPoints = obj.GetOutput (int idx)` - Set/Get the output of this source.

31.200 vtkStructuredPointsToPolyDataFilter

31.200.1 Usage

vtkStructuredPointsToPolyDataFilter is an abstract filter class whose subclasses take on input structured points and generate polygonal data on output.

To create an instance of class vtkStructuredPointsToPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkStructuredPointsToPolyDataFilter
```

31.200.2 Methods

The class vtkStructuredPointsToPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredPointsToPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredPointsToPolyDataFilter = obj.NewInstance ()`
- `vtkStructuredPointsToPolyDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImageData input)` - Set / get the input data or filter.
- `vtkImageData = obj.GetInput ()` - Set / get the input data or filter.

31.201 vtkStructuredPointsToStructuredPointsFilter

31.201.1 Usage

vtkStructuredPointsToStructuredPointsFilter is an abstract filter class whose subclasses take on input structured points and generate structured points on output.

To create an instance of class vtkStructuredPointsToStructuredPointsFilter, simply invoke its constructor as follows

```
obj = vtkStructuredPointsToStructuredPointsFilter
```

31.201.2 Methods

The class vtkStructuredPointsToStructuredPointsFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredPointsToStructuredPointsFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredPointsToStructuredPointsFilter = obj.NewInstance ()`
- `vtkStructuredPointsToStructuredPointsFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImageData input)` - Set / get the input data or filter.
- `vtkImageData = obj.GetInput ()` - Set / get the input data or filter.

31.202 vtkStructuredPointsToUnstructuredGridFilter

31.202.1 Usage

vtkStructuredPointsToUnstructuredGridFilter is an abstract filter class whose subclasses take on input structured points and generate unstructured grid data on output.

To create an instance of class vtkStructuredPointsToUnstructuredGridFilter, simply invoke its constructor as follows

```
obj = vtkStructuredPointsToUnstructuredGridFilter
```

31.202.2 Methods

The class vtkStructuredPointsToUnstructuredGridFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredPointsToUnstructuredGridFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredPointsToUnstructuredGridFilter = obj.NewInstance ()`
- `vtkStructuredPointsToUnstructuredGridFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImageData input)` - Set / get the input data or filter.
- `vtkImageData = obj.GetInput ()` - Set / get the input data or filter.

31.203 vtkSuperquadric

31.203.1 Usage

vtkSuperquadric computes the implicit function and function gradient for a superquadric. vtkSuperquadric is a concrete implementation of vtkImplicitFunction. The superquadric is centered at Center and axes of rotation is along the y-axis. (Use the superclass' vtkImplicitFunction transformation matrix if necessary to reposition.) Roundness parameters (PhiRoundness and ThetaRoundness) control the shape of the superquadric. The Toroidal boolean controls whether a toroidal superquadric is produced. If so, the Thickness parameter controls the thickness of the toroid: 0 is the thinnest allowable toroid, and 1 has a minimum sized

hole. The Scale parameters allow the superquadric to be scaled in x, y, and z (normal vectors are correctly generated in any case). The Size parameter controls size of the superquadric.

This code is based on "Rigid physically based superquadrics", A. H. Barr, in "Graphics Gems III", David Kirk, ed., Academic Press, 1992.

To create an instance of class vtkSuperquadric, simply invoke its constructor as follows

```
obj = vtkSuperquadric
```

31.203.2 Methods

The class vtkSuperquadric has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSuperquadric class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSuperquadric = obj.NewInstance ()`
- `vtkSuperquadric = obj.SafeDownCast (vtkObject o)`
- `double = obj.EvaluateFunction (double x[3])`
- `double = obj.EvaluateFunction (double x, double y, double z)`
- `obj.EvaluateGradient (double x[3], double g[3])`
- `obj.SetCenter (double , double , double )` - Set the center of the superquadric. Default is 0,0,0.
- `obj.SetCenter (double  a[3])` - Set the center of the superquadric. Default is 0,0,0.
- `double = obj. GetCenter ()` - Set the center of the superquadric. Default is 0,0,0.
- `obj.SetScale (double , double , double )` - Set the scale factors of the superquadric. Default is 1,1,1.
- `obj.SetScale (double  a[3])` - Set the scale factors of the superquadric. Default is 1,1,1.
- `double = obj. GetScale ()` - Set the scale factors of the superquadric. Default is 1,1,1.
- `double = obj.GetThickness ()` - Set/Get Superquadric ring thickness (toroids only). Changing thickness maintains the outside diameter of the toroid.
- `obj.SetThickness (double )` - Set/Get Superquadric ring thickness (toroids only). Changing thickness maintains the outside diameter of the toroid.
- `double = obj.GetThicknessMinValue ()` - Set/Get Superquadric ring thickness (toroids only). Changing thickness maintains the outside diameter of the toroid.
- `double = obj.GetThicknessMaxValue ()` - Set/Get Superquadric ring thickness (toroids only). Changing thickness maintains the outside diameter of the toroid.
- `double = obj.GetPhiRoundness ()` - Set/Get Superquadric north/south roundness. Values range from 0 (rectangular) to 1 (circular) to higher orders.
- `obj.SetPhiRoundness (double e)` - Set/Get Superquadric north/south roundness. Values range from 0 (rectangular) to 1 (circular) to higher orders.
- `double = obj.GetThetaRoundness ()` - Set/Get Superquadric east/west roundness. Values range from 0 (rectangular) to 1 (circular) to higher orders.

- `obj.SetThetaRoundness (double e)` - Set/Get Superquadric east/west roundness. Values range from 0 (rectangular) to 1 (circular) to higher orders.
- `obj.SetSize (double )` - Set/Get Superquadric isotropic size.
- `double = obj.GetSize ()` - Set/Get Superquadric isotropic size.
- `obj.ToroidalOn ()` - Set/Get whether or not the superquadric is toroidal (1) or ellipsoidal (0).
- `obj.ToroidalOff ()` - Set/Get whether or not the superquadric is toroidal (1) or ellipsoidal (0).
- `int = obj.GetToroidal ()` - Set/Get whether or not the superquadric is toroidal (1) or ellipsoidal (0).
- `obj.SetToroidal (int )` - Set/Get whether or not the superquadric is toroidal (1) or ellipsoidal (0).

31.204 vtkTable

31.204.1 Usage

vtkTable is a basic data structure for storing columns of data. Internally, columns are stored in a vtkDataSetAttributes structure called RowData. However, using the vtkTable API additionally ensures that every column has the same number of entries, and provides row access (using vtkVariantArray) and single entry access (using vtkVariant).

The field data inherited from vtkDataObject may be used to store metadata related to the table.

To create an instance of class vtkTable, simply invoke its constructor as follows

```
obj = vtkTable
```

31.204.2 Methods

The class vtkTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTable = obj.NewInstance ()`
- `vtkTable = obj.SafeDownCast (vtkObject o)`
- `obj.Dump (int colWidth)` - Dump table contents.
- `int = obj.GetDataObjectType ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value).
- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value).
- `vtkDataSetAttributes = obj.GetRowData ()` - Get/Set the main data (columns) of the table.
- `obj.SetRowData (vtkDataSetAttributes data)` - Get/Set the main data (columns) of the table.
- `vtkIdType = obj.GetNumberOfRows ()` - Get the number of rows in the table.

- `obj.SetNumberOfRows (vtkIdType )` - Set the number of rows in the table. Note that memory allocation might be performed as a result of this, but no memory will be released.
- `vtkVariantArray = obj.GetRow (vtkIdType row)` - Get a row of the table as a vtkVariantArray which has one entry for each column. NOTE: This version of the method is NOT thread safe.
- `obj.GetRow (vtkIdType row, vtkVariantArray values)` - Get a row of the table as a vtkVariantArray which has one entry for each column.
- `obj.SetRow (vtkIdType row, vtkVariantArray values)` - Set a row of the table with a vtkVariantArray which has one entry for each column.
- `vtkIdType = obj.InsertNextBlankRow (double default\_num\_val)` - Insert a blank row at the end of the table.
- `vtkIdType = obj.InsertNextRow (vtkVariantArray arr)` - Insert a row specified by a vtkVariantArray. The number of entries in the array should match the number of columns in the table.
- `obj.RemoveRow (vtkIdType row)` - Delete a row from the table. Rows below the deleted row are shifted up.
- `vtkIdType = obj.GetNumberOfColumns ()` - Get the number of columns in the table.
- `string = obj.GetColumnName (vtkIdType col)`
- `vtkAbstractArray = obj.GetColumnByName (string name)` - Get a column of the table by its name.
- `vtkAbstractArray = obj.GetColumn (vtkIdType col)` - Get a column of the table by its column index.
- `obj.AddColumn (vtkAbstractArray arr)` - Add a column to the table.
- `obj.RemoveColumnByName (string name)` - Remove a column from the table by its name.
- `obj.RemoveColumn (vtkIdType col)` - Remove a column from the table by its column index.
- `obj.Initialize ()` - Initialize to an empty table.
- `obj.ShallowCopy (vtkDataObject src)` - Shallow/deep copy the data from src into this object.
- `obj.DeepCopy (vtkDataObject src)` - Shallow/deep copy the data from src into this object.
- `vtkFieldData = obj.GetAttributesAsFieldData (int type)` - Returns the attributes of the data object as a vtkFieldData. This returns non-null values in all the same cases as GetAttributes, in addition to the case of FIELD, which will return the field data for any vtkDataObject subclass.
- `vtkIdType = obj.GetNumberOfElements (int type)` - Get the number of elements for a specific attribute type (ROW, etc.).

31.205 vtkTableAlgorithm

31.205.1 Usage

vtkTableAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline architecture. There are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this class constructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be Tree. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into separate functions such as ExecuteData and ExecuteInformation.

For new algorithms you should implement RequestData(request, inputVec, outputVec) but for older filters there is a default implementation that calls the old ExecuteData(output) signature. For even older filters that don't implement ExecuteData the default implementation calls the even older Execute() signature.

.SECTION Thanks Thanks to Brian Wylie for creating this class.

To create an instance of class vtkTableAlgorithm, simply invoke its constructor as follows

```
obj = vtkTableAlgorithm
```

31.205.2 Methods

The class vtkTableAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableAlgorithm = obj.NewInstance ()`
- `vtkTableAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkTable = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkTable = obj.GetOutput (int index)` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int index, vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.206 vtkTemporalDataSet

31.206.1 Usage

vtkTemporalDataSet is a vtkCompositeDataSet that stores multiple time steps of data.

To create an instance of class vtkTemporalDataSet, simply invoke its constructor as follows

```
obj = vtkTemporalDataSet
```

31.206.2 Methods

The class vtkTemporalDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkTemporalDataSet = obj.NewInstance ()`
- `vtkTemporalDataSet = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Set the number of time steps in theis dataset
- `obj.SetNumberOfTimeSteps (int numLevels)` - Returns the number of time steps.
- `int = obj.GetNumberOfTimeSteps ()` - Set a data object as a timestep. Cannot be vtkTemporalDataSet.
- `obj.SetTimeStep (int timestep, vtkDataObject dobj)` - Set a data object as a timestep. Cannot be vtkTemporalDataSet.
- `vtkDataObject = obj.GetTimeStep (int timestep)` - Get timestep meta-data.
- `vtkInformation = obj.GetMetaData (int timestep)` - Returns if timestep meta-data is present.
- `int = obj.HasMetaData (int timestep)` - The extent type is a 3D extent
- `int = obj.GetExtentType ()` - The extent type is a 3D extent
- `vtkInformation = obj.GetMetaData (vtkCompositeDataIterator iter)` - Unhiding superclass method.
- `int = obj.HasMetaData (vtkCompositeDataIterator iter)`

31.207 vtkTemporalDataSetAlgorithm

31.207.1 Usage

Algorithms that take any type of data object (including composite dataset) and produce a vtkTemporalDataSet in the output can subclass from this class.

To create an instance of class vtkTemporalDataSetAlgorithm, simply invoke its constructor as follows

```
obj = vtkTemporalDataSetAlgorithm
```

31.207.2 Methods

The class vtkTemporalDataSetAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalDataSetAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTemporalDataSetAlgorithm = obj.NewInstance ()`
- `vtkTemporalDataSetAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkTemporalDataSet = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkTemporalDataSet = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.208 vtkTetra

31.208.1 Usage

vtkTetra is a concrete implementation of vtkCell to represent a 3D tetrahedron. vtkTetra uses the standard isoparametric shape functions for a linear tetrahedron. The tetrahedron is defined by the four points (0-3); where (0,1,2) is the base of the tetrahedron which, using the right hand rule, forms a triangle whose normal points in the direction of the fourth point.

To create an instance of class vtkTetra, simply invoke its constructor as follows

```
obj = vtkTetra
```

31.208.2 Methods

The class vtkTetra has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTetra class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTetra = obj.NewInstance ()`
- `vtkTetra = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - Returns the set of points that are on the boundary of the tetrahedron that are closest parametrically to the point specified. This may include faces, edges, or vertices.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the tetrahedron in parametric coordinates.

- `double = obj.GetParametricDistance (double pcoords[3])` - Return the distance of the parametric coordinate provided to the cell. If inside the cell, a distance of zero is returned.
- `obj.InterpolateFunctions (double pcoords[3], double weights[4])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[12])` - Return the ids of the vertices defining edge/face ('edgeId'/'faceId'). Ids are related to the cell, not to the dataset.

31.209 vtkThreadedImageAlgorithm

31.209.1 Usage

vtkThreadedImageAlgorithm is a filter superclass that hides much of the pipeline complexity. It handles breaking the pipeline execution into smaller extents so that the vtkImageData limits are observed. It also provides support for multithreading. If you don't need any of this functionality, consider using vtkSimpleImageToImageAlgorithm instead. .SECTION See also vtkSimpleImageToImageAlgorithm

To create an instance of class vtkThreadedImageAlgorithm, simply invoke its constructor as follows

```
obj = vtkThreadedImageAlgorithm
```

31.209.2 Methods

The class vtkThreadedImageAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkThreadedImageAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkThreadedImageAlgorithm = obj.NewInstance ()`
- `vtkThreadedImageAlgorithm = obj.SafeDownCast (vtkObject o)`
- `obj.ThreadedExecute (vtkImageData inData, vtkImageData outData, int extent[6], int threadId)`
- `obj.SetNumberOfThreads (int )` - Get/Set the number of threads to create when rendering
- `int = obj.GetNumberOfThreadsMinValue ()` - Get/Set the number of threads to create when rendering
- `int = obj.GetNumberOfThreadsMaxValue ()` - Get/Set the number of threads to create when rendering
- `int = obj.GetNumberOfThreads ()` - Get/Set the number of threads to create when rendering
- `int = obj.SplitExtent (int splitExt[6], int startExt[6], int num, int total)` - Putting this here until I merge graphics and imaging streaming.

31.210 vtkTree

31.210.1 Usage

vtkTree is a connected directed graph with no cycles. A tree is a type of directed graph, so works with all graph algorithms.

vtkTree is a read-only data structure. To construct a tree, create an instance of vtkMutableDirectedGraph. Add vertices and edges with AddVertex() and AddEdge(). You may alternately start by adding a single vertex as the root then call graph-¿AddChild(parent) which adds a new vertex and connects the parent to the child. The tree MUST have all edges in the proper direction, from parent to child. After building the tree, call tree-¿CheckedShallowCopy(graph) to copy the structure into a vtkTree. This method will return false if the graph is an invalid tree.

vtkTree provides some convenience methods for obtaining the parent and children of a vertex, for finding the root, and determining if a vertex is a leaf (a vertex with no children).

To create an instance of class vtkTree, simply invoke its constructor as follows

```
obj = vtkTree
```

31.210.2 Methods

The class vtkTree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTree class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTree = obj.NewInstance ()`
- `vtkTree = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Get the root vertex of the tree.
- `vtkIdType = obj.GetRoot ()` - Get the root vertex of the tree.
- `vtkIdType = obj.GetNumberOfChildren (vtkIdType v)` - Get the i-th child of a parent vertex.
- `vtkIdType = obj.GetChild (vtkIdType v, vtkIdType i)` - Get the i-th child of a parent vertex.
- `obj.GetChildren (vtkIdType v, vtkAdjacentVertexIterator it)` - Get the parent of a vertex.
- `vtkIdType = obj.GetParent (vtkIdType v)` - Get the parent of a vertex.
- `vtkIdType = obj.GetLevel (vtkIdType v)` - Get the level of the vertex in the tree. The root vertex has level 0. Returns -1 if the vertex id is ¡ 0 or greater than the number of vertices in the tree.
- `bool = obj.IsLeaf (vtkIdType vertex)` - Return whether the vertex is a leaf (i.e. it has no children).
- `obj.ReorderChildren (vtkIdType parent, vtkIdTypeArray children)` - Reorder the children of a parent vertex. The children array must contain all the children of parent, just in a different order. This does not change the topology of the tree.

31.211 vtkTreeAlgorithm

31.211.1 Usage

vtkTreeAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline edgehitecture. There are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this class constructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be Tree. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into separate functions such as ExecuteData and ExecuteInformation.

For new algorithms you should implement RequestData(request, inputVec, outputVec) but for older filters there is a default implementation that calls the old ExecuteData(output) signature. For even older filters that don't implement ExecuteData the default implementation calls the even older Execute() signature.

To create an instance of class vtkTreeAlgorithm, simply invoke its constructor as follows

```
obj = vtkTreeAlgorithm
```

31.211.2 Methods

The class vtkTreeAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeAlgorithm = obj.NewInstance ()`
- `vtkTreeAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkTree = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkTree = obj.GetOutput (int index)` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int index, vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.212 vtkTreeDFSIterator

31.212.1 Usage

vtkTreeDFSIterator performs a depth first seedgeh of a tree. First, you must set the tree on which you are going to iterate, and set the starting vertex and mode. The mode is either DISCOVER, in which case vertices are visited as they are first reached, or FINISH, in which case vertices are visited when they are done, i.e. all adjacent vertices have been discovered already.

After setting up the iterator, the normal mode of operation is to set up a ¡code¿while(iter-¿HasNext())¡/code¿ loop, with the statement ¡code¿vtkIdType vertex = iter-¿Next()¡/code¿ inside the loop.

To create an instance of class vtkTreeDFSIterator, simply invoke its constructor as follows

```
obj = vtkTreeDFSIterator
```

31.212.2 Methods

The class vtkTreeDFSIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeDFSIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeDFSIterator = obj.NewInstance ()`
- `vtkTreeDFSIterator = obj.SafeDownCast (vtkObject o)`
- `obj.SetTree (vtkTree graph)` - Set the graph to iterate over.
- `obj.SetMode (int mode)` - Set the visit mode of the iterator. Mode can be DISCOVER (0): Order by discovery time FINISH (1): Order by finish time Default is DISCOVER. Use DISCOVER for top-down algorithms where parents need to be processed before children. Use FINISH for bottom-up algorithms where children need to be processed before parents.
- `int = obj.GetMode ()` - Set the visit mode of the iterator. Mode can be DISCOVER (0): Order by discovery time FINISH (1): Order by finish time Default is DISCOVER. Use DISCOVER for top-down algorithms where parents need to be processed before children. Use FINISH for bottom-up algorithms where children need to be processed before parents.
- `obj.SetStartVertex (vtkIdType vertex)` - The start vertex of the seedgeh. The tree iterator will only iterate over the subtree rooted at vertex. If not set (or set to a negative value), starts at the root of the tree.
- `vtkIdType = obj.GetStartVertex ()` - The start vertex of the seedgeh. The tree iterator will only iterate over the subtree rooted at vertex. If not set (or set to a negative value), starts at the root of the tree.
- `vtkIdType = obj.Next ()` - The next vertex visited in the graph.
- `bool = obj.HasNext ()` - Return true when all vertices have been visited.

31.213 vtkTriangle

31.213.1 Usage

vtkTriangle is a concrete implementation of vtkCell to represent a triangle located in 3-space.

To create an instance of class vtkTriangle, simply invoke its constructor as follows

```
obj = vtkTriangle
```

31.213.2 Methods

The class vtkTriangle has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTriangle class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTriangle = obj.NewInstance ()`

- `vtkTriangle = obj.SafeDownCast (vtkObject o)`
- `vtkCell = obj.GetEdge (int edgeId)` - Get the edge specified by edgeId (range 0 to 2) and return that edge's coordinates.
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `double = obj.ComputeArea ()` - A convenience function to compute the area of a vtkTriangle.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this triangle using scalar value provided. Like contouring, except that it cuts the triangle to produce other triangles.
- `obj.InterpolateFunctions (double pcoords[3], double sf[3])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[6])` - Return the ids of the vertices defining edge ('edgeId'). Ids are related to the cell, not to the dataset.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the triangle in parametric coordinates.
- `double = obj.GetParametricDistance (double pcoords[3])` - Return the distance of the parametric coordinate provided to the cell. If inside the cell, a distance of zero is returned.

31.214 vtkTriangleStrip

31.214.1 Usage

vtkTriangleStrip is a concrete implementation of vtkCell to represent a 2D triangle strip. A triangle strip is a compact representation of triangles connected edge to edge in strip fashion. The connectivity of a triangle strip is three points defining an initial triangle, then for each additional triangle, a single point that, combined with the previous two points, defines the next triangle.

To create an instance of class vtkTriangleStrip, simply invoke its constructor as follows

```
obj = vtkTriangleStrip
```

31.214.2 Methods

The class vtkTriangleStrip has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTriangleStrip class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTriangleStrip = obj.NewInstance ()`
- `vtkTriangleStrip = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `int = obj.IsPrimaryCell ()` - Return the center of the point cloud in parametric coordinates.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the point cloud in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs)` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)

31.215 vtkTriQuadraticHexahedron

31.215.1 Usage

vtkTriQuadraticHexahedron is a concrete implementation of vtkNonLinearCell to represent a three-dimensional, 27-node isoparametric triquadratic hexahedron. The interpolation is the standard finite element, triquadratic isoparametric shape function. The cell includes 8 edge nodes, 12 mid-edge nodes, 6 mid-face nodes and one mid-volume node. The ordering of the 27 points defining the cell is point ids (0-7,8-19, 20-25, 26) where point ids 0-7 are the eight corner vertices of the cube; followed by twelve midedge nodes (8-19); followed by 6 mid-face nodes (20-25) and the last node (26) is the mid-volume node. Note that these midedge

nodes correspond lie on the edges defined by (0,1), (1,2), (2,3), (3,0), (4,5), (5,6), (6,7), (7,4), (0,4), (1,5), (2,6), (3,7). The mid-surface nodes lies on the faces defined by (first edge nodes id's, than mid-edge nodes id's): (0,1,5,4;8,17,12,16), (1,2,6,5;9,18,13,17), (2,3,7,6,10,19,14,18), (3,0,4,7;11,16,15,19), (0,1,2,3;8,9,10,11), (4,5,6,7;12,13,14,15). The last point lies in the center of the cell (0,1,2,3,4,5,6,7).

```
top
 7--14--6
 |      |
15  25  13
 |      |
 4--12--5

 middle
19--23--18
 |      |
20  26  21
 |      |
16--22--17

bottom
 3--10--2
 |      |
11  24  9
 |      |
 0-- 8--1
```

To create an instance of class vtkTriQuadraticHexahedron, simply invoke its constructor as follows

```
obj = vtkTriQuadraticHexahedron
```

31.215.2 Methods

The class vtkTriQuadraticHexahedron has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTriQuadraticHexahedron class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTriQuadraticHexahedron = obj.NewInstance ()`
- `vtkTriQuadraticHexahedron = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.

- `int = obj.GetNumberOfFaces ()` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - Implement the vtkCell API. See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)`
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr`
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)`
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)`
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - Clip this triquadratic hexahedron using scalar value provided. Like contouring, except that it cuts the hex to produce linear tetrahedron.
- `obj.InterpolateFunctions (double pcoords[3], double weights[27])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[81])` - Return the ids of the vertices defining edge/face ('edgeId'/'faceId'). Ids are related to the cell, not to the dataset.

31.216 vtkTrivialProducer

31.216.1 Usage

vtkTrivialProducer allows stand-alone data objects to be connected as inputs in a pipeline. All data objects that are connected to a pipeline involving vtkAlgorithm must have a producer. This trivial producer allows data objects that are hand-constructed in a program without another vtk producer to be connected.

To create an instance of class vtkTrivialProducer, simply invoke its constructor as follows

```
obj = vtkTrivialProducer
```

31.216.2 Methods

The class vtkTrivialProducer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTrivialProducer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTrivialProducer = obj.NewInstance ()`
- `vtkTrivialProducer = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutput (vtkDataObject output)` - Set the data object that is "produced" by this producer. It is never really modified.
- `long = obj.GetMTime ()` - The modified time of this producer is the newer of this object or the assigned output.

31.217 vtkUndirectedGraph

31.217.1 Usage

vtkUndirectedGraph is a collection of vertices along with a collection of undirected edges (they connect two vertices in no particular order). ShallowCopy(), DeepCopy(), CheckedShallowCopy(), CheckedDeepCopy() accept instances of vtkUndirectedGraph and vtkMutableUndirectedGraph. GetOutEdges(v, it) and GetInEdges(v, it) return the same list of edges, which is the list of all edges which have a v as an endpoint. GetInDegree(v), GetOutDegree(v) and GetDegree(v) all return the full degree of vertex v.

vtkUndirectedGraph is read-only. To create an undirected graph, use an instance of vtkMutableUndirectedGraph, then you may set the structure to a vtkUndirectedGraph using ShallowCopy().

To create an instance of class vtkUndirectedGraph, simply invoke its constructor as follows

```
obj = vtkUndirectedGraph
```

31.217.2 Methods

The class vtkUndirectedGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUndirectedGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUndirectedGraph = obj.NewInstance ()`
- `vtkUndirectedGraph = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Returns the full degree of the vertex.
- `vtkIdType = obj.GetInDegree (vtkIdType v)` - Returns the full degree of the vertex.
- `obj.GetInEdge (vtkIdType v, vtkIdType i, vtkGraphEdge e)` - Initialize the iterator to get the incoming edges to a vertex. For an undirected graph, this is all incident edges.
- `obj.GetInEdges (vtkIdType v, vtkInEdgeIterator it)`

31.218 vtkUndirectedGraphAlgorithm

31.218.1 Usage

vtkUndirectedGraphAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline edgehitecture. There are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this class constructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be Graph. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into separate functions such as ExecuteData and ExecuteInformation. For new algorithms you should implement RequestData(request, inputVec, outputVec) but for older filters there is a default implementation that calls the old ExecuteData(output) signature. For even older filters that don't implement ExecuteData the default implementation calls the even older Execute() signature.

.SECTION Thanks Thanks to Patricia Crossno, Ken Moreland, Andrew Wilson and Brian Wylie from Sandia National Laboratories for their help in developing this class.

To create an instance of class vtkUndirectedGraphAlgorithm, simply invoke its constructor as follows

```
obj = vtkUndirectedGraphAlgorithm
```

31.218.2 Methods

The class vtkUndirectedGraphAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUndirectedGraphAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUndirectedGraphAlgorithm = obj.NewInstance ()`
- `vtkUndirectedGraphAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkUndirectedGraph = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkUndirectedGraph = obj.GetOutput (int index)` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int index, vtkDataObject obj)` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

31.219 vtkUniformGrid

31.219.1 Usage

vtkUniformGrid is a subclass of vtkImageData. In addition to all the image data functionality, it supports blanking.

To create an instance of class vtkUniformGrid, simply invoke its constructor as follows

```
obj = vtkUniformGrid
```

31.219.2 Methods

The class vtkUniformGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUniformGrid class.

- `string = obj.GetClassName ()` - Construct an empty uniform grid.
- `int = obj.IsA (string name)` - Construct an empty uniform grid.
- `vtkUniformGrid = obj.NewInstance ()` - Construct an empty uniform grid.
- `vtkUniformGrid = obj.SafeDownCast (vtkObject o)` - Construct an empty uniform grid.
- `obj.CopyStructure (vtkDataSet ds)` - Copy the geometric and topological structure of an input image data object.

- `int = obj.GetDataObjectType ()` - Return what type of dataset this is.
- `vtkCell = obj.GetCell (vtkIdType cellId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCell (vtkIdType cellId, vtkGenericCell cell)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `int = obj.GetCellType (vtkIdType cellId)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetCellPoints (vtkIdType cellId, vtkIdList ptIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.GetPointCells (vtkIdType ptId, vtkIdList cellIds)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.Initialize ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `int = obj.GetMaxCellSize ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.BlankPoint (vtkIdType ptId)` - Methods for supporting blanking of cells. Blanking turns on or off points in the structured grid, and hence the cells connected to them. These methods should be called only after the dimensions of the grid are set.
- `obj.UnBlankPoint (vtkIdType ptId)` - Methods for supporting blanking of cells. Blanking turns on or off points in the structured grid, and hence the cells connected to them. These methods should be called only after the dimensions of the grid are set.
- `obj.BlankCell (vtkIdType ptId)` - Methods for supporting blanking of cells. Blanking turns on or off cells in the structured grid. These methods should be called only after the dimensions of the grid are set.
- `obj.UnBlankCell (vtkIdType ptId)` - Methods for supporting blanking of cells. Blanking turns on or off cells in the structured grid. These methods should be called only after the dimensions of the grid are set.
- `vtkUnsignedCharArray = obj.GetPointVisibilityArray ()` - Get the array that defines the blanking (visibility) of each point.
- `obj.SetPointVisibilityArray (vtkUnsignedCharArray pointVisibility)` - Set an array that defines the (blanking) visibility of the points in the grid. Make sure that length of the visibility array matches the number of points in the grid.
- `vtkUnsignedCharArray = obj.GetCellVisibilityArray ()` - Get the array that defines the blanking (visibility) of each cell.
- `obj.SetCellVisibilityArray (vtkUnsignedCharArray pointVisibility)` - Set an array that defines the (blanking) visibility of the cells in the grid. Make sure that length of the visibility array matches the number of points in the grid.
- `char = obj.IsPointVisible (vtkIdType ptId)` - Return non-zero value if specified point is visible. These methods should be called only after the dimensions of the grid are set.
- `char = obj.IsCellVisible (vtkIdType cellId)` - Return non-zero value if specified cell is visible. These methods should be called only after the dimensions of the grid are set.

- `char = obj.GetPointBlanking ()` - Returns 1 if there is any visibility constraint on the points, 0 otherwise.
- `char = obj.GetCellBlanking ()` - Returns 1 if there is any visibility constraint on the cells, 0 otherwise.
- `vtkImageData = obj.NewImageDataCopy ()`

31.220 vtkUnstructuredGrid

31.220.1 Usage

vtkUnstructuredGrid is a data object that is a concrete implementation of vtkDataSet. vtkUnstructuredGrid represents any combinations of any cell types. This includes 0D (e.g., points), 1D (e.g., lines, polylines), 2D (e.g., triangles, polygons), and 3D (e.g., hexahedron, tetrahedron).

To create an instance of class vtkUnstructuredGrid, simply invoke its constructor as follows

```
obj = vtkUnstructuredGrid
```

31.220.2 Methods

The class vtkUnstructuredGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGrid = obj.NewInstance ()`
- `vtkUnstructuredGrid = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataObjectType ()` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `obj.Allocate (vtkIdType numCells, int extSize)` - Standard vtkDataSet API methods. See vtkDataSet for more information.
- `vtkIdType = obj.InsertNextCell (int type, vtkIdList ptIds)` - Insert/create cell in object by type and list of point ids defining cell topology.
- `obj.Reset ()`
- `obj.CopyStructure (vtkDataSet ds)`
- `vtkIdType = obj.GetNumberOfCells ()`
- `vtkCell = obj.GetCell (vtkIdType cellId)`
- `obj.GetCell (vtkIdType cellId, vtkGenericCell cell)`
- `obj.GetCellBounds (vtkIdType cellId, double bounds[6])`
- `obj.GetCellPoints (vtkIdType cellId, vtkIdList ptIds)`
- `obj.GetPointCells (vtkIdType ptId, vtkIdList cellIds)`
- `int = obj.GetCellType (vtkIdType cellId)`

- `vtkUnsignedCharArray = obj.GetCellTypesArray ()`
- `vtkIdTypeArray = obj.GetCellLocationsArray ()`
- `obj.Squeeze ()`
- `obj.Initialize ()`
- `int = obj.GetMaxCellSize ()`
- `obj.BuildLinks ()`
- `vtkCellLinks = obj.GetCellLinks ()`
- `obj.SetCells (int type, vtkCellArray cells)` - Special methods specific to vtkUnstructuredGrid for defining the cells composing the dataset.
- `obj.SetCells (int types, vtkCellArray cells)` - Special methods specific to vtkUnstructuredGrid for defining the cells composing the dataset.
- `obj.SetCells (vtkUnsignedCharArray cellTypes, vtkIdTypeArray cellLocations, vtkCellArray cells)` - Special methods specific to vtkUnstructuredGrid for defining the cells composing the dataset.
- `vtkCellArray = obj.GetCells ()` - Special methods specific to vtkUnstructuredGrid for defining the cells composing the dataset.
- `obj.RemoveReferenceToCell (vtkIdType ptId, vtkIdType cellId)` - Special methods specific to vtkUnstructuredGrid for defining the cells composing the dataset.
- `obj.AddReferenceToCell (vtkIdType ptId, vtkIdType cellId)` - Special methods specific to vtkUnstructuredGrid for defining the cells composing the dataset.
- `obj.ResizeCellList (vtkIdType ptId, int size)` - Special methods specific to vtkUnstructuredGrid for defining the cells composing the dataset.
- `obj.GetCellNeighbors (vtkIdType cellId, vtkIdList ptIds, vtkIdList cellIds)` - Topological inquiry to get all cells using list of points exclusive of cell specified (e.g., cellId). THIS METHOD IS THREAD SAFE IF FIRST CALLED FROM A SINGLE THREAD AND THE DATASET IS NOT MODIFIED
- `int = obj.GetUpdateExtent ()` - We need this here to avoid hiding superclass method
- `obj.GetUpdateExtent (int extent[6])` - We need this here to avoid hiding superclass method
- `int = obj.GetPiece ()` - Set / Get the piece and the number of pieces. Similar to extent in 3D.
- `int = obj.GetNumberOfPieces ()` - Set / Get the piece and the number of pieces. Similar to extent in 3D.
- `int = obj.GetGhostLevel ()` - Get the ghost level.
- `long = obj.GetActualMemorySize ()` - Return the actual size of the data in kilobytes. This number is valid only after the pipeline has updated. The memory size returned is guaranteed to be greater than or equal to the memory required to represent the data (e.g., extra space in arrays, etc. are not included in the return value). THIS METHOD IS THREAD SAFE.
- `obj.ShallowCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkDataObject src)` - Shallow and Deep copy.
- `obj.GetIdsOfCellsOfType (int type, vtkIdTypeArray array)` - Fill vtkIdTypeArray container with list of cell Ids. This method traverses all cells and, for a particular cell type, inserts the cell Id into the container.

- `int = obj.IsHomogeneous ()` - Traverse cells and determine if cells are all of the same type.
- `obj.RemoveGhostCells (int level)` - This method will remove any cell that has a ghost level array value greater or equal to level.

31.221 vtkUnstructuredGridAlgorithm

31.221.1 Usage

To create an instance of class vtkUnstructuredGridAlgorithm, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridAlgorithm
```

31.221.2 Methods

The class vtkUnstructuredGridAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridAlgorithm = obj.NewInstance ()`
- `vtkUnstructuredGridAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkUnstructuredGrid = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkUnstructuredGrid = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput (int port)`
- `vtkDataObject = obj.GetInput ()`
- `vtkUnstructuredGrid = obj.GetUnstructuredGridInput (int port)`
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

31.222 vtkUnstructuredGridSource

31.222.1 Usage

vtkUnstructuredGridSource is an abstract class whose subclasses generate unstructured grid data.

To create an instance of class vtkUnstructuredGridSource, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridSource
```

31.222.2 Methods

The class vtkUnstructuredGridSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridSource = obj.NewInstance ()`
- `vtkUnstructuredGridSource = obj.SafeDownCast (vtkObject o)`
- `vtkUnstructuredGrid = obj.GetOutput ()` - Get the output of this source.
- `vtkUnstructuredGrid = obj.GetOutput (int idx)` - Get the output of this source.
- `obj.SetOutput (vtkUnstructuredGrid output)` - Get the output of this source.

31.223 vtkUnstructuredGridToPolyDataFilter

31.223.1 Usage

vtkUnstructuredGridToPolyDataFilter is an abstract filter class whose subclasses take as input datasets of type vtkUnstructuredGrid and generate polygonal data on output.

To create an instance of class vtkUnstructuredGridToPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridToPolyDataFilter
```

31.223.2 Methods

The class vtkUnstructuredGridToPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridToPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridToPolyDataFilter = obj.NewInstance ()`
- `vtkUnstructuredGridToPolyDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkUnstructuredGrid input)` - Set / get the input data or filter.
- `vtkUnstructuredGrid = obj.GetInput ()` - Set / get the input data or filter.
- `obj.ComputeInputUpdateExtents (vtkDataObject output)` - Do not let datasets return more than requested.

31.224 vtkUnstructuredGridToUnstructuredGridFilter

31.224.1 Usage

To create an instance of class vtkUnstructuredGridToUnstructuredGridFilter, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridToUnstructuredGridFilter
```

31.224.2 Methods

The class vtkUnstructuredGridToUnstructuredGridFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridToUnstructuredGridFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridToUnstructuredGridFilter = obj.NewInstance ()`
- `vtkUnstructuredGridToUnstructuredGridFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkUnstructuredGrid input)` - Set / get the input Grid or filter.
- `vtkUnstructuredGrid = obj.GetInput ()` - Set / get the input Grid or filter.
- `int = obj.FillInputPortInformation (int , vtkInformation )`

31.225 vtkVertex

31.225.1 Usage

vtkVertex is a concrete implementation of vtkCell to represent a 3D point.

To create an instance of class vtkVertex, simply invoke its constructor as follows

```
obj = vtkVertex
```

31.225.2 Methods

The class vtkVertex has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVertex class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVertex = obj.NewInstance ()`
- `vtkVertex = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.

- `vtkCell = obj.GetEdge (int )` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int )` - See the vtkCell API for descriptions of these methods.
- `obj.Clip (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArray` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - Given parametric coordinates of a point, return the closest cell boundary, and whether the point is inside or outside of the cell. The cell boundary is defined by a list of points (pts) that specify a vertex (1D cell). If the return value of the method is != 0, then the point is inside the cell.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellArr` - Generate contouring primitives. The scalar list cellScalars are scalar values at each cell point. The point locator is essentially a points list that merges points as they are inserted (i.e., prevents duplicates).
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the triangle in parametric coordinates.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - Triangulate the vertex. This method fills pts and ptIds with information from the only point in the vertex.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - Get the derivative of the vertex. Returns (0.0, 0.0, 0.0) for all dimensions.
- `obj.InterpolateFunctions (double pcoords[3], double weights[1])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[3])`

31.226 vtkVertexListIterator

31.226.1 Usage

vtkVertexListIterator iterates through all vertices in a graph. Create an instance of this and call graph-¿GetVertices(it) to initialize this iterator. You may alternately call SetGraph() to initialize the iterator.

To create an instance of class vtkVertexListIterator, simply invoke its constructor as follows

```
obj = vtkVertexListIterator
```

31.226.2 Methods

The class vtkVertexListIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVertexListIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVertexListIterator = obj.NewInstance ()`
- `vtkVertexListIterator = obj.SafeDownCast (vtkObject o)`
- `obj.SetGraph (vtkGraph graph)` - Setup the iterator with a graph.
- `vtkGraph = obj.GetGraph ()` - Get the graph associated with this iterator.
- `vtkIdType = obj.Next ()` - Whether this iterator has more edges.
- `bool = obj.HasNext ()`

31.227 vtkViewDependentErrorMetric

31.227.1 Usage

It is a concrete error metric, based on a geometric criterium in the screen space: the variation of the projected edge from a projected straight line

To create an instance of class vtkViewDependentErrorMetric, simply invoke its constructor as follows

```
obj = vtkViewDependentErrorMetric
```

31.227.2 Methods

The class vtkViewDependentErrorMetric has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkViewDependentErrorMetric class.

- `string = obj.GetClassName ()` - Standard VTK type and error macros.
- `int = obj.IsA (string name)` - Standard VTK type and error macros.
- `vtkViewDependentErrorMetric = obj.NewInstance ()` - Standard VTK type and error macros.
- `vtkViewDependentErrorMetric = obj.SafeDownCast (vtkObject o)` - Standard VTK type and error macros.
- `double = obj.GetPixelTolerance ()` - Return the squared screen-based geometric accurary measured in pixels. An accuracy less or equal to 0.25 ($0.5\hat{2}$) ensures that the screen-space interpolation of a mid-point matchs exactly with the projection of the mid-point (a value less than 1 but greater than 0.25 is not enough, because of 8-neighbors). Maybe it is useful for lower accuracy in case of anti-aliasing?
- `obj.SetPixelTolerance (double value)` - Set the squared screen-based geometric accuracy measured in pixels. Subdivision will be required if the square distance between the projection of the real point and the straight line passing through the projection of the vertices of the edge is greater than 'value'. For instance, 0.25 will give better result than 1.
- `vtkViewport = obj.GetViewport ()` - Set/Get the renderer with 'renderer' on which the error metric is based. The error metric use the active camera of the renderer.
- `obj.SetViewport (vtkViewport viewport)` - Set/Get the renderer with 'renderer' on which the error metric is based. The error metric use the active camera of the renderer.
- `int = obj.RequiresEdgeSubdivision (double leftPoint, double midPoint, double rightPoint, double alp` - Does the edge need to be subdivided according to the distance between the line passing through its endpoints in screen space and the projection of its mid point? The edge is defined by its 'leftPoint' and its 'rightPoint'. 'leftPoint', 'midPoint' and 'rightPoint' have to be initialized before calling RequiresEdgeSubdivision(). Their format is global coordinates, parametric coordinates and point centered attributes: xyx rst abc de... 'alpha' is the normalized abscissa of the midpoint along the edge. (close to 0 means close to the left point, close to 1 means close to the right point)

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6
- `double = obj.GetError (double leftPoint, double midPoint, double rightPoint, double alpha)` - Return the error at the mid-point. The type of error depends on the state of the concrete error metric. For instance, it can return an absolute or relative error metric. See RequiresEdgeSubdivision() for a description of the arguments.

  =GetAttributeCollection()-¿GetNumberOfPointCenteredComponents()+6

31.228 vtkViewport

31.228.1 Usage

vtkViewport provides an abstract specification for Viewports. A Viewport is an object that controls the rendering process for objects. Rendering is the process of converting geometry, a specification for lights, and a camera view into an image. vtkViewport also performs coordinate transformation between world coordinates, view coordinates (the computer graphics rendering coordinate system), and display coordinates (the actual screen coordinates on the display device). Certain advanced rendering features such as two-sided lighting can also be controlled.

To create an instance of class vtkViewport, simply invoke its constructor as follows

```
obj = vtkViewport
```

31.228.2 Methods

The class vtkViewport has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkViewport class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkViewport = obj.NewInstance ()`
- `vtkViewport = obj.SafeDownCast (vtkObject o)`
- `obj.AddViewProp (vtkProp )` - Add a prop to the list of props. Prop is the superclass of all actors, volumes, 2D actors, composite props etc.
- `vtkPropCollection = obj.GetViewProps ()` - Return any props in this viewport.
- `int = obj.HasViewProp (vtkProp )` - Query if a prop is in the list of props.
- `obj.RemoveViewProp (vtkProp )` - Remove an actor from the list of actors.
- `obj.RemoveAllViewProps (void )` - Remove all actors from the list of actors.
- `obj.AddActor2D (vtkProp p)` - Add/Remove different types of props to the renderer. These methods are all synonyms to AddViewProp and RemoveViewProp. They are here for convenience and backwards compatibility.
- `obj.RemoveActor2D (vtkProp p)` - Add/Remove different types of props to the renderer. These methods are all synonyms to AddViewProp and RemoveViewProp. They are here for convenience and backwards compatibility.
- `vtkActor2DCollection = obj.GetActors2D ()` - Add/Remove different types of props to the renderer. These methods are all synonyms to AddViewProp and RemoveViewProp. They are here for convenience and backwards compatibility.
- `obj.SetBackground (double , double , double )` - Set/Get the background color of the rendering screen using an rgb color specification.
- `obj.SetBackground (double  a[3])` - Set/Get the background color of the rendering screen using an rgb color specification.
- `double = obj. GetBackground ()` - Set/Get the background color of the rendering screen using an rgb color specification.

- `obj.SetBackground2 (double , double , double )` - Set/Get the second background color of the rendering screen for gradient backgrounds using an rgb color specification.
- `obj.SetBackground2 (double a[3])` - Set/Get the second background color of the rendering screen for gradient backgrounds using an rgb color specification.
- `double = obj. GetBackground2 ()` - Set/Get the second background color of the rendering screen for gradient backgrounds using an rgb color specification.
- `obj.SetGradientBackground (bool )` - Set/Get whether this viewport should have a gradient background using the Background (top) and Background2 (bottom) colors. Default is off.
- `bool = obj.GetGradientBackground ()` - Set/Get whether this viewport should have a gradient background using the Background (top) and Background2 (bottom) colors. Default is off.
- `obj.GradientBackgroundOn ()` - Set/Get whether this viewport should have a gradient background using the Background (top) and Background2 (bottom) colors. Default is off.
- `obj.GradientBackgroundOff ()` - Set/Get whether this viewport should have a gradient background using the Background (top) and Background2 (bottom) colors. Default is off.
- `obj.SetAspect (double , double )` - Set the aspect ratio of the rendered image. This is computed automatically and should not be set by the user.
- `obj.SetAspect (double a[2])` - Set the aspect ratio of the rendered image. This is computed automatically and should not be set by the user.
- `double = obj. GetAspect ()` - Set the aspect ratio of the rendered image. This is computed automatically and should not be set by the user.
- `obj.ComputeAspect ()` - Set the aspect ratio of the rendered image. This is computed automatically and should not be set by the user.
- `obj.SetPixelAspect (double , double )` - Set the aspect ratio of a pixel in the rendered image. This factor permits the image to rendered anisotropically (i.e., stretched in one direction or the other).
- `obj.SetPixelAspect (double a[2])` - Set the aspect ratio of a pixel in the rendered image. This factor permits the image to rendered anisotropically (i.e., stretched in one direction or the other).
- `double = obj. GetPixelAspect ()` - Set the aspect ratio of a pixel in the rendered image. This factor permits the image to rendered anisotropically (i.e., stretched in one direction or the other).
- `obj.SetViewport (double , double , double , double )` - Specify the viewport for the Viewport to draw in the rendering window. Coordinates are expressed as (xmin,ymin,xmax,ymax), where each coordinate is 0 ¡= coordinate ¡= 1.0.
- `obj.SetViewport (double a[4])` - Specify the viewport for the Viewport to draw in the rendering window. Coordinates are expressed as (xmin,ymin,xmax,ymax), where each coordinate is 0 ¡= coordinate ¡= 1.0.
- `double = obj. GetViewport ()` - Specify the viewport for the Viewport to draw in the rendering window. Coordinates are expressed as (xmin,ymin,xmax,ymax), where each coordinate is 0 ¡= coordinate ¡= 1.0.
- `obj.SetDisplayPoint (double , double , double )` - Set/get a point location in display (or screen) coordinates. The lower left corner of the window is the origin and y increases as you go up the screen.
- `obj.SetDisplayPoint (double a[3])` - Set/get a point location in display (or screen) coordinates. The lower left corner of the window is the origin and y increases as you go up the screen.

- `double = obj. GetDisplayPoint ()` - Set/get a point location in display (or screen) coordinates. The lower left corner of the window is the origin and y increases as you go up the screen.
- `obj.SetViewPoint (double , double , double )` - Specify a point location in view coordinates. The origin is in the middle of the viewport and it extends from -1 to 1 in all three dimensions.
- `obj.SetViewPoint (double a[3])` - Specify a point location in view coordinates. The origin is in the middle of the viewport and it extends from -1 to 1 in all three dimensions.
- `double = obj. GetViewPoint ()` - Specify a point location in view coordinates. The origin is in the middle of the viewport and it extends from -1 to 1 in all three dimensions.
- `obj.SetWorldPoint (double , double , double , double )` - Specify a point location in world coordinates. This method takes homogeneous coordinates.
- `obj.SetWorldPoint (double a[4])` - Specify a point location in world coordinates. This method takes homogeneous coordinates.
- `double = obj. GetWorldPoint ()` - Specify a point location in world coordinates. This method takes homogeneous coordinates.
- `double = obj.GetCenter ()` - Return the center of this viewport in display coordinates.
- `int = obj.IsInViewport (int x, int y)` - Is a given display point in this Viewport's viewport.
- `vtkWindow = obj.GetVTKWindow ()` - Return the vtkWindow that owns this vtkViewport.
- `obj.DisplayToView ()` - Convert display coordinates to view coordinates.
- `obj.ViewToDisplay ()` - Convert view coordinates to display coordinates.
- `obj.WorldToView ()` - Convert world point coordinates to view coordinates.
- `obj.ViewToWorld ()` - Convert view point coordinates to world coordinates.
- `obj.DisplayToWorld ()` - Convert display (or screen) coordinates to world coordinates.
- `obj.WorldToDisplay ()` - Convert world point coordinates to display (or screen) coordinates.
- `int = obj.GetSize ()` - Get the size and origin of the viewport in display coordinates. Note: if the window has not yet been realized, GetSize() and GetOrigin() return (0,0).
- `int = obj.GetOrigin ()` - Get the size and origin of the viewport in display coordinates. Note: if the window has not yet been realized, GetSize() and GetOrigin() return (0,0).
- `obj.GetTiledSize (int width, int height)` - Get the size and origin of the viewport in display coordinates. Note: if the window has not yet been realized, GetSize() and GetOrigin() return (0,0).
- `obj.GetTiledSizeAndOrigin (int width, int height, int lowerLeftX, int lowerLeftY)` - Get the size and origin of the viewport in display coordinates. Note: if the window has not yet been realized, GetSize() and GetOrigin() return (0,0).
- `vtkAssemblyPath = obj.PickProp (double selectionX, double selectionY)` - Return the Prop that has the highest z value at the given x, y position in the viewport. Basically, the top most prop that renders the pixel at selectionX, selectionY will be returned. If no Props are there NULL is returned. This method selects from the Viewports Prop list.
- `vtkAssemblyPath = obj.PickPropFrom (double selectionX, double selectionY, vtkPropCollection )` - Same as PickProp with two arguments, but selects from the given collection of Props instead of the Renderers props. Make sure the Props in the collection are in this renderer.

- `double = obj.GetPickX () const` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `double = obj.GetPickY () const` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `double = obj.GetPickWidth () const` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `double = obj.GetPickHeight () const` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `double = obj.GetPickX1 () const` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `double = obj.GetPickY1 () const` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `double = obj.GetPickX2 () const` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `double = obj.GetPickY2 () const` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `int = obj.GetIsPicking ()` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `vtkPropCollection = obj.GetPickResultProps ()` - Methods used to return the pick (x,y) in local display coordinates (i.e., it's that same as selectionX and selectionY).
- `double = obj.GetPickedZ ()` - Return the Z value for the last picked Prop.
- `obj.RemoveProp (vtkProp )` - @deprecated Replaced by vtkViewport::RemoveViewProp() as of VTK 5.0.
- `obj.AddProp (vtkProp )` - @deprecated Replaced by vtkViewport::AddViewProp() as of VTK 5.0.
- `vtkPropCollection = obj.GetProps ()` - @deprecated Replaced by vtkViewport::GetViewProps() as of VTK 5.0.
- `int = obj.HasProp (vtkProp )` - @deprecated Replaced by vtkViewport::HasViewProp() as of VTK 5.0.
- `obj.RemoveAllProps ()` - @deprecated Replaced by vtkViewport::RemoveAllViewProps() as of VTK 5.0.

31.229 vtkVoxel

31.229.1 Usage

vtkVoxel is a concrete implementation of vtkCell to represent a 3D orthogonal parallelepiped. Unlike vtkHexahedron, vtkVoxel has interior angles of 90 degrees, and sides are parallel to coordinate axes. This results in large increases in computational performance.

To create an instance of class vtkVoxel, simply invoke its constructor as follows

```
obj = vtkVoxel
```

31.229.2 Methods

The class vtkVoxel has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVoxel class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVoxel = obj.NewInstance ()`
- `vtkVoxel = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `obj.InterpolateFunctions (double pcoords[3], double weights[8])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[24])` - Return the ids of the vertices defining edge/face (‘edgeId‘/‘faceId’). Ids are related to the cell, not to the dataset.

31.230 vtkWedge

31.230.1 Usage

vtkWedge is a concrete implementation of vtkCell to represent a linear 3D wedge. A wedge consists of two triangular and three quadrilateral faces and is defined by the six points (0-5). vtkWedge uses the standard isoparametric shape functions for a linear wedge. The wedge is defined by the six points (0-5) where (0,1,2) is the base of the wedge which, using the right hand rule, forms a triangle whose normal points outward (away from the triangular face (3,4,5)).

To create an instance of class vtkWedge, simply invoke its constructor as follows

```
obj = vtkWedge
```

31.230.2 Methods

The class vtkWedge has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWedge class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWedge = obj.NewInstance ()`
- `vtkWedge = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetCellType ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetCellDimension ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfEdges ()` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetNumberOfFaces ()` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetEdge (int edgeId)` - See the vtkCell API for descriptions of these methods.
- `vtkCell = obj.GetFace (int faceId)` - See the vtkCell API for descriptions of these methods.
- `int = obj.CellBoundary (int subId, double pcoords[3], vtkIdList pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Contour (double value, vtkDataArray cellScalars, vtkIncrementalPointLocator locator, vtkCellAr` - See the vtkCell API for descriptions of these methods.
- `int = obj.Triangulate (int index, vtkIdList ptIds, vtkPoints pts)` - See the vtkCell API for descriptions of these methods.
- `obj.Derivatives (int subId, double pcoords[3], double values, int dim, double derivs)` - See the vtkCell API for descriptions of these methods.
- `int = obj.GetParametricCenter (double pcoords[3])` - Return the center of the wedge in parametric coordinates.
- `obj.InterpolateFunctions (double pcoords[3], double weights[6])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)
- `obj.InterpolateDerivs (double pcoords[3], double derivs[18])` - Compute the interpolation functions/derivatives (aka shape functions/derivatives)

Chapter 32

Visualization Toolkit Geo Vis Classes

32.1 vtkCompassRepresentation

32.1.1 Usage

This class is used to represent and render a compass.

To create an instance of class vtkCompassRepresentation, simply invoke its constructor as follows

```
obj = vtkCompassRepresentation
```

32.1.2 Methods

The class vtkCompassRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompassRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkCompassRepresentation = obj.NewInstance ()` - Standard methods for the class.
- `vtkCompassRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.
- `vtkCoordinate = obj.GetPoint1Coordinate ()` - Position the first end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point1Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.
- `vtkCoordinate = obj.GetPoint2Coordinate ()` - Position the second end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point2Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.
- `vtkProperty2D = obj.GetRingProperty ()` - Get the slider properties. The properties of the slider when selected and unselected can be manipulated.
- `vtkProperty2D = obj.GetSelectedProperty ()` - Get the selection property. This property is used to modify the appearance of selected objects (e.g., the slider).
- `vtkTextProperty = obj.GetLabelProperty ()` - Set/Get the properties for the label and title text.

- `obj.PlaceWidget (double bounds[6])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.BuildRepresentation ()` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.StartWidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.WidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.TiltWidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.DistanceWidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.Highlight (int )` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.GetActors (vtkPropCollection )`
- `obj.ReleaseGraphicsResources (vtkWindow )`
- `int = obj.RenderOverlay (vtkViewport )`
- `int = obj.RenderOpaqueGeometry (vtkViewport )`
- `obj.SetHeading (double value)`
- `double = obj.GetHeading ()`
- `obj.SetTilt (double value)`
- `double = obj.GetTilt ()`
- `obj.UpdateTilt (double time)`
- `obj.EndTilt ()`
- `obj.SetDistance (double value)`
- `double = obj.GetDistance ()`
- `obj.UpdateDistance (double time)`
- `obj.EndDistance ()`
- `obj.SetRenderer (vtkRenderer ren)`

32.2 vtkCompassWidget

32.2.1 Usage

The vtkCompassWidget is used to adjust a scalar value in an application. Note that the actual appearance of the widget depends on the specific representation for the widget.

To use this widget, set the widget representation. (the details may vary depending on the particulars of the representation).

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
If the slider bead is selected:
  LeftButtonPressEvent - select slider
  LeftButtonReleaseEvent - release slider
  MouseMoveEvent - move slider
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkCompassWidget's widget events:

```
vtkWidgetEvent::Select -- some part of the widget has been selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Move -- a request for slider motion has been invoked
```

In turn, when these widget events are processed, the vtkCompassWidget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkCompassWidget, simply invoke its constructor as follows

```
obj = vtkCompassWidget
```

32.2.2 Methods

The class vtkCompassWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompassWidget class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkCompassWidget = obj.NewInstance ()` - Standard macros.
- `vtkCompassWidget = obj.SafeDownCast (vtkObject o)` - Standard macros.
- `obj.SetRepresentation (vtkCompassRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `double = obj.GetHeading ()` - Get the value for this widget.
- `obj.SetHeading (double v)` - Get the value for this widget.

- `double = obj.GetTilt ()` - Get the value for this widget.
- `obj.SetTilt (double t)` - Get the value for this widget.
- `double = obj.GetDistance ()` - Get the value for this widget.
- `obj.SetDistance (double t)` - Get the value for this widget.

32.3 vtkGeoAdaptiveArcs

32.3.1 Usage

To create an instance of class vtkGeoAdaptiveArcs, simply invoke its constructor as follows

```
obj = vtkGeoAdaptiveArcs
```

32.3.2 Methods

The class vtkGeoAdaptiveArcs has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoAdaptiveArcs class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoAdaptiveArcs = obj.NewInstance ()`
- `vtkGeoAdaptiveArcs = obj.SafeDownCast (vtkObject o)`
- `obj.SetGlobeRadius (double )` - The base radius used to determine the earth's surface. Default is the earth's radius in meters. TODO: Change this to take in a vtkGeoTerrain to get altitude.
- `double = obj.GetGlobeRadius ()` - The base radius used to determine the earth's surface. Default is the earth's radius in meters. TODO: Change this to take in a vtkGeoTerrain to get altitude.
- `obj.SetMaximumPixelSeparation (double )` - The maximum number of pixels between points on the arcs. If two adjacent points are farther than the threshold, the line segment will be subdivided such that each point is separated by at most the threshold.
- `double = obj.GetMaximumPixelSeparation ()` - The maximum number of pixels between points on the arcs. If two adjacent points are farther than the threshold, the line segment will be subdivided such that each point is separated by at most the threshold.
- `obj.SetMinimumPixelSeparation (double )` - The minimum number of pixels between points on the arcs. Points closer than the threshold will be skipped until a point farther than the minimum threshold is reached.
- `double = obj.GetMinimumPixelSeparation ()` - The minimum number of pixels between points on the arcs. Points closer than the threshold will be skipped until a point farther than the minimum threshold is reached.
- `obj.SetRenderer (vtkRenderer ren)` - The renderer used to estimate the number of pixels between points.
- `vtkRenderer = obj.GetRenderer ()` - The renderer used to estimate the number of pixels between points.
- `long = obj.GetMTime ()` - Return the modified time of this object.

32.4 vtkGeoAlignedImageRepresentation

32.4.1 Usage

vtkGeoAlignedImageRepresentation represents a high resolution image over the globle. It has an associated vtkGeoSource which is responsible for fetching new data. This class keeps the fetched data in a quad-tree structure organized by latitude and longitude.

To create an instance of class vtkGeoAlignedImageRepresentation, simply invoke its constructor as follows

```
obj = vtkGeoAlignedImageRepresentation
```

32.4.2 Methods

The class vtkGeoAlignedImageRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoAlignedImageRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoAlignedImageRepresentation = obj.NewInstance ()`
- `vtkGeoAlignedImageRepresentation = obj.SafeDownCast (vtkObject o)`
- `vtkGeoImageNode = obj.GetBestImageForBounds (double bounds[4])` - Retrieve the most refined image patch that covers the specified latitude and longitude bounds (lat-min, lat-max, long-min, long-max).
- `vtkGeoSource = obj.GetSource ()` - The source for this representation. This must be set first before calling GetBestImageForBounds.
- `obj.SetSource (vtkGeoSource source)` - The source for this representation. This must be set first before calling GetBestImageForBounds.
- `obj.SaveDatabase (string path)` - Serialize the database to the specified directory. Each image is stored as a .vti file. The Origin and Spacing of the saved image contain (lat-min, long-min) and (lat-max, long-max), respectively. Files are named based on their level and id within that level.

32.5 vtkGeoAlignedImageSource

32.5.1 Usage

vtkGeoAlignedImageSource uses a high resolution image to generate tiles at multiple resolutions in a hierarchy. It should be used as a source in vtkGeoAlignedImageRepresentation.

To create an instance of class vtkGeoAlignedImageSource, simply invoke its constructor as follows

```
obj = vtkGeoAlignedImageSource
```

32.5.2 Methods

The class vtkGeoAlignedImageSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoAlignedImageSource class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkGeoAlignedImageSource = obj.NewInstance ()`
- `vtkGeoAlignedImageSource = obj.SafeDownCast (vtkObject o)`
- `bool = obj.FetchRoot (vtkGeoTreeNode node)` - Fetch the root image.
- `bool = obj.FetchChild (vtkGeoTreeNode parent, int index, vtkGeoTreeNode child)` - Fetch a child image.
- `vtkImageData = obj.GetImage ()` - The high-resolution image to be used to cover the globe.
- `obj.SetImage (vtkImageData image)` - The high-resolution image to be used to cover the globe.
- `obj.SetLatitudeRange (double , double )` - The range of the input hi-res image.
- `obj.SetLatitudeRange (double  a[2])` - The range of the input hi-res image.
- `double = obj. GetLatitudeRange ()` - The range of the input hi-res image.
- `obj.SetLongitudeRange (double , double )` - The range of the input hi-res image.
- `obj.SetLongitudeRange (double  a[2])` - The range of the input hi-res image.
- `double = obj. GetLongitudeRange ()` - The range of the input hi-res image.
- `obj.SetOverlap (double )` - The overlap of adjacent tiles.
- `double = obj.GetOverlapMinValue ()` - The overlap of adjacent tiles.
- `double = obj.GetOverlapMaxValue ()` - The overlap of adjacent tiles.
- `double = obj.GetOverlap ()` - The overlap of adjacent tiles.
- `obj.SetPowerOfTwoSize (bool )` - Whether to force image sizes to a power of two.
- `bool = obj.GetPowerOfTwoSize ()` - Whether to force image sizes to a power of two.
- `obj.PowerOfTwoSizeOn ()` - Whether to force image sizes to a power of two.
- `obj.PowerOfTwoSizeOff ()` - Whether to force image sizes to a power of two.

32.6 vtkGeoArcs

32.6.1 Usage

vtkGeoArcs produces arcs for each line in the input polydata. This is useful for viewing lines on a sphere (e.g. the earth). The arcs may "jump" above the sphere's surface using ExplodeFactor.

To create an instance of class vtkGeoArcs, simply invoke its constructor as follows

```
obj = vtkGeoArcs
```

32.6.2 Methods

The class vtkGeoArcs has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoArcs class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoArcs = obj.NewInstance ()`
- `vtkGeoArcs = obj.SafeDownCast (vtkObject o)`
- `obj.SetGlobeRadius (double )` - The base radius used to determine the earth's surface. Default is the earth's radius in meters.
- `double = obj.GetGlobeRadius ()` - The base radius used to determine the earth's surface. Default is the earth's radius in meters.
- `obj.SetExplodeFactor (double )` - Factor on which to "explode" the arcs away from the surface. A value of 0.0 keeps the values on the surface. Values larger than 0.0 push the arcs away from the surface by a distance proportional to the distance between the points. The default is 0.2.
- `double = obj.GetExplodeFactor ()` - Factor on which to "explode" the arcs away from the surface. A value of 0.0 keeps the values on the surface. Values larger than 0.0 push the arcs away from the surface by a distance proportional to the distance between the points. The default is 0.2.
- `obj.SetNumberOfSubdivisions (int )` - The number of subdivisions in the arc. The default is 20.
- `int = obj.GetNumberOfSubdivisions ()` - The number of subdivisions in the arc. The default is 20.

32.7 vtkGeoAssignCoordinates

32.7.1 Usage

Givem latitude and longitude arrays, take the values in those arrays and convert them to x,y,z world coordinates. Uses a spherical model of the earth to do the conversion. The position is in meters relative to the center of the earth.

If a transform is given, use the transform to convert latitude and longitude to the world coordinate.

To create an instance of class vtkGeoAssignCoordinates, simply invoke its constructor as follows

```
obj = vtkGeoAssignCoordinates
```

32.7.2 Methods

The class vtkGeoAssignCoordinates has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoAssignCoordinates class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoAssignCoordinates = obj.NewInstance ()`
- `vtkGeoAssignCoordinates = obj.SafeDownCast (vtkObject o)`
- `obj.SetLongitudeArrayName (string )` - Set the longitude coordinate array name.

- `string = obj.GetLongitudeArrayName ()` - Set the longitude coordinate array name.
- `obj.SetLatitudeArrayName (string )` - Set the latitude coordinate array name.
- `string = obj.GetLatitudeArrayName ()` - Set the latitude coordinate array name.
- `obj.SetGlobeRadius (double )` - The base radius to use in GLOBAL mode. Default is the earth's radius.
- `double = obj.GetGlobeRadius ()` - The base radius to use in GLOBAL mode. Default is the earth's radius.
- `obj.SetTransform (vtkAbstractTransform trans)` - The transform to use to convert coordinates of the form (lat, long, 0) to (x, y z). If this is NULL (the default), use GlobeRadius to perform a spherical embedding.
- `vtkAbstractTransform = obj.GetTransform ()` - The transform to use to convert coordinates of the form (lat, long, 0) to (x, y z). If this is NULL (the default), use GlobeRadius to perform a spherical embedding.
- `obj.SetCoordinatesInArrays (bool )` - If on, uses LatitudeArrayName and LongitudeArrayName to move values in data arrays into the points of the data set. Turn off if the lattitude and longitude are already in the points.
- `bool = obj.GetCoordinatesInArrays ()` - If on, uses LatitudeArrayName and LongitudeArrayName to move values in data arrays into the points of the data set. Turn off if the lattitude and longitude are already in the points.
- `obj.CoordinatesInArraysOn ()` - If on, uses LatitudeArrayName and LongitudeArrayName to move values in data arrays into the points of the data set. Turn off if the lattitude and longitude are already in the points.
- `obj.CoordinatesInArraysOff ()` - If on, uses LatitudeArrayName and LongitudeArrayName to move values in data arrays into the points of the data set. Turn off if the lattitude and longitude are already in the points.

32.8 vtkGeoCamera

32.8.1 Usage

I wanted to hide the normal vtkCamera API so I did not make this a subclass. The camera is a helper object. You can get a pointer to the camera, but it should be treated like a const.

To create an instance of class vtkGeoCamera, simply invoke its constructor as follows

```
obj = vtkGeoCamera
```

32.8.2 Methods

The class vtkGeoCamera has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoCamera class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoCamera = obj.NewInstance ()`

- `vtkGeoCamera = obj.SafeDownCast (vtkObject o)`
- `double = obj. GetPosition ()` - Get the world position without the origin shift.
- `obj.SetLongitude (double longitude)` - Longitude is in degrees: (-180-¿180) Relative to absolute coordinates. Rotate Longitude around z axis (earth axis),
- `double = obj.GetLongitude ()` - Longitude is in degrees: (-180-¿180) Relative to absolute coordinates. Rotate Longitude around z axis (earth axis),
- `obj.SetLatitude (double latitude)` - Latitude is in degrees: (-90-¿90) Relative to Longitude. Rotate Latitude around x axis by Latitude,
- `double = obj.GetLatitude ()` - Latitude is in degrees: (-90-¿90) Relative to Longitude. Rotate Latitude around x axis by Latitude,
- `obj.SetDistance (double Distance)` - Distance is in Meters Relative to Longitude and Latitude. above sea level ???? should we make this from center of earth ???? ???? what about equatorial bulge ????
- `double = obj.GetDistance ()` - Distance is in Meters Relative to Longitude and Latitude. above sea level ???? should we make this from center of earth ???? ???? what about equatorial bulge ????
- `obj.SetHeading (double heading)` - Heading is in degrees: (-180-¿180) Relative to Logitude and Latitude. 0 is north. 90 is east. ???? what is the standard ???? 180 is south. -90 is west. Rotate Heading around -y axis Center,
- `double = obj.GetHeading ()` - Heading is in degrees: (-180-¿180) Relative to Logitude and Latitude. 0 is north. 90 is east. ???? what is the standard ???? 180 is south. -90 is west. Rotate Heading around -y axis Center,
- `obj.SetTilt (double tilt)` - Tilt is also know as pitch. Tilt is in degrees: (0-¿90) Relative to Longitude, Latitude, and Heading. Rotate Tilt around x axis,
- `double = obj.GetTilt ()` - Tilt is also know as pitch. Tilt is in degrees: (0-¿90) Relative to Longitude, Latitude, and Heading. Rotate Tilt around x axis,
- `vtkCamera = obj.GetVTKCamera ()` - This vtk camera is updated to match this geo cameras state. It should be treated as a const and should not be modified.
- `obj.InitializeNodeAnalysis (int rendererSize[2])` - We precompute some values to speed up update of the terrain. Unfortunately, they have to be manually/explicitely updated when the camera or renderer size changes.
- `double = obj.GetNodeCoverage (vtkGeoTerrainNode node)` - This method estimates how much of the view is covered by the sphere. Returns a value from 0 to 1.
- `bool = obj.GetLockHeading ()`
- `obj.SetLockHeading (bool )`
- `obj.LockHeadingOn ()`
- `obj.LockHeadingOff ()`
- `obj.SetOriginLatitude (double oLat)` - This point is shifted to 0,0,0 to avoid openGL issues.
- `double = obj.GetOriginLatitude ()` - This point is shifted to 0,0,0 to avoid openGL issues.
- `obj.SetOriginLongitude (double oLat)` - This point is shifted to 0,0,0 to avoid openGL issues.
- `double = obj.GetOriginLongitude ()` - This point is shifted to 0,0,0 to avoid openGL issues.

- `double = obj. GetOrigin ()` - Get the rectilinear cooridinate location of the origin. This is used to shift the terrain points.
- `obj.SetOrigin (double ox, double oy, double oz)`

32.9 vtkGeoFileImageSource

32.9.1 Usage

vtkGeoFileImageSource is a vtkGeoSource that fetches .vti images from disk in a directory with a certain naming scheme. You may use vtkGeoAlignedImageRepresentation's SaveDatabase method to generate an database of image tiles in this format.

To create an instance of class vtkGeoFileImageSource, simply invoke its constructor as follows

```
obj = vtkGeoFileImageSource
```

32.9.2 Methods

The class vtkGeoFileImageSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoFileImageSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoFileImageSource = obj.NewInstance ()`
- `vtkGeoFileImageSource = obj.SafeDownCast (vtkObject o)`
- `vtkGeoFileImageSource = obj.()`
- `~vtkGeoFileImageSource = obj.()`
- `bool = obj.FetchRoot (vtkGeoTreeNode root)` - Fetches the root image representing the whole globe.
- `bool = obj.FetchChild (vtkGeoTreeNode node, int index, vtkGeoTreeNode child)` - Fetches the child image of a parent from disk.
- `obj.SetPath (string )` - The path the tiled image database.
- `string = obj.GetPath ()` - The path the tiled image database.

32.10 vtkGeoFileTerrainSource

32.10.1 Usage

vtkGeoFileTerrainSource reads geometry tiles as .vtp files from a directory that follow a certain naming convention containing the level of the patch and the position within that level. Use vtkGeoTerrain's SaveDatabase method to create a database of files in this format.

To create an instance of class vtkGeoFileTerrainSource, simply invoke its constructor as follows

```
obj = vtkGeoFileTerrainSource
```

32.10.2 Methods

The class vtkGeoFileTerrainSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoFileTerrainSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoFileTerrainSource = obj.NewInstance ()`
- `vtkGeoFileTerrainSource = obj.SafeDownCast (vtkObject o)`
- `vtkGeoFileTerrainSource = obj.()`
- `~vtkGeoFileTerrainSource = obj.()`
- `bool = obj.FetchRoot (vtkGeoTreeNode root)` - Retrieve the root geometry representing the entire globe.
- `bool = obj.FetchChild (vtkGeoTreeNode node, int index, vtkGeoTreeNode child)`
- `obj.SetPath (string )` - The path the tiled geometry database.
- `string = obj.GetPath ()` - The path the tiled geometry database.

32.11 vtkGeoGlobeSource

32.11.1 Usage

vtkGeoGlobeSource is a 3D vtkGeoSource suitable for use in vtkGeoTerrain. It uses the vtkGlobeSource filter to produce terrain patches.

To create an instance of class vtkGeoGlobeSource, simply invoke its constructor as follows

```
obj = vtkGeoGlobeSource
```

32.11.2 Methods

The class vtkGeoGlobeSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoGlobeSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoGlobeSource = obj.NewInstance ()`
- `vtkGeoGlobeSource = obj.SafeDownCast (vtkObject o)`
- `bool = obj.FetchRoot (vtkGeoTreeNode root)` - Fetches a low-resolution sphere for the entire globe.
- `bool = obj.FetchChild (vtkGeoTreeNode node, int index, vtkGeoTreeNode child)` - Fetches a refined geometry patch, a section of a sphere.

32.12 vtkGeoGraticule

32.12.1 Usage

This filter generates polydata to illustrate the distortions introduced by a map projection. The level parameter specifies the number of lines to be drawn. Poles are treated differently than other regions; hence the use of a Level parameter instead of a NumberOfLines parameter. The latitude and longitude are specified as half-open intervals with units of degrees. By default the latitude bounds are [-90,90[and the longitude bounds are [0,180[.

To create an instance of class vtkGeoGraticule, simply invoke its constructor as follows

```
obj = vtkGeoGraticule
```

32.12.2 Methods

The class vtkGeoGraticule has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoGraticule class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoGraticule = obj.NewInstance ()`
- `vtkGeoGraticule = obj.SafeDownCast (vtkObject o)`
- `obj.SetLatitudeBounds (double , double )` - The latitude bounds of the graticule.
- `obj.SetLatitudeBounds (double  a[2])` - The latitude bounds of the graticule.
- `double = obj. GetLatitudeBounds ()` - The latitude bounds of the graticule.
- `obj.SetLongitudeBounds (double , double )` - The longitude bounds of the graticule.
- `obj.SetLongitudeBounds (double  a[2])` - The longitude bounds of the graticule.
- `double = obj. GetLongitudeBounds ()` - The longitude bounds of the graticule.
- `obj.SetLatitudeLevel (int )` - The frequency level of latitude lines.
- `int = obj.GetLatitudeLevelMinValue ()` - The frequency level of latitude lines.
- `int = obj.GetLatitudeLevelMaxValue ()` - The frequency level of latitude lines.
- `int = obj.GetLatitudeLevel ()` - The frequency level of latitude lines.
- `obj.SetLongitudeLevel (int )` - The frequency level of longitude lines.
- `int = obj.GetLongitudeLevelMinValue ()` - The frequency level of longitude lines.
- `int = obj.GetLongitudeLevelMaxValue ()` - The frequency level of longitude lines.
- `int = obj.GetLongitudeLevel ()` - The frequency level of longitude lines.
- `obj.SetGeometryType (int )` - Set//get the type(s) of cells that will be output by the filter. By default, polylines are output. You may also request quadrilaterals. This is a bit vector of GeometryType enums.
- `int = obj.GetGeometryType ()` - Set//get the type(s) of cells that will be output by the filter. By default, polylines are output. You may also request quadrilaterals. This is a bit vector of GeometryType enums.

32.13 vtkGeoImageNode

32.13.1 Usage

vtkGeoImageNode contains an image tile in a multi-resolution image tree, along with metadata about that image's extents.

To create an instance of class vtkGeoImageNode, simply invoke its constructor as follows

```
obj = vtkGeoImageNode
```

32.13.2 Methods

The class vtkGeoImageNode has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoImageNode class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoImageNode = obj.NewInstance ()`
- `vtkGeoImageNode = obj.SafeDownCast (vtkObject o)`
- `vtkGeoImageNode = obj.GetChild (int idx)`
- `vtkGeoImageNode = obj.GetParent ()`
- `vtkImageData = obj.GetImage ()` - Get the image tile.
- `obj.SetImage (vtkImageData image)` - Get the image tile.
- `vtkTexture = obj.GetTexture ()` - Get the image tile.
- `obj.SetTexture (vtkTexture texture)` - Get the image tile.
- `obj.CropImageForTile (vtkImageData image, double imageLonLatExt, string prefix)` - This crops the image as small as possible while still covering the patch. The Longitude Latitude range may get bigger to reflect the actual size of the image. If prefix is specified, writes the tile to that location.
- `obj.LoadAnImage (string prefix)` - This loads the image from a tile database at the specified location.
- `obj.ShallowCopy (vtkGeoTreeNode src)` - Shallow and Deep copy.
- `obj.DeepCopy (vtkGeoTreeNode src)` - Shallow and Deep copy.
- `bool = obj.HasData ()`
- `obj.DeleteData ()` - Deletes the data associated with the node to make this an "empty" node. This is performed when the node has been unused for a certain amount of time.

32.14 vtkGeoInteractorStyle

32.14.1 Usage

vtkGeoInteractorStyle contains interaction capabilities for a geographic view including orbit, zoom, and tilt. It also includes a compass widget for changing view parameters.

To create an instance of class vtkGeoInteractorStyle, simply invoke its constructor as follows

```
obj = vtkGeoInteractorStyle
```

32.14.2 Methods

The class vtkGeoInteractorStyle has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoInteractorStyle class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoInteractorStyle = obj.NewInstance ()`
- `vtkGeoInteractorStyle = obj.SafeDownCast (vtkObject o)`
- `obj.OnEnter ()` - Event bindings
- `obj.OnLeave ()` - Event bindings
- `obj.OnMouseMove ()` - Event bindings
- `obj.OnLeftButtonUp ()` - Event bindings
- `obj.OnMiddleButtonUp ()` - Event bindings
- `obj.OnRightButtonUp ()` - Event bindings
- `obj.OnLeftButtonDown ()` - Event bindings
- `obj.OnMiddleButtonDown ()` - Event bindings
- `obj.OnRightButtonDown ()` - Event bindings
- `obj.OnChar ()` - Event bindings
- `obj.RubberBandZoom ()`
- `obj.Pan ()`
- `obj.Dolly ()`
- `obj.RedrawRectangle ()`
- `obj.StartState (int newstate)`
- `vtkGeoCamera = obj.GetGeoCamera ()`
- `obj.ResetCamera ()` - This can be used to set the camera to the standard view of the earth.
- `obj.WidgetInteraction (vtkObject caller)`
- `obj.SetInteractor (vtkRenderWindowInteractor interactor)` - Set/Get the Interactor wrapper being controlled by this object. (Satisfy superclass API.)
- `int = obj.GetRayIntersection (double origin[3], double direction[3], double intersection[3])`
- `obj.SetCurrentRenderer (vtkRenderer )` - Override to make the renderer use this camera subclass
- `bool = obj.GetLockHeading ()`
- `obj.SetLockHeading (bool )`
- `obj.LockHeadingOn ()`
- `obj.LockHeadingOff ()`
- `obj.ResetCameraClippingRange ()`

32.15 vtkGeoProjection

32.15.1 Usage

This class uses the PROJ.4 library to represent geographic coordinate projections.

To create an instance of class vtkGeoProjection, simply invoke its constructor as follows

```
obj = vtkGeoProjection
```

32.15.2 Methods

The class vtkGeoProjection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoProjection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoProjection = obj.NewInstance ()`
- `vtkGeoProjection = obj.SafeDownCast (vtkObject o)`
- `obj.SetName (string )` - Set/get the short name describing the projection you wish to use. This defaults to "rpoly" for no reason other than I like it. To get a list of valid values, use the GetNumberOfProjections() and GetProjectionName(int) static methods.
- `string = obj.GetName ()` - Set/get the short name describing the projection you wish to use. This defaults to "rpoly" for no reason other than I like it. To get a list of valid values, use the GetNumberOfProjections() and GetProjectionName(int) static methods.
- `int = obj.GetIndex ()` - Return the index of the current projection's type in the list of all projection types. On error, this will return -1. On success, it returns a number in [0,GetNumberOfProjections()[.
- `string = obj.GetDescription ()` - Get the description of a projection. This will return NULL if the projection name is invalid.
- `obj.SetCentralMeridian (double )` - Set/get the longitude which corresponds to the central meridian of the projection. This defaults to 0, the Greenwich Meridian.
- `double = obj.GetCentralMeridian ()` - Set/get the longitude which corresponds to the central meridian of the projection. This defaults to 0, the Greenwich Meridian.

32.16 vtkGeoProjectionSource

32.16.1 Usage

vtkGeoProjectionSource is a vtkGeoSource suitable for use in vtkTerrain2D. This source uses the libproj4 library to produce geometry patches at multiple resolutions. Each patch covers a specific region in projected space.

To create an instance of class vtkGeoProjectionSource, simply invoke its constructor as follows

```
obj = vtkGeoProjectionSource
```

32.16.2 Methods

The class vtkGeoProjectionSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoProjectionSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoProjectionSource = obj.NewInstance ()`
- `vtkGeoProjectionSource = obj.SafeDownCast (vtkObject o)`
- `vtkGeoProjectionSource = obj.()`
- `~vtkGeoProjectionSource = obj.()`
- `bool = obj.FetchRoot (vtkGeoTreeNode root)` - Blocking methods for sources with low latency.
- `bool = obj.FetchChild (vtkGeoTreeNode node, int index, vtkGeoTreeNode child)` - Blocking methods for sources with low latency.
- `int = obj.GetProjection ()` - The projection ID defining the projection. Initial value is 0.
- `obj.SetProjection (int projection)` - The projection ID defining the projection. Initial value is 0.
- `int = obj.GetMinCellsPerNode ()` - The minimum number of cells per node.
- `obj.SetMinCellsPerNode (int )` - The minimum number of cells per node.
- `vtkAbstractTransform = obj.GetTransform ()` - Return the projection transformation used by this 2D terrain.

32.17 vtkGeoRandomGraphSource

32.17.1 Usage

Generates a graph with a specified number of vertices, with the density of edges specified by either an exact number of edges or the probability of an edge. You may additionally specify whether to begin with a random tree (which enforces graph connectivity).

The filter also adds random vertex attributes called latitude and longitude. The latitude is distributed uniformly from -90 to 90, while longitude is distributed uniformly from -180 to 180.

To create an instance of class vtkGeoRandomGraphSource, simply invoke its constructor as follows

```
obj = vtkGeoRandomGraphSource
```

32.17.2 Methods

The class vtkGeoRandomGraphSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoRandomGraphSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoRandomGraphSource = obj.NewInstance ()`
- `vtkGeoRandomGraphSource = obj.SafeDownCast (vtkObject o)`

32.18 vtkGeoSampleArcs

32.18.1 Usage

vtkGeoSampleArcs refines lines in the input polygonal data so that the distance between adjacent points is no more than a threshold distance. Points are interpolated along the surface of the globe. This is useful in order to keep lines such as political boundaries from intersecting the globe and becoming invisible.

To create an instance of class vtkGeoSampleArcs, simply invoke its constructor as follows

```
obj = vtkGeoSampleArcs
```

32.18.2 Methods

The class vtkGeoSampleArcs has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoSampleArcs class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoSampleArcs = obj.NewInstance ()`
- `vtkGeoSampleArcs = obj.SafeDownCast (vtkObject o)`
- `obj.SetGlobeRadius (double )` - The base radius used to determine the earth's surface. Default is the earth's radius in meters. TODO: Change this to take in a vtkGeoTerrain to get altitude.
- `double = obj.GetGlobeRadius ()` - The base radius used to determine the earth's surface. Default is the earth's radius in meters. TODO: Change this to take in a vtkGeoTerrain to get altitude.
- `obj.SetMaximumDistanceMeters (double )` - The maximum distance, in meters, between adjacent points.
- `double = obj.GetMaximumDistanceMeters ()` - The maximum distance, in meters, between adjacent points.
- `obj.SetInputCoordinateSystem (int )` - The input coordinate system. RECTANGULAR is x,y,z meters relative the the earth center. SPHERICAL is longitude,latitude,altitude.
- `int = obj.GetInputCoordinateSystem ()` - The input coordinate system. RECTANGULAR is x,y,z meters relative the the earth center. SPHERICAL is longitude,latitude,altitude.
- `obj.SetInputCoordinateSystemToRectangular ()` - The input coordinate system. RECTANGULAR is x,y,z meters relative the the earth center. SPHERICAL is longitude,latitude,altitude.
- `obj.SetInputCoordinateSystemToSpherical ()` - The desired output coordinate system. RECTANGULAR is x,y,z meters relative the the earth center. SPHERICAL is longitude,latitude,altitude.
- `obj.SetOutputCoordinateSystem (int )` - The desired output coordinate system. RECTANGULAR is x,y,z meters relative the the earth center. SPHERICAL is longitude,latitude,altitude.
- `int = obj.GetOutputCoordinateSystem ()` - The desired output coordinate system. RECTANGULAR is x,y,z meters relative the the earth center. SPHERICAL is longitude,latitude,altitude.
- `obj.SetOutputCoordinateSystemToRectangular ()` - The desired output coordinate system. RECTANGULAR is x,y,z meters relative the the earth center. SPHERICAL is longitude,latitude,altitude.
- `obj.SetOutputCoordinateSystemToSpherical ()`

32.19 vtkGeoSource

32.19.1 Usage

vtkGeoSource is an abstract superclass for all multi-resolution data sources shown in a geographic view like vtkGeoView or vtkGeoView2D. vtkGeoSource subclasses need to implement the FetchRoot() method, which fills a vtkGeoTreeNode with the low-res data at the root, and FetchChild(), which produces a refinement of a parent node. Other geovis classes such as vtkGeoTerrain, vtkGeoTerrain2D, and vtkGeoAlignedImageSource use a vtkGeoSource subclass to build their geometry or image caches which are stored in trees. The source itself does not maintain the tree, but simply provides a mechanism for generating refined tree nodes.

Sources are multi-threaded. Each source may have one or more worker threads associated with it, which this superclass manages. It is essential that the FetchChild() method is thread-safe, since it may be called from multiple workers simultaneously.

To create an instance of class vtkGeoSource, simply invoke its constructor as follows

```
obj = vtkGeoSource
```

32.19.2 Methods

The class vtkGeoSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoSource = obj.NewInstance ()`
- `vtkGeoSource = obj.SafeDownCast (vtkObject o)`
- `vtkGeoSource = obj.()`
- `~vtkGeoSource = obj.()`
- `bool = obj.FetchRoot (vtkGeoTreeNode root)` - Blocking access methods to be implemented in subclasses.
- `bool = obj.FetchChild (vtkGeoTreeNode node, int index, vtkGeoTreeNode child)` - Blocking access methods to be implemented in subclasses.
- `obj.RequestChildren (vtkGeoTreeNode node)` - Non-blocking methods for to use from the main application. After calling RequestChildren() for a certain node, GetRequestedNodes() will after a certain period of time return a non-null pointer to a collection of four vtkGeoTreeNode objects, which are the four children of the requested node. The collection is reference counted, so you need to eventually call Delete() on the returned collection pointer (if it is non-null).
- `vtkCollection = obj.GetRequestedNodes (vtkGeoTreeNode node)` - Non-blocking methods for to use from the main application. After calling RequestChildren() for a certain node, GetRequestedNodes() will after a certain period of time return a non-null pointer to a collection of four vtkGeoTreeNode objects, which are the four children of the requested node. The collection is reference counted, so you need to eventually call Delete() on the returned collection pointer (if it is non-null).
- `obj.Initialize (int numThreads)` - Spawn worker threads.
- `obj.ShutDown ()` - Shut down the source. This terminates the thread and releases memory.
- `obj.WorkerThread ()`
- `vtkAbstractTransform = obj.GetTransform ()`

32.20 vtkGeoSphereTransform

32.20.1 Usage

the cartesian coordinate system is the following (if BaseAltitude is 0), - the origin is at the center of the earth - the x axis goes from the origin to (longtitude=-90,latitude=0), intersection of equator and the meridian passing just east of Galapagos Islands - the y axis goes from the origin to the intersection of Greenwitch meridian and equator (longitude=0,latitude=0) - the z axis goes from the origin to the Geographic North Pole (latitude=90) - therefore the frame is right-handed.

To create an instance of class vtkGeoSphereTransform, simply invoke its constructor as follows

```
obj = vtkGeoSphereTransform
```

32.20.2 Methods

The class vtkGeoSphereTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoSphereTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoSphereTransform = obj.NewInstance ()`
- `vtkGeoSphereTransform = obj.SafeDownCast (vtkObject o)`
- `obj.Inverse ()` - Invert the transformation.
- `obj.InternalTransformPoint (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.
- `obj.SetToRectangular (bool )` - If on, this transform converts (long,lat,alt) triples to (x,y,z) as an offset from the center of the earth. Alt, x, y, and z are all be in meters. If off, the tranform works in the reverse direction. Initial value is on.
- `bool = obj.GetToRectangular ()` - If on, this transform converts (long,lat,alt) triples to (x,y,z) as an offset from the center of the earth. Alt, x, y, and z are all be in meters. If off, the tranform works in the reverse direction. Initial value is on.
- `obj.ToRectangularOn ()` - If on, this transform converts (long,lat,alt) triples to (x,y,z) as an offset from the center of the earth. Alt, x, y, and z are all be in meters. If off, the tranform works in the reverse direction. Initial value is on.
- `obj.ToRectangularOff ()` - If on, this transform converts (long,lat,alt) triples to (x,y,z) as an offset from the center of the earth. Alt, x, y, and z are all be in meters. If off, the tranform works in the reverse direction. Initial value is on.
- `obj.SetBaseAltitude (double )` - The base altitude to transform coordinates to. This can be useful for transforming lines just above the earth's surface. Default is 0.
- `double = obj.GetBaseAltitude ()` - The base altitude to transform coordinates to. This can be useful for transforming lines just above the earth's surface. Default is 0.

32.21 vtkGeoTerrain

32.21.1 Usage

vtkGeoTerrain contains a multi-resolution tree of geometry representing the globe. It uses a vtkGeoSource subclass to generate the terrain, such as vtkGeoGlobeSource. This source must be set before using the terrain in a vtkGeoView. The terrain also contains an AddActors() method which will update the set of actors representing the globe given the current camera position.

To create an instance of class vtkGeoTerrain, simply invoke its constructor as follows

```
obj = vtkGeoTerrain
```

32.21.2 Methods

The class vtkGeoTerrain has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoTerrain class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoTerrain = obj.NewInstance ()`
- `vtkGeoTerrain = obj.SafeDownCast (vtkObject o)`
- `vtkGeoSource = obj.GetSource ()` - The source used to obtain geometry patches.
- `obj.SetSource (vtkGeoSource source)` - The source used to obtain geometry patches.
- `obj.SaveDatabase (string path, int depth)` - Save the set of patches up to a given maximum depth.
- `obj.AddActors (vtkRenderer ren, vtkAssembly assembly, vtkCollection imageReps)` - Update the actors in an assembly used to render the globe. ren is the current renderer, and imageReps holds the collection of vtkGeoAlignedImageRepresentations that will be blended together to form the image on the globe.
- `obj.SetOrigin (double , double , double )` - The world-coordinate origin offset used to eliminate precision errors when zoomed in to a particular region of the globe.
- `obj.SetOrigin (double a[3])` - The world-coordinate origin offset used to eliminate precision errors when zoomed in to a particular region of the globe.
- `double = obj. GetOrigin ()` - The world-coordinate origin offset used to eliminate precision errors when zoomed in to a particular region of the globe.
- `obj.SetMaxLevel (int )` - The maximum level of the terrain tree.
- `int = obj.GetMaxLevelMinValue ()` - The maximum level of the terrain tree.
- `int = obj.GetMaxLevelMaxValue ()` - The maximum level of the terrain tree.
- `int = obj.GetMaxLevel ()` - The maximum level of the terrain tree.

32.22 vtkGeoTerrain2D

32.22.1 Usage

vtkGeoTerrain2D contains a multi-resolution tree of geometry representing the globe. It uses a vtkGeoSource subclass to generate the terrain, such as vtkGeoProjectionSource. This source must be set before using the terrain in a vtkGeoView2D. The terrain also contains an AddActors() method which updates the set of actors representing the globe given the current camera position.

To create an instance of class vtkGeoTerrain2D, simply invoke its constructor as follows

```
obj = vtkGeoTerrain2D
```

32.22.2 Methods

The class vtkGeoTerrain2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoTerrain2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoTerrain2D = obj.NewInstance ()`
- `vtkGeoTerrain2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetTextureTolerance (double )` - The maximum size of a single texel in pixels. Images will be refined if a texel becomes larger than the tolerance.
- `double = obj.GetTextureTolerance ()` - The maximum size of a single texel in pixels. Images will be refined if a texel becomes larger than the tolerance.
- `obj.SetLocationTolerance (double )` - The maximum allowed deviation of geometry in pixels. Geometry will be refined if the deviation is larger than the tolerance.
- `double = obj.GetLocationTolerance ()` - The maximum allowed deviation of geometry in pixels. Geometry will be refined if the deviation is larger than the tolerance.
- `vtkAbstractTransform = obj.GetTransform ()` - Return the projection transformation used by this 2D terrain.

32.23 vtkGeoTerrainNode

32.23.1 Usage

To create an instance of class vtkGeoTerrainNode, simply invoke its constructor as follows

```
obj = vtkGeoTerrainNode
```

32.23.2 Methods

The class vtkGeoTerrainNode has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoTerrainNode class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkGeoTerrainNode = obj.NewInstance ()`
- `vtkGeoTerrainNode = obj.SafeDownCast (vtkObject o)`
- `vtkGeoTerrainNode = obj.GetChild (int idx)`
- `vtkGeoTerrainNode = obj.GetParent ()`
- `double = obj.GetAltitude (double longitude, double latitude)`
- `vtkPolyData = obj.GetModel ()` - Get the terrrain model. The user has to copy the terrain into this object.
- `obj.SetModel (vtkPolyData model)` - Get the terrrain model. The user has to copy the terrain into this object.
- `obj.UpdateBoundingSphere ()` - Bounding sphere is precomputed for faster updates of terrain.
- `double = obj.GetBoundingSphereRadius ()` - Bounding sphere is precomputed for faster updates of terrain.
- `double = obj. GetBoundingSphereCenter ()` - Bounding sphere is precomputed for faster updates of terrain.
- `double = obj. GetCornerNormal00 ()`
- `double = obj. GetCornerNormal01 ()`
- `double = obj. GetCornerNormal10 ()`
- `double = obj. GetCornerNormal11 ()`
- `double = obj. GetProjectionBounds ()` - For 2D projections, store the bounds of the node in projected space to quickly determine if a node is offscreen.
- `obj.SetProjectionBounds (double , double , double , double )` - For 2D projections, store the bounds of the node in projected space to quickly determine if a node is offscreen.
- `obj.SetProjectionBounds (double a[4])` - For 2D projections, store the bounds of the node in projected space to quickly determine if a node is offscreen.
- `int = obj.GetGraticuleLevel ()` - For 2D projections, store the granularity of the graticule in this node.
- `obj.SetGraticuleLevel (int )` - For 2D projections, store the granularity of the graticule in this node.
- `double = obj.GetError ()` - For 2D projections, store the maximum deviation of line segment centers from the actual projection value.
- `obj.SetError (double )` - For 2D projections, store the maximum deviation of line segment centers from the actual projection value.
- `float = obj.GetCoverage ()` - For 2D projections, store the maximum deviation of line segment centers from the actual projection value.
- `obj.SetCoverage (float )` - For 2D projections, store the maximum deviation of line segment centers from the actual projection value.
- `obj.ShallowCopy (vtkGeoTreeNode src)` - Shallow and Deep copy.

- `obj.DeepCopy (vtkGeoTreeNode src)` - Shallow and Deep copy.
- `bool = obj.HasData ()` - Returns whether this node has valid data associated with it, or if it is an "empty" node.
- `obj.DeleteData ()` - Deletes the data associated with the node to make this an "empty" node. This is performed when the node has been unused for a certain amount of time.

32.24 vtkGeoTransform

32.24.1 Usage

This class takes two geographic projections and transforms point coordinates between them.

To create an instance of class vtkGeoTransform, simply invoke its constructor as follows

```
obj = vtkGeoTransform
```

32.24.2 Methods

The class vtkGeoTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoTransform = obj.NewInstance ()`
- `vtkGeoTransform = obj.SafeDownCast (vtkObject o)`
- `obj.SetSourceProjection (vtkGeoProjection source)` - The source geographic projection.
- `vtkGeoProjection = obj.GetSourceProjection ()` - The source geographic projection.
- `obj.SetDestinationProjection (vtkGeoProjection dest)` - The target geographic projection.
- `vtkGeoProjection = obj.GetDestinationProjection ()` - The target geographic projection.
- `obj.TransformPoints (vtkPoints src, vtkPoints dst)` - Transform many points at once.
- `obj.Inverse ()` - Invert the transformation.
- `obj.InternalTransformPoint (float in[3], float out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `obj.InternalTransformPoint (double in[3], double out[3])` - This will calculate the transformation without calling Update. Meant for use only within other VTK classes.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.

32.25 vtkGeoTreeNode

32.25.1 Usage

A self-referential data structure for storing geometry or imagery for the geospatial views. The data is organized in a quadtree. Each node contains a pointer to its parent and owns references to its four child nodes. The ID of each node is unique in its level, and encodes the path from the root node in its bits.

To create an instance of class vtkGeoTreeNode, simply invoke its constructor as follows

```
obj = vtkGeoTreeNode
```

32.25.2 Methods

The class vtkGeoTreeNode has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoTreeNode class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoTreeNode = obj.NewInstance ()`
- `vtkGeoTreeNode = obj.SafeDownCast (vtkObject o)`
- `obj.SetId (long )` - The id uniquely specified this node. For this implementation I am going to store the branch path in the bits.
- `long = obj.GetId ()` - The id uniquely specified this node. For this implementation I am going to store the branch path in the bits.
- `obj.SetLevel (int )`
- `int = obj.GetLevel ()`
- `obj.SetLongitudeRange (double , double )` - Longitude and latitude range of the terrain model.
- `obj.SetLongitudeRange (double  a[2])` - Longitude and latitude range of the terrain model.
- `double = obj. GetLongitudeRange ()` - Longitude and latitude range of the terrain model.
- `obj.SetLatitudeRange (double , double )` - Longitude and latitude range of the terrain model.
- `obj.SetLatitudeRange (double  a[2])` - Longitude and latitude range of the terrain model.
- `double = obj. GetLatitudeRange ()` - Longitude and latitude range of the terrain model.
- `obj.SetChild (vtkGeoTreeNode node, int idx)` - Get a child of this node. If one is set, then they all should set. No not mix subclasses.
- `obj.SetParent (vtkGeoTreeNode node)` - Manage links to older and newer tree nodes. These are used to periodically delete unused patches.
- `obj.SetOlder (vtkGeoTreeNode node)` - Manage links to older and newer tree nodes. These are used to periodically delete unused patches.
- `vtkGeoTreeNode = obj.GetOlder ()` - Manage links to older and newer tree nodes. These are used to periodically delete unused patches.
- `obj.SetNewer (vtkGeoTreeNode node)` - Manage links to older and newer tree nodes. These are used to periodically delete unused patches.
- `vtkGeoTreeNode = obj.GetNewer ()` - Returns whether this node has valid data associated with it, or if it is an "empty" node.
- `bool = obj.HasData ()` - Deletes the data associated with the node to make this an "empty" node. This is performed when the node has been unused for a certain amount of time.
- `obj.DeleteData ()`
- `int = obj.GetWhichChildAreYou ()`

- `bool = obj.IsDescendantOf (vtkGeoTreeNode elder)` - This method returns true if this node descends from the elder node. The descision is made from the node ids, so the nodes do not have to be in the same tree!
- `int = obj.CreateChildren ()`
- `vtkGeoTreeNode = obj.GetChildTreeNode (int idx)` - Get the parent as a vtkGeoTreeNode. Subclasses also implement GetParent() which returns the parent as the appropriate subclass type.
- `vtkGeoTreeNode = obj.GetParentTreeNode ()` - Shallow and Deep copy. Deep copy performs a shallow copy of the Child nodes.
- `obj.ShallowCopy (vtkGeoTreeNode src)` - Shallow and Deep copy. Deep copy performs a shallow copy of the Child nodes.
- `obj.DeepCopy (vtkGeoTreeNode src)` - Shallow and Deep copy. Deep copy performs a shallow copy of the Child nodes.

32.26 vtkGeoTreeNodeCache

32.26.1 Usage

vtkGeoTreeNodeCache keeps track of a linked list of vtkGeoTreeNodes, and has operations to move nodes to the front of the list and to delete data from the least used nodes. This is used to recover memory from nodes that store data that hasn't been used in a while.

To create an instance of class vtkGeoTreeNodeCache, simply invoke its constructor as follows

```
obj = vtkGeoTreeNodeCache
```

32.26.2 Methods

The class vtkGeoTreeNodeCache has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoTreeNodeCache class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoTreeNodeCache = obj.NewInstance ()`
- `vtkGeoTreeNodeCache = obj.SafeDownCast (vtkObject o)`
- `obj.SetCacheMaximumLimit (int )` - The size of the cache of geospatial nodes. When the size reaches this limit, the list of non-empty nodes will be shortened to CacheMinimumLimit.
- `int = obj.GetCacheMaximumLimit ()` - The size of the cache of geospatial nodes. When the size reaches this limit, the list of non-empty nodes will be shortened to CacheMinimumLimit.
- `obj.SetCacheMinimumLimit (int )` - The cache is reduced to this size when the maximum limit is reached.
- `int = obj.GetCacheMinimumLimit ()` - The cache is reduced to this size when the maximum limit is reached.
- `obj.SendToFront (vtkGeoTreeNode node)` - Send a node to the front of the list. Perform this whenever a node is accessed, so that the most recently accessed nodes' data are not deleted.
- `obj.RemoveNode (vtkGeoTreeNode node)` - Remove the node from the list.
- `int = obj.GetSize ()` - The current size of the list.

32.27 vtkGeoView

32.27.1 Usage

vtkGeoView is a 3D globe view. The globe may contain a multi-resolution geometry source (vtkGeoTerrain), multiple multi-resolution image sources (vtkGeoAlignedImageRepresentation), as well as other representations such as vtkRenderedGraphRepresentation. At a minimum, the view must have a terrain and one image representation. The view uses vtkGeoInteractorStyle to orbit, zoom, and tilt the view, and contains a vtkCompassWidget for manipulating the camera.

Each terrain or image representation contains a vtkGeoSource subclass which generates geometry or imagery at multiple resolutions. As the camera position changes, the terrain and/or image representations may ask its vtkGeoSource to refine the geometry. This refinement is performed on a separate thread, and the data is added to the view when it becomes available.

To create an instance of class vtkGeoView, simply invoke its constructor as follows

```
obj = vtkGeoView
```

32.27.2 Methods

The class vtkGeoView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoView = obj.NewInstance ()`
- `vtkGeoView = obj.SafeDownCast (vtkObject o)`
- `vtkGeoAlignedImageRepresentation = obj.AddDefaultImageRepresentation (vtkImageData image)` - Adds an image representation with a simple terrain model using the image in the specified file as the globe terrain.
- `obj.PrepareForRendering ()`
- `obj.BuildLowResEarth (double origin[3])` - Rebuild low-res earth source; call after (re)setting origin.
- `obj.SetLockHeading (bool lock)` - Whether the view locks the heading when panning. Default is off.
- `bool = obj.GetLockHeading ()` - Whether the view locks the heading when panning. Default is off.
- `obj.LockHeadingOn ()` - Whether the view locks the heading when panning. Default is off.
- `obj.LockHeadingOff ()` - Whether the view locks the heading when panning. Default is off.
- `vtkGeoInteractorStyle = obj.GetGeoInteractorStyle ()` - Convenience method for obtaining the internal interactor style.
- `obj.SetGeoInteractorStyle (vtkGeoInteractorStyle style)` - Method to change the interactor style.
- `obj.SetTerrain (vtkGeoTerrain terrain)` - The terrain (geometry) model for this earth view.
- `vtkGeoTerrain = obj.GetTerrain ()` - The terrain (geometry) model for this earth view.
- `obj.Render ()` - Update and render the view.

32.28 vtkGeoView2D

32.28.1 Usage

vtkGeoView is a 2D globe view. The globe may contain a multi-resolution geometry source (vtkGeoTerrain2D), multiple multi-resolution image sources (vtkGeoAlignedImageRepresentation), as well as other representations such as vtkGeoGraphRepresentation2D. At a minimum, the view must have a terrain and one image representation. By default, you may select in the view with the left mouse button, pan with the middle button, and zoom with the right mouse button or scroll wheel.

Each terrain or image representation contains a vtkGeoSource subclass which generates geometry or imagery at multiple resolutions. As the camera position changes, the terrain and/or image representations may ask its vtkGeoSource to refine the geometry. This refinement is performed on a separate thread, and the data is added to the view when it becomes available.

To create an instance of class vtkGeoView2D, simply invoke its constructor as follows

```
obj = vtkGeoView2D
```

32.28.2 Methods

The class vtkGeoView2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoView2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoView2D = obj.NewInstance ()`
- `vtkGeoView2D = obj.SafeDownCast (vtkObject o)`
- `vtkGeoView2D = obj.()`
- `~vtkGeoView2D = obj.()`
- `vtkGeoTerrain2D = obj.GetSurface ()`
- `obj.SetSurface (vtkGeoTerrain2D surf)`
- `vtkAbstractTransform = obj.GetTransform ()` - Returns the transform associated with the surface.
- `obj.ApplyViewTheme (vtkViewTheme theme)` - Apply the view theme to this view.
- `obj.Render ()` - Update and render the view.

32.29 vtkGlobeSource

32.29.1 Usage

vtkGlobeSource will generate any "rectangular" patch of the globe given its Longitude-Latitude extent. It adds two point scalar arrays Longitude and Latitude to the output. These arrays can be transformed to generate texture coordinates for any texture map. This source is imperfect near the poles as implmented. It should really reduce the longitude resolution as the triangles become slivers.

To create an instance of class vtkGlobeSource, simply invoke its constructor as follows

```
obj = vtkGlobeSource
```

32.29.2 Methods

The class vtkGlobeSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGlobeSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGlobeSource = obj.NewInstance ()`
- `vtkGlobeSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetOrigin (double , double , double )`
- `obj.SetOrigin (double  a[3])`
- `obj.SetStartLongitude (double )` - Longitude Latitude clamps.
- `double = obj.GetStartLongitudeMinValue ()` - Longitude Latitude clamps.
- `double = obj.GetStartLongitudeMaxValue ()` - Longitude Latitude clamps.
- `obj.SetEndLongitude (double )` - Longitude Latitude clamps.
- `double = obj.GetEndLongitudeMinValue ()` - Longitude Latitude clamps.
- `double = obj.GetEndLongitudeMaxValue ()` - Longitude Latitude clamps.
- `obj.SetStartLatitude (double )` - Longitude Latitude clamps.
- `double = obj.GetStartLatitudeMinValue ()` - Longitude Latitude clamps.
- `double = obj.GetStartLatitudeMaxValue ()` - Longitude Latitude clamps.
- `obj.SetEndLatitude (double )` - Longitude Latitude clamps.
- `double = obj.GetEndLatitudeMinValue ()` - Longitude Latitude clamps.
- `double = obj.GetEndLatitudeMaxValue ()` - Longitude Latitude clamps.
- `obj.SetLongitudeResolution (int )` - Set the number of points in the longitude direction (ranging from StartLongitude to EndLongitude).
- `int = obj.GetLongitudeResolutionMinValue ()` - Set the number of points in the longitude direction (ranging from StartLongitude to EndLongitude).
- `int = obj.GetLongitudeResolutionMaxValue ()` - Set the number of points in the longitude direction (ranging from StartLongitude to EndLongitude).
- `int = obj.GetLongitudeResolution ()` - Set the number of points in the longitude direction (ranging from StartLongitude to EndLongitude).
- `obj.SetLatitudeResolution (int )` - Set the number of points in the latitude direction (ranging from StartLatitude to EndLatitude).
- `int = obj.GetLatitudeResolutionMinValue ()` - Set the number of points in the latitude direction (ranging from StartLatitude to EndLatitude).
- `int = obj.GetLatitudeResolutionMaxValue ()` - Set the number of points in the latitude direction (ranging from StartLatitude to EndLatitude).

- `int = obj.GetLatitudeResolution ()` - Set the number of points in the latitude direction (ranging from StartLatitude to EndLatitude).
- `obj.SetRadius (double )` - Set radius of sphere. Default is 6356750.0
- `double = obj.GetRadiusMinValue ()` - Set radius of sphere. Default is 6356750.0
- `double = obj.GetRadiusMaxValue ()` - Set radius of sphere. Default is 6356750.0
- `double = obj.GetRadius ()` - Set radius of sphere. Default is 6356750.0
- `obj.SetCurtainHeight (double )`
- `double = obj.GetCurtainHeightMinValue ()`
- `double = obj.GetCurtainHeightMaxValue ()`
- `double = obj.GetCurtainHeight ()`
- `obj.SetQuadrilateralTessellation (int )` - Cause the sphere to be tessellated with edges along the latitude and longitude lines. If off, triangles are generated at non-polar regions, which results in edges that are not parallel to latitude and longitude lines. If on, quadrilaterals are generated everywhere except at the poles. This can be useful for generating a wireframe sphere with natural latitude and longitude lines.
- `int = obj.GetQuadrilateralTessellation ()` - Cause the sphere to be tessellated with edges along the latitude and longitude lines. If off, triangles are generated at non-polar regions, which results in edges that are not parallel to latitude and longitude lines. If on, quadrilaterals are generated everywhere except at the poles. This can be useful for generating a wireframe sphere with natural latitude and longitude lines.
- `obj.QuadrilateralTessellationOn ()` - Cause the sphere to be tessellated with edges along the latitude and longitude lines. If off, triangles are generated at non-polar regions, which results in edges that are not parallel to latitude and longitude lines. If on, quadrilaterals are generated everywhere except at the poles. This can be useful for generating a wireframe sphere with natural latitude and longitude lines.
- `obj.QuadrilateralTessellationOff ()` - Cause the sphere to be tessellated with edges along the latitude and longitude lines. If off, triangles are generated at non-polar regions, which results in edges that are not parallel to latitude and longitude lines. If on, quadrilaterals are generated everywhere except at the poles. This can be useful for generating a wireframe sphere with natural latitude and longitude lines.

Chapter 33

Visualization Toolkit Graphics Classes

33.1 vtkAnnotationLink

33.1.1 Usage

vtkAnnotationLink is a simple source filter which outputs the vtkAnnotationLayers object stored internally. Multiple objects may share the same annotation link filter and connect it to an internal pipeline so that if one object changes the annotation set, it will be pulled into all the other objects when their pipelines update.

The shared vtkAnnotationLayers object (a collection of annotations) is shallow copied to output port 0.

vtkAnnotationLink can also store a set of domain maps. A domain map is simply a table associating values between domains. The domain of each column is defined by the array name of the column. The domain maps are sent to a multi-block dataset in output port 1.

Output ports 0 and 1 can be set as input ports 0 and 1 to vtkConvertSelectionDomain, which can use the domain maps to convert the domains of selections in the vtkAnnotationLayers to match a particular data object (set as port 2 on vtkConvertSelectionDomain).

The shared vtkAnnotationLayers object also stores a "current selection" normally interpreted as the interactive selection of an application. As a convenience, this selection is sent to output port 2 so that it can be connected to pipelines requiring a vtkSelection.

To create an instance of class vtkAnnotationLink, simply invoke its constructor as follows

```
obj = vtkAnnotationLink
```

33.1.2 Methods

The class vtkAnnotationLink has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAnnotationLink class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAnnotationLink = obj.NewInstance ()`
- `vtkAnnotationLink = obj.SafeDownCast (vtkObject o)`
- `vtkAnnotationLayers = obj.GetAnnotationLayers ()` - The annotations to be shared.
- `obj.SetAnnotationLayers (vtkAnnotationLayers layers)` - The annotations to be shared.
- `obj.SetCurrentSelection (vtkSelection sel)` - Set or get the current selection in the annotation layers.

- `vtkSelection = obj.GetCurrentSelection ()` - Set or get the current selection in the annotation layers.
- `obj.AddDomainMap (vtkTable map)` - The domain mappings.
- `obj.RemoveDomainMap (vtkTable map)` - The domain mappings.
- `obj.RemoveAllDomainMaps ()` - The domain mappings.
- `int = obj.GetNumberOfDomainMaps ()` - The domain mappings.
- `vtkTable = obj.GetDomainMap (int i)` - The domain mappings.
- `long = obj.GetMTime ()` - Get the mtime of this object.

33.2 vtkAppendCompositeDataLeaves

33.2.1 Usage

vtkAppendCompositeDataLeaves is a filter that takes input composite datasets with the same structure: (1) the same number of entries and – if any children are composites – the same constraint holds from them; and (2) the same type of dataset at each position. It then creates an output dataset with the same structure whose leaves contain all the cells from the datasets at the corresponding leaves of the input datasets.

Currently, only input polydata and unstructured grids are handled; other types of leaf datasets will be ignored and their positions in the output dataset will be NULL pointers. Point attributes (i.e., scalars, vectors, normals, field data, etc.) are extracted and appended only if all datasets have the point attributes available. (For example, if one dataset has scalars but another does not, scalars will not be appended.)

To create an instance of class vtkAppendCompositeDataLeaves, simply invoke its constructor as follows

```
obj = vtkAppendCompositeDataLeaves
```

33.2.2 Methods

The class vtkAppendCompositeDataLeaves has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAppendCompositeDataLeaves class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAppendCompositeDataLeaves = obj.NewInstance ()`
- `vtkAppendCompositeDataLeaves = obj.SafeDownCast (vtkObject o)`
- `obj.RemoveInput (vtkDataSet in)` - Remove a dataset from the list of data to append.
- `obj.SetAppendFieldData (int )` - Set/get whether the field data of each dataset in the composite dataset is copied to the output. If AppendFieldData is non-zero, then field data arrays from all the inputs are added to the output. If there are duplicates, the array on the first input encountered is taken.
- `int = obj.GetAppendFieldData ()` - Set/get whether the field data of each dataset in the composite dataset is copied to the output. If AppendFieldData is non-zero, then field data arrays from all the inputs are added to the output. If there are duplicates, the array on the first input encountered is taken.

- `obj.AppendFieldDataOn ()` - Set/get whether the field data of each dataset in the composite dataset is copied to the output. If AppendFieldData is non-zero, then field data arrays from all the inputs are added to the output. If there are duplicates, the array on the first input encountered is taken.

- `obj.AppendFieldDataOff ()` - Set/get whether the field data of each dataset in the composite dataset is copied to the output. If AppendFieldData is non-zero, then field data arrays from all the inputs are added to the output. If there are duplicates, the array on the first input encountered is taken.

33.3 vtkAppendFilter

33.3.1 Usage

vtkAppendFilter is a filter that appends one of more datasets into a single unstructured grid. All geometry is extracted and appended, but point attributes (i.e., scalars, vectors, normals, field data, etc.) are extracted and appended only if all datasets have the point attributes available. (For example, if one dataset has scalars but another does not, scalars will not be appended.)

To create an instance of class vtkAppendFilter, simply invoke its constructor as follows

```
obj = vtkAppendFilter
```

33.3.2 Methods

The class vtkAppendFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAppendFilter class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkAppendFilter = obj.NewInstance ()`

- `vtkAppendFilter = obj.SafeDownCast (vtkObject o)`

- `obj.RemoveInput (vtkDataSet in)` - Remove a dataset from the list of data to append.

- `vtkDataSetCollection = obj.GetInputList ()` - Returns a copy of the input array. Modifications to this list will not be reflected in the actual inputs.

33.4 vtkAppendPolyData

33.4.1 Usage

vtkAppendPolyData is a filter that appends one of more polygonal datasets into a single polygonal dataset. All geometry is extracted and appended, but point and cell attributes (i.e., scalars, vectors, normals) are extracted and appended only if all datasets have the point and/or cell attributes available. (For example, if one dataset has point scalars but another does not, point scalars will not be appended.)

To create an instance of class vtkAppendPolyData, simply invoke its constructor as follows

```
obj = vtkAppendPolyData
```

33.4.2 Methods

The class vtkAppendPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAppendPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAppendPolyData = obj.NewInstance ()`
- `vtkAppendPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetUserManagedInputs (int )` - UserManagedInputs allows the user to set inputs by number instead of using the AddInput/RemoveInput functions. Calls to SetNumberOfInputs/SetInputByNumber should not be mixed with calls to AddInput/RemoveInput. By default, UserManagedInputs is false.
- `int = obj.GetUserManagedInputs ()` - UserManagedInputs allows the user to set inputs by number instead of using the AddInput/RemoveInput functions. Calls to SetNumberOfInputs/SetInputByNumber should not be mixed with calls to AddInput/RemoveInput. By default, UserManagedInputs is false.
- `obj.UserManagedInputsOn ()` - UserManagedInputs allows the user to set inputs by number instead of using the AddInput/RemoveInput functions. Calls to SetNumberOfInputs/SetInputByNumber should not be mixed with calls to AddInput/RemoveInput. By default, UserManagedInputs is false.
- `obj.UserManagedInputsOff ()` - UserManagedInputs allows the user to set inputs by number instead of using the AddInput/RemoveInput functions. Calls to SetNumberOfInputs/SetInputByNumber should not be mixed with calls to AddInput/RemoveInput. By default, UserManagedInputs is false.
- `obj.AddInput (vtkPolyData )` - Add a dataset to the list of data to append. Should not be used when UserManagedInputs is true, use SetInputByNumber instead.
- `obj.RemoveInput (vtkPolyData )` - Remove a dataset from the list of data to append. Should not be used when UserManagedInputs is true, use SetInputByNumber (NULL) instead.
- `obj.SetNumberOfInputs (int num)` - Directly set(allocate) number of inputs, should only be used when UserManagedInputs is true.
- `obj.SetInputByNumber (int num, vtkPolyData input)`
- `obj.SetParallelStreaming (int )` - ParallelStreaming is for a particular application. It causes this filter to ask for a different piece from each of its inputs. If all the inputs are the same, then the output of this append filter is the whole dataset pieced back together. Duplicate points are create along the seams. The purpose of this feature is to get data parallelism at a course scale. Each of the inputs can be generated in a different process at the same time.
- `int = obj.GetParallelStreaming ()` - ParallelStreaming is for a particular application. It causes this filter to ask for a different piece from each of its inputs. If all the inputs are the same, then the output of this append filter is the whole dataset pieced back together. Duplicate points are create along the seams. The purpose of this feature is to get data parallelism at a course scale. Each of the inputs can be generated in a different process at the same time.
- `obj.ParallelStreamingOn ()` - ParallelStreaming is for a particular application. It causes this filter to ask for a different piece from each of its inputs. If all the inputs are the same, then the output of this append filter is the whole dataset pieced back together. Duplicate points are create along the seams. The purpose of this feature is to get data parallelism at a course scale. Each of the inputs can be generated in a different process at the same time.

- `obj.ParallelStreamingOff ()` - ParallelStreaming is for a particular application. It causes this filter to ask for a different piece from each of its inputs. If all the inputs are the same, then the output of this append filter is the whole dataset pieced back together. Duplicate points are create along the seams. The purpose of this feature is to get data parallelism at a course scale. Each of the inputs can be generated in a different process at the same time.

33.5 vtkAppendSelection

33.5.1 Usage

vtkAppendSelection is a filter that appends one of more selections into a single selection. All selections must have the same content type unless AppendByUnion is false.

To create an instance of class vtkAppendSelection, simply invoke its constructor as follows

```
obj = vtkAppendSelection
```

33.5.2 Methods

The class vtkAppendSelection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAppendSelection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAppendSelection = obj.NewInstance ()`
- `vtkAppendSelection = obj.SafeDownCast (vtkObject o)`
- `obj.SetUserManagedInputs (int )` - UserManagedInputs allows the user to set inputs by number instead of using the AddInput/RemoveInput functions. Calls to SetNumberOfInputs/SetInputByNumber should not be mixed with calls to AddInput/RemoveInput. By default, UserManagedInputs is false.
- `int = obj.GetUserManagedInputs ()` - UserManagedInputs allows the user to set inputs by number instead of using the AddInput/RemoveInput functions. Calls to SetNumberOfInputs/SetInputByNumber should not be mixed with calls to AddInput/RemoveInput. By default, UserManagedInputs is false.
- `obj.UserManagedInputsOn ()` - UserManagedInputs allows the user to set inputs by number instead of using the AddInput/RemoveInput functions. Calls to SetNumberOfInputs/SetInputByNumber should not be mixed with calls to AddInput/RemoveInput. By default, UserManagedInputs is false.
- `obj.UserManagedInputsOff ()` - UserManagedInputs allows the user to set inputs by number instead of using the AddInput/RemoveInput functions. Calls to SetNumberOfInputs/SetInputByNumber should not be mixed with calls to AddInput/RemoveInput. By default, UserManagedInputs is false.
- `obj.AddInput (vtkSelection )` - Add a dataset to the list of data to append. Should not be used when UserManagedInputs is true, use SetInputByNumber instead.
- `obj.RemoveInput (vtkSelection )` - Remove a dataset from the list of data to append. Should not be used when UserManagedInputs is true, use SetInputByNumber (NULL) instead.
- `obj.SetNumberOfInputs (int num)` - Directly set(allocate) number of inputs, should only be used when UserManagedInputs is true.
- `obj.SetInputByNumber (int num, vtkSelection input)`

- `obj.SetAppendByUnion (int )` - When set to true, all the selections are combined together to form a single vtkSelection output. When set to false, the output is a composite selection with input selections as the children of the composite selection. This allows for selections with different content types and properties. Default is true.
- `int = obj.GetAppendByUnion ()` - When set to true, all the selections are combined together to form a single vtkSelection output. When set to false, the output is a composite selection with input selections as the children of the composite selection. This allows for selections with different content types and properties. Default is true.
- `obj.AppendByUnionOn ()` - When set to true, all the selections are combined together to form a single vtkSelection output. When set to false, the output is a composite selection with input selections as the children of the composite selection. This allows for selections with different content types and properties. Default is true.
- `obj.AppendByUnionOff ()` - When set to true, all the selections are combined together to form a single vtkSelection output. When set to false, the output is a composite selection with input selections as the children of the composite selection. This allows for selections with different content types and properties. Default is true.

33.6 vtkApproximatingSubdivisionFilter

33.6.1 Usage

vtkApproximatingSubdivisionFilter is an abstract class that defines the protocol for Approximating subdivision surface filters.

To create an instance of class vtkApproximatingSubdivisionFilter, simply invoke its constructor as follows

```
obj = vtkApproximatingSubdivisionFilter
```

33.6.2 Methods

The class vtkApproximatingSubdivisionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkApproximatingSubdivisionFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkApproximatingSubdivisionFilter = obj.NewInstance ()`
- `vtkApproximatingSubdivisionFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfSubdivisions (int )` - Set/get the number of subdivisions.
- `int = obj.GetNumberOfSubdivisions ()` - Set/get the number of subdivisions.

33.7 vtkArcSource

33.7.1 Usage

vtkArcSource is a source object that creates an arc defined by two endpoints and a center. The number of segments composing the polyline is controlled by setting the object resolution.

To create an instance of class vtkArcSource, simply invoke its constructor as follows

```
obj = vtkArcSource
```

33.7.2 Methods

The class vtkArcSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArcSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArcSource = obj.NewInstance ()`
- `vtkArcSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetPoint1 (double , double , double )` - Set position of first end point.
- `obj.SetPoint1 (double  a[3])` - Set position of first end point.
- `double = obj. GetPoint1 ()` - Set position of first end point.
- `obj.SetPoint2 (double , double , double )` - Set position of other end point.
- `obj.SetPoint2 (double  a[3])` - Set position of other end point.
- `double = obj. GetPoint2 ()` - Set position of other end point.
- `obj.SetCenter (double , double , double )` - Set position of the center of the circle that define the arc. Note: you can use the function vtkMath::Solve3PointCircle to find the center from 3 points located on a circle.
- `obj.SetCenter (double  a[3])` - Set position of the center of the circle that define the arc. Note: you can use the function vtkMath::Solve3PointCircle to find the center from 3 points located on a circle.
- `double = obj. GetCenter ()` - Set position of the center of the circle that define the arc. Note: you can use the function vtkMath::Solve3PointCircle to find the center from 3 points located on a circle.
- `obj.SetResolution (int )` - Divide line into resolution number of pieces. Note: if Resolution is set to 1 (default), the arc is a straight line.
- `int = obj.GetResolutionMinValue ()` - Divide line into resolution number of pieces. Note: if Resolution is set to 1 (default), the arc is a straight line.
- `int = obj.GetResolutionMaxValue ()` - Divide line into resolution number of pieces. Note: if Resolution is set to 1 (default), the arc is a straight line.
- `int = obj.GetResolution ()` - Divide line into resolution number of pieces. Note: if Resolution is set to 1 (default), the arc is a straight line.
- `obj.SetNegative (bool )` - Use the angle that is a negative coterminal of the vectors angle: the longest angle. Note: false by default.
- `bool = obj.GetNegative ()` - Use the angle that is a negative coterminal of the vectors angle: the longest angle. Note: false by default.
- `obj.NegativeOn ()` - Use the angle that is a negative coterminal of the vectors angle: the longest angle. Note: false by default.
- `obj.NegativeOff ()` - Use the angle that is a negative coterminal of the vectors angle: the longest angle. Note: false by default.

33.8 vtkArrayCalculator

33.8.1 Usage

vtkArrayCalculator performs operations on vectors or scalars in field data arrays. It uses vtkFunctionParser to do the parsing and to evaluate the function for each entry in the input arrays. The arrays used in a given function must be all in point data or all in cell data. The resulting array will be stored as a field data array. The result array can either be stored in a new array or it can overwrite an existing array.

The functions that this array calculator understands is:

```
standard operations: + - * / \^ .
build unit vectors: iHat, jHat, kHat (ie (1,0,0), (0,1,0), (0,0,1))
abs
acos
asin
atan
ceil
cos
cosh
exp
floor
log
mag
min
max
norm
sign
sin
sinh
sqrt
tan
tanh
```

Note that some of these operations work on scalars, some on vectors, and some on both (e.g., you can multiply a scalar times a vector). The operations are performed tuple-wise (i.e., tuple-by-tuple). The user must specify which arrays to use as vectors and/or scalars, and the name of the output data array.

To create an instance of class vtkArrayCalculator, simply invoke its constructor as follows

```
obj = vtkArrayCalculator
```

33.8.2 Methods

The class vtkArrayCalculator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArrayCalculator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArrayCalculator = obj.NewInstance ()`
- `vtkArrayCalculator = obj.SafeDownCast (vtkObject o)`
- `obj.SetFunction (string function)` - Set/Get the function to be evaluated.

- `string = obj.GetFunction ()` - Set/Get the function to be evaluated.
- `obj.AddScalarArrayName (string arrayName, int component)` - Add an array name to the list of arrays used in the function and specify which components of the array to use in evaluating the function. The array name must match the name in the function. Use AddScalarVariable or AddVectorVariable to use a variable name different from the array name.
- `obj.AddVectorArrayName (string arrayName, int component0, int component1, int component2)` - Add an array name to the list of arrays used in the function and specify which components of the array to use in evaluating the function. The array name must match the name in the function. Use AddScalarVariable or AddVectorVariable to use a variable name different from the array name.
- `obj.AddScalarVariable (string variableName, string arrayName, int component)` - Add a variable name, a corresponding array name, and which components of the array to use.
- `obj.AddVectorVariable (string variableName, string arrayName, int component0, int component1, int c` - Add a variable name, a corresponding array name, and which components of the array to use.
- `obj.AddCoordinateScalarVariable (string variableName, int component)` - Add a variable name, a corresponding array name, and which components of the array to use.
- `obj.AddCoordinateVectorVariable (string variableName, int component0, int component1, int component` - Add a variable name, a corresponding array name, and which components of the array to use.
- `obj.SetResultArrayName (string name)` - Set the name of the array in which to store the result of evaluating this function. If this is the name of an existing array, that array will be overwritten. Otherwise a new array will be created with the specified name.
- `string = obj.GetResultArrayName ()` - Set the name of the array in which to store the result of evaluating this function. If this is the name of an existing array, that array will be overwritten. Otherwise a new array will be created with the specified name.
- `int = obj.GetResultArrayType ()` - Type of the result array. It is ignored if CoordinateResults is true. Initial value is VTK_DOUBLE.
- `obj.SetResultArrayType (int )` - Type of the result array. It is ignored if CoordinateResults is true. Initial value is VTK_DOUBLE.
- `int = obj.GetCoordinateResults ()` - Set whether to output results as coordinates. ResultArrayName will be ignored. Outputing as coordinates is only valid with vector results and if the AttributeMode is AttributeModeToUsePointData. If a valid output can't be made, an error will occur.
- `obj.SetCoordinateResults (int )` - Set whether to output results as coordinates. ResultArrayName will be ignored. Outputing as coordinates is only valid with vector results and if the AttributeMode is AttributeModeToUsePointData. If a valid output can't be made, an error will occur.
- `obj.CoordinateResultsOn ()` - Set whether to output results as coordinates. ResultArrayName will be ignored. Outputing as coordinates is only valid with vector results and if the AttributeMode is AttributeModeToUsePointData. If a valid output can't be made, an error will occur.
- `obj.CoordinateResultsOff ()` - Set whether to output results as coordinates. ResultArrayName will be ignored. Outputing as coordinates is only valid with vector results and if the AttributeMode is AttributeModeToUsePointData. If a valid output can't be made, an error will occur.
- `obj.SetAttributeMode (int )` - Control whether the filter operates on point data or cell data. By default (AttributeModeToDefault), the filter uses point data. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData). For graphs you can set the filter to use vertex data (AttributeModeToUseVertexData) or edge data (AttributeModeToUseEdgeData).

- `int = obj.GetAttributeMode ()` - Control whether the filter operates on point data or cell data. By default (AttributeModeToDefault), the filter uses point data. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData). For graphs you can set the filter to use vertex data (AttributeModeToUseVertexData) or edge data (AttributeModeToUseEdgeData).
- `obj.SetAttributeModeToDefault ()` - Control whether the filter operates on point data or cell data. By default (AttributeModeToDefault), the filter uses point data. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData). For graphs you can set the filter to use vertex data (AttributeModeToUseVertexData) or edge data (AttributeModeToUseEdgeData).
- `obj.SetAttributeModeToUsePointData ()` - Control whether the filter operates on point data or cell data. By default (AttributeModeToDefault), the filter uses point data. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData). For graphs you can set the filter to use vertex data (AttributeModeToUseVertexData) or edge data (AttributeModeToUseEdgeData).
- `obj.SetAttributeModeToUseCellData ()` - Control whether the filter operates on point data or cell data. By default (AttributeModeToDefault), the filter uses point data. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData). For graphs you can set the filter to use vertex data (AttributeModeToUseVertexData) or edge data (AttributeModeToUseEdgeData).
- `obj.SetAttributeModeToUseVertexData ()` - Control whether the filter operates on point data or cell data. By default (AttributeModeToDefault), the filter uses point data. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData). For graphs you can set the filter to use vertex data (AttributeModeToUseVertexData) or edge data (AttributeModeToUseEdgeData).
- `obj.SetAttributeModeToUseEdgeData ()` - Control whether the filter operates on point data or cell data. By default (AttributeModeToDefault), the filter uses point data. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData). For graphs you can set the filter to use vertex data (AttributeModeToUseVertexData) or edge data (AttributeModeToUseEdgeData).
- `string = obj.GetAttributeModeAsString ()` - Control whether the filter operates on point data or cell data. By default (AttributeModeToDefault), the filter uses point data. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData). For graphs you can set the filter to use vertex data (AttributeModeToUseVertexData) or edge data (AttributeModeToUseEdgeData).
- `obj.RemoveAllVariables ()` - Remove all the variable names and their associated array names.
- `obj.RemoveScalarVariables ()` - Remove all the scalar variable names and their associated array names.
- `obj.RemoveVectorVariables ()` - Remove all the scalar variable names and their associated array names.
- `obj.RemoveCoordinateScalarVariables ()` - Remove all the coordinate variables.
- `obj.RemoveCoordinateVectorVariables ()` - Remove all the coordinate variables.
- `string = obj.GetScalarArrayName (int i)` - Methods to get information about the current variables.
- `string = obj.GetVectorArrayName (int i)` - Methods to get information about the current variables.

- `string = obj.GetScalarVariableName (int i)` - Methods to get information about the current variables.
- `string = obj.GetVectorVariableName (int i)` - Methods to get information about the current variables.
- `int = obj.GetSelectedScalarComponent (int i)` - Methods to get information about the current variables.
- `int = obj.GetNumberOfScalarArrays ()` - Methods to get information about the current variables.
- `int = obj.GetNumberOfVectorArrays ()` - Methods to get information about the current variables.
- `obj.SetReplaceInvalidValues (int )` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `int = obj.GetReplaceInvalidValues ()` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `obj.ReplaceInvalidValuesOn ()` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `obj.ReplaceInvalidValuesOff ()` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `obj.SetReplacementValue (double )` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported
- `double = obj.GetReplacementValue ()` - When ReplaceInvalidValues is on, all invalid values (such as sqrt(-2), note that function parser does not handle complex numbers) will be replaced by ReplacementValue. Otherwise an error will be reported

33.9 vtkArrowSource

33.9.1 Usage

vtkArrowSource was intended to be used as the source for a glyph. The shaft base is always at (0,0,0). The arrow tip is always at (1,0,0). If "Invert" is true, then the ends are flipped i.e. tip is at (0,0,0) while base is at (1, 0, 0). The resolution of the cone and shaft can be set and default to 6. The radius of the cone and shaft can be set and default to 0.03 and 0.1. The length of the tip can also be set, and defaults to 0.35.

To create an instance of class vtkArrowSource, simply invoke its constructor as follows

```
obj = vtkArrowSource
```

33.9.2 Methods

The class vtkArrowSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArrowSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkArrowSource = obj.NewInstance ()`
- `vtkArrowSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetTipLength (double )` - Set the length, and radius of the tip. They default to 0.35 and 0.1
- `double = obj.GetTipLengthMinValue ()` - Set the length, and radius of the tip. They default to 0.35 and 0.1
- `double = obj.GetTipLengthMaxValue ()` - Set the length, and radius of the tip. They default to 0.35 and 0.1
- `double = obj.GetTipLength ()` - Set the length, and radius of the tip. They default to 0.35 and 0.1
- `obj.SetTipRadius (double )` - Set the length, and radius of the tip. They default to 0.35 and 0.1
- `double = obj.GetTipRadiusMinValue ()` - Set the length, and radius of the tip. They default to 0.35 and 0.1
- `double = obj.GetTipRadiusMaxValue ()` - Set the length, and radius of the tip. They default to 0.35 and 0.1
- `double = obj.GetTipRadius ()` - Set the length, and radius of the tip. They default to 0.35 and 0.1
- `obj.SetTipResolution (int )` - Set the resolution of the tip. The tip behaves the same as a cone. Resoultion 1 gives a single triangle, 2 gives two crossed triangles.
- `int = obj.GetTipResolutionMinValue ()` - Set the resolution of the tip. The tip behaves the same as a cone. Resoultion 1 gives a single triangle, 2 gives two crossed triangles.
- `int = obj.GetTipResolutionMaxValue ()` - Set the resolution of the tip. The tip behaves the same as a cone. Resoultion 1 gives a single triangle, 2 gives two crossed triangles.
- `int = obj.GetTipResolution ()` - Set the resolution of the tip. The tip behaves the same as a cone. Resoultion 1 gives a single triangle, 2 gives two crossed triangles.
- `obj.SetShaftRadius (double )` - Set the radius of the shaft. Defaults to 0.03.
- `double = obj.GetShaftRadiusMinValue ()` - Set the radius of the shaft. Defaults to 0.03.
- `double = obj.GetShaftRadiusMaxValue ()` - Set the radius of the shaft. Defaults to 0.03.
- `double = obj.GetShaftRadius ()` - Set the radius of the shaft. Defaults to 0.03.
- `obj.SetShaftResolution (int )` - Set the resolution of the shaft. 2 gives a rectangle. I would like to extend the cone to produce a line, but this is not an option now.
- `int = obj.GetShaftResolutionMinValue ()` - Set the resolution of the shaft. 2 gives a rectangle. I would like to extend the cone to produce a line, but this is not an option now.
- `int = obj.GetShaftResolutionMaxValue ()` - Set the resolution of the shaft. 2 gives a rectangle. I would like to extend the cone to produce a line, but this is not an option now.
- `int = obj.GetShaftResolution ()` - Set the resolution of the shaft. 2 gives a rectangle. I would like to extend the cone to produce a line, but this is not an option now.
- `obj.InvertOn ()` - Inverts the arrow direction. When set to true, base is at (1, 0, 0) while the tip is at (0, 0, 0). The default is false, i.e. base at (0, 0, 0) and the tip at (1, 0, 0).
- `obj.InvertOff ()` - Inverts the arrow direction. When set to true, base is at (1, 0, 0) while the tip is at (0, 0, 0). The default is false, i.e. base at (0, 0, 0) and the tip at (1, 0, 0).

- `obj.SetInvert (bool )` - Inverts the arrow direction. When set to true, base is at (1, 0, 0) while the tip is at (0, 0, 0). The default is false, i.e. base at (0, 0, 0) and the tip at (1, 0, 0).
- `bool = obj.GetInvert ()` - Inverts the arrow direction. When set to true, base is at (1, 0, 0) while the tip is at (0, 0, 0). The default is false, i.e. base at (0, 0, 0) and the tip at (1, 0, 0).

33.10 vtkAssignAttribute

33.10.1 Usage

vtkAssignAttribute is use to label a field (vtkDataArray) as an attribute. A field name or an attribute to labeled can be specified. For example: @verbatim aa-¿Assign("foo", vtkDataSetAttributes::SCALARS, vtkAssignAttribute::POINT_DATA); @endverbatim tells vtkAssignAttribute to make the array in the point data called "foo" the active scalars. On the other hand, @verbatim aa-¿Assign(vtkDataSetAttributes::VECTORS, vtkDataSetAttributes::SCALARS, vtkAssignAttribute::POINT_DATA); @endverbatim tells vtkAssignAttribute to make the active vectors also the active scalars. The same can be done more easily from Tcl by using the Assign() method which takes strings: @verbatim aa Assign "foo" SCALARS POINT_DATA or aa Assign SCALARS VECTORS POINT_DATA

AttributeTypes: SCALARS, VECTORS, NORMALS, TCOORDS, TENSORS Attribute locations: POINT_DATA, CELL_DATA @endverbatim

To create an instance of class vtkAssignAttribute, simply invoke its constructor as follows

```
obj = vtkAssignAttribute
```

33.10.2 Methods

The class vtkAssignAttribute has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAssignAttribute class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAssignAttribute = obj.NewInstance ()`
- `vtkAssignAttribute = obj.SafeDownCast (vtkObject o)`
- `obj.Assign (int inputAttributeType, int attributeType, int attributeLoc)` - Label an attribute as another attribute.
- `obj.Assign (string fieldName, int attributeType, int attributeLoc)` - Label an array as an attribute.
- `obj.Assign (string name, string attributeType, string attributeLoc)` - Helper method used by other language bindings. Allows the caller to specify arguments as strings instead of enums.

33.11 vtkAttributeDataToFieldDataFilter

33.11.1 Usage

vtkAttributeDataToFieldDataFilter is a class that maps attribute data into field data. Since this filter is a subclass of vtkDataSetAlgorithm, the output dataset (whose structure is the same as the input dataset), will contain the field data that is generated. The filter will convert point and cell attribute data to field data and assign it as point and cell field data, replacing any point or field data that was there previously. By default,

the original non-field point and cell attribute data will be passed to the output of the filter, although you can shut this behavior down.

To create an instance of class vtkAttributeDataToFieldDataFilter, simply invoke its constructor as follows

```
obj = vtkAttributeDataToFieldDataFilter
```

33.11.2 Methods

The class vtkAttributeDataToFieldDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAttributeDataToFieldDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAttributeDataToFieldDataFilter = obj.NewInstance ()`
- `vtkAttributeDataToFieldDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetPassAttributeData (int )` - Turn on/off the passing of point and cell non-field attribute data to the output of the filter.
- `int = obj.GetPassAttributeData ()` - Turn on/off the passing of point and cell non-field attribute data to the output of the filter.
- `obj.PassAttributeDataOn ()` - Turn on/off the passing of point and cell non-field attribute data to the output of the filter.
- `obj.PassAttributeDataOff ()` - Turn on/off the passing of point and cell non-field attribute data to the output of the filter.

33.12 vtkAxes

33.12.1 Usage

vtkAxes creates three lines that form an x-y-z axes. The origin of the axes is user specified (0,0,0 is default), and the size is specified with a scale factor. Three scalar values are generated for the three lines and can be used (via color map) to indicate a particular coordinate axis.

To create an instance of class vtkAxes, simply invoke its constructor as follows

```
obj = vtkAxes
```

33.12.2 Methods

The class vtkAxes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAxes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAxes = obj.NewInstance ()`
- `vtkAxes = obj.SafeDownCast (vtkObject o)`
- `obj.SetOrigin (double , double , double )` - Set the origin of the axes.

- `obj.SetOrigin (double  a[3])` - Set the origin of the axes.
- `double = obj. GetOrigin ()` - Set the origin of the axes.
- `obj.SetScaleFactor (double )` - Set the scale factor of the axes. Used to control size.
- `double = obj.GetScaleFactor ()` - Set the scale factor of the axes. Used to control size.
- `obj.SetSymmetric (int )` - If Symetric is on, the the axis continue to negative values.
- `int = obj.GetSymmetric ()` - If Symetric is on, the the axis continue to negative values.
- `obj.SymmetricOn ()` - If Symetric is on, the the axis continue to negative values.
- `obj.SymmetricOff ()` - If Symetric is on, the the axis continue to negative values.
- `obj.SetComputeNormals (int )` - Option for computing normals. By default they are computed.
- `int = obj.GetComputeNormals ()` - Option for computing normals. By default they are computed.
- `obj.ComputeNormalsOn ()` - Option for computing normals. By default they are computed.
- `obj.ComputeNormalsOff ()` - Option for computing normals. By default they are computed.

33.13 vtkBandedPolyDataContourFilter

33.13.1 Usage

vtkBandedPolyDataContourFilter is a filter that takes as input vtkPolyData and produces as output filled contours (also represented as vtkPolyData). Filled contours are bands of cells that all have the same cell scalar value, and can therefore be colored the same. The method is also referred to as filled contour generation.

To use this filter you must specify one or more contour values. You can either use the method SetValue() to specify each contour value, or use GenerateValues() to generate a series of evenly spaced contours. Each contour value divides (or clips) the data into two pieces, values below the contour value, and values above it. The scalar values of each band correspond to the specified contour value. Note that if the first and last contour values are not the minimum/maximum contour range, then two extra contour values are added corresponding to the minimum and maximum range values. These extra contour bands can be prevented from being output by turning clipping on.

To create an instance of class vtkBandedPolyDataContourFilter, simply invoke its constructor as follows

```
obj = vtkBandedPolyDataContourFilter
```

33.13.2 Methods

The class vtkBandedPolyDataContourFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBandedPolyDataContourFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBandedPolyDataContourFilter = obj.NewInstance ()`
- `vtkBandedPolyDataContourFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)` - Methods to set / get contour values. A single value at a time can be set with SetValue(). Multiple contour values can be set with GenerateValues(). Note that GenerateValues() generates n values inclusive of the start and end range values.

- `double = obj.GetValue (int i)` - Methods to set / get contour values. A single value at a time can be set with SetValue(). Multiple contour values can be set with GenerateValues(). Note that GenerateValues() generates n values inclusive of the start and end range values.

- `obj.GetValues (double contourValues)` - Methods to set / get contour values. A single value at a time can be set with SetValue(). Multiple contour values can be set with GenerateValues(). Note that GenerateValues() generates n values inclusive of the start and end range values.

- `obj.SetNumberOfContours (int number)` - Methods to set / get contour values. A single value at a time can be set with SetValue(). Multiple contour values can be set with GenerateValues(). Note that GenerateValues() generates n values inclusive of the start and end range values.

- `int = obj.GetNumberOfContours ()` - Methods to set / get contour values. A single value at a time can be set with SetValue(). Multiple contour values can be set with GenerateValues(). Note that GenerateValues() generates n values inclusive of the start and end range values.

- `obj.GenerateValues (int numContours, double range[2])` - Methods to set / get contour values. A single value at a time can be set with SetValue(). Multiple contour values can be set with GenerateValues(). Note that GenerateValues() generates n values inclusive of the start and end range values.

- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Methods to set / get contour values. A single value at a time can be set with SetValue(). Multiple contour values can be set with GenerateValues(). Note that GenerateValues() generates n values inclusive of the start and end range values.

- `obj.SetClipping (int )` - Indicate whether to clip outside the range specified by the user. (The range is contour value[0] to contour value[numContours-1].) Clipping means all cells outside of the range specified are not sent to the output.

- `int = obj.GetClipping ()` - Indicate whether to clip outside the range specified by the user. (The range is contour value[0] to contour value[numContours-1].) Clipping means all cells outside of the range specified are not sent to the output.

- `obj.ClippingOn ()` - Indicate whether to clip outside the range specified by the user. (The range is contour value[0] to contour value[numContours-1].) Clipping means all cells outside of the range specified are not sent to the output.

- `obj.ClippingOff ()` - Indicate whether to clip outside the range specified by the user. (The range is contour value[0] to contour value[numContours-1].) Clipping means all cells outside of the range specified are not sent to the output.

- `obj.SetScalarMode (int )` - Control whether the cell scalars are output as an integer index or a scalar value. If an index, the index refers to the bands produced by the clipping range. If a value, then a scalar value which is a value between clip values is used.

- `int = obj.GetScalarModeMinValue ()` - Control whether the cell scalars are output as an integer index or a scalar value. If an index, the index refers to the bands produced by the clipping range. If a value, then a scalar value which is a value between clip values is used.

- `int = obj.GetScalarModeMaxValue ()` - Control whether the cell scalars are output as an integer index or a scalar value. If an index, the index refers to the bands produced by the clipping range. If a value, then a scalar value which is a value between clip values is used.

- `int = obj.GetScalarMode ()` - Control whether the cell scalars are output as an integer index or a scalar value. If an index, the index refers to the bands produced by the clipping range. If a value, then a scalar value which is a value between clip values is used.

- `obj.SetScalarModeToIndex ()` - Control whether the cell scalars are output as an integer index or a scalar value. If an index, the index refers to the bands produced by the clipping range. If a value, then a scalar value which is a value between clip values is used.
- `obj.SetScalarModeToValue ()` - Turn on/off a flag to control whether contour edges are generated. Contour edges are the edges between bands. If enabled, they are generated from polygons/triangle strips and placed into the second output (the ContourEdgesOutput).
- `obj.SetGenerateContourEdges (int )` - Turn on/off a flag to control whether contour edges are generated. Contour edges are the edges between bands. If enabled, they are generated from polygons/triangle strips and placed into the second output (the ContourEdgesOutput).
- `int = obj.GetGenerateContourEdges ()` - Turn on/off a flag to control whether contour edges are generated. Contour edges are the edges between bands. If enabled, they are generated from polygons/triangle strips and placed into the second output (the ContourEdgesOutput).
- `obj.GenerateContourEdgesOn ()` - Turn on/off a flag to control whether contour edges are generated. Contour edges are the edges between bands. If enabled, they are generated from polygons/triangle strips and placed into the second output (the ContourEdgesOutput).
- `obj.GenerateContourEdgesOff ()` - Turn on/off a flag to control whether contour edges are generated. Contour edges are the edges between bands. If enabled, they are generated from polygons/triangle strips and placed into the second output (the ContourEdgesOutput).
- `vtkPolyData = obj.GetContourEdgesOutput ()` - Get the second output which contains the edges dividing the contour bands. This output is empty unless GenerateContourEdges is enabled.
- `long = obj.GetMTime ()` - Overload GetMTime because we delegate to vtkContourValues so its modified time must be taken into account.

33.14 vtkBlankStructuredGrid

33.14.1 Usage

vtkBlankStructuredGrid is a filter that sets the blanking field in a vtkStructuredGrid dataset. The blanking field is set by examining a specified point attribute data array (e.g., scalars) and converting values in the data array to either a "1" (visible) or "0" (blanked) value in the blanking array. The values to be blanked are specified by giving a min/max range. All data values in the data array indicated and laying within the range specified (inclusive on both ends) are translated to a "off" blanking value.

To create an instance of class vtkBlankStructuredGrid, simply invoke its constructor as follows

```
obj = vtkBlankStructuredGrid
```

33.14.2 Methods

The class vtkBlankStructuredGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBlankStructuredGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBlankStructuredGrid = obj.NewInstance ()`
- `vtkBlankStructuredGrid = obj.SafeDownCast (vtkObject o)`

- `obj.SetMinBlankingValue (double )` - Specify the lower data value in the data array specified which will be converted into a "blank" (or off) value in the blanking array.
- `double = obj.GetMinBlankingValue ()` - Specify the lower data value in the data array specified which will be converted into a "blank" (or off) value in the blanking array.
- `obj.SetMaxBlankingValue (double )` - Specify the upper data value in the data array specified which will be converted into a "blank" (or off) value in the blanking array.
- `double = obj.GetMaxBlankingValue ()` - Specify the upper data value in the data array specified which will be converted into a "blank" (or off) value in the blanking array.
- `obj.SetArrayName (string )` - Specify the data array name to use to generate the blanking field. Alternatively, you can specify the array id. (If both are set, the array name takes precedence.)
- `string = obj.GetArrayName ()` - Specify the data array name to use to generate the blanking field. Alternatively, you can specify the array id. (If both are set, the array name takes precedence.)
- `obj.SetArrayId (int )` - Specify the data array id to use to generate the blanking field. Alternatively, you can specify the array name. (If both are set, the array name takes precedence.)
- `int = obj.GetArrayId ()` - Specify the data array id to use to generate the blanking field. Alternatively, you can specify the array name. (If both are set, the array name takes precedence.)
- `obj.SetComponent (int )` - Specify the component in the data array to use to generate the blanking field.
- `int = obj.GetComponentMinValue ()` - Specify the component in the data array to use to generate the blanking field.
- `int = obj.GetComponentMaxValue ()` - Specify the component in the data array to use to generate the blanking field.
- `int = obj.GetComponent ()` - Specify the component in the data array to use to generate the blanking field.

33.15 vtkBlankStructuredGridWithImage

33.15.1 Usage

This filter can be used to set the blanking in a structured grid with an image. The filter takes two inputs: the structured grid to blank, and the image used to set the blanking. Make sure that the dimensions of both the image and the structured grid are identical.

Note that the image is interpreted as follows: zero values indicate that the structured grid point is blanked; non-zero values indicate that the structured grid point is visible. The blanking data must be unsigned char.

To create an instance of class vtkBlankStructuredGridWithImage, simply invoke its constructor as follows

```
obj = vtkBlankStructuredGridWithImage
```

33.15.2 Methods

The class vtkBlankStructuredGridWithImage has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBlankStructuredGridWithImage class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkBlankStructuredGridWithImage = obj.NewInstance ()`
- `vtkBlankStructuredGridWithImage = obj.SafeDownCast (vtkObject o)`
- `obj.SetBlankingInput (vtkImageData input)` - Set / get the input image used to perform the blanking.
- `vtkImageData = obj.GetBlankingInput ()` - Set / get the input image used to perform the blanking.

33.16 vtkBlockIdScalars

33.16.1 Usage

vtkBlockIdScalars is a filter that generates scalars using the block index for each block. Note that all sub-blocks within a block get the same scalar. The new scalars array is named ßlockIdScalars.

To create an instance of class vtkBlockIdScalars, simply invoke its constructor as follows

```
obj = vtkBlockIdScalars
```

33.16.2 Methods

The class vtkBlockIdScalars has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBlockIdScalars class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBlockIdScalars = obj.NewInstance ()`
- `vtkBlockIdScalars = obj.SafeDownCast (vtkObject o)`

33.17 vtkBoxClipDataSet

33.17.1 Usage

Clipping means that is actually 'cuts' through the cells of the dataset, returning tetrahedral cells inside of the box. The output of this filter is an unstructured grid.

This filter can be configured to compute a second output. The second output is the part of the cell that is clipped away. Set the GenerateClippedData boolean on if you wish to access this output data.

The vtkBoxClipDataSet will triangulate all types of 3D cells (i.e, create tetrahedra). This is necessary to preserve compatibility across face neighbors.

To use this filter,you can decide if you will be clipping with a box or a hexahedral box. 1) Set orientation if(SetOrientation(0)): box (parallel with coordinate axis) SetBoxClip(xmin,xmax,ymin,ymax,zmin,zmax) if(SetOrientation(1)): hexahedral box (Default) SetBoxClip(n[0],o[0],n[1],o[1],n[2],o[2],n[3],o[3],n[4],o[4],n[5],o[5]) PlaneNormal[] normal of each plane PlanePoint[] point on the plane 2) Apply the GenerateClipScalarsOn() 3) Execute clipping Update();

To create an instance of class vtkBoxClipDataSet, simply invoke its constructor as follows

```
obj = vtkBoxClipDataSet
```

33.17.2 Methods

The class vtkBoxClipDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBoxClipDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBoxClipDataSet = obj.NewInstance ()`
- `vtkBoxClipDataSet = obj.SafeDownCast (vtkObject o)`
- `obj.SetBoxClip (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)`
- `obj.SetBoxClip (double n0, double o0, double n1, double o1, double n2, double o2, double n3, double`
- `obj.SetGenerateClipScalars (int )` - If this flag is enabled, then the output scalar values will be interpolated, and not the input scalar data.
- `int = obj.GetGenerateClipScalars ()` - If this flag is enabled, then the output scalar values will be interpolated, and not the input scalar data.
- `obj.GenerateClipScalarsOn ()` - If this flag is enabled, then the output scalar values will be interpolated, and not the input scalar data.
- `obj.GenerateClipScalarsOff ()` - If this flag is enabled, then the output scalar values will be interpolated, and not the input scalar data.
- `obj.SetGenerateClippedOutput (int )` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `int = obj.GetGenerateClippedOutput ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `obj.GenerateClippedOutputOn ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `obj.GenerateClippedOutputOff ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `vtkUnstructuredGrid = obj.GetClippedOutput ()` - Return the Clipped output.
- `int = obj.GetNumberOfOutputs ()` - Return the Clipped output.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.
- `long = obj.GetMTime ()` - Return the mtime also considering the locator.
- `int = obj.GetOrientation ()` - Tells if clipping happens with a box parallel with coordinate axis (0) or with an hexahedral box (1). Initial value is 1.
- `obj.SetOrientation (int )` - Tells if clipping happens with a box parallel with coordinate axis (0) or with an hexahedral box (1). Initial value is 1.

- obj.ClipBox (vtkPoints newPoints, vtkGenericCell cell, vtkIncrementalPointLocator locator, vtkCellA
- obj.ClipHexahedron (vtkPoints newPoints, vtkGenericCell cell, vtkIncrementalPointLocator locator, v
- obj.ClipBox2D (vtkPoints newPoints, vtkGenericCell cell, vtkIncrementalPointLocator locator, vtkCel
- obj.ClipHexahedron2D (vtkPoints newPoints, vtkGenericCell cell, vtkIncrementalPointLocator locator,
- obj.ClipBox1D (vtkPoints newPoints, vtkGenericCell cell, vtkIncrementalPointLocator locator, vtkCel
- obj.ClipHexahedron1D (vtkPoints newPoints, vtkGenericCell cell, vtkIncrementalPointLocator locator,
- obj.ClipBox0D (vtkGenericCell cell, vtkIncrementalPointLocator locator, vtkCellArray verts, vtkPoi
- obj.ClipHexahedron0D (vtkGenericCell cell, vtkIncrementalPointLocator locator, vtkCellArray verts,

33.18 vtkBrownianPoints

33.18.1 Usage

vtkBrownianPoints is a filter object that assigns a random vector (i.e., magnitude and direction) to each point. The minimum and maximum speed values can be controlled by the user.

To create an instance of class vtkBrownianPoints, simply invoke its constructor as follows

```
obj = vtkBrownianPoints
```

33.18.2 Methods

The class vtkBrownianPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBrownianPoints class.

- string = obj.GetClassName ()
- int = obj.IsA (string name)
- vtkBrownianPoints = obj.NewInstance ()
- vtkBrownianPoints = obj.SafeDownCast (vtkObject o)
- obj.SetMinimumSpeed (double) - Set the minimum speed value.
- double = obj.GetMinimumSpeedMinValue () - Set the minimum speed value.
- double = obj.GetMinimumSpeedMaxValue () - Set the minimum speed value.
- double = obj.GetMinimumSpeed () - Set the minimum speed value.
- obj.SetMaximumSpeed (double) - Set the maximum speed value.
- double = obj.GetMaximumSpeedMinValue () - Set the maximum speed value.
- double = obj.GetMaximumSpeedMaxValue () - Set the maximum speed value.
- double = obj.GetMaximumSpeed () - Set the maximum speed value.

33.19 vtkButterflySubdivisionFilter

33.19.1 Usage

vtkButterflySubdivisionFilter is an interpolating subdivision scheme that creates four new triangles for each triangle in the mesh. The user can specify the NumberOfSubdivisions. This filter implements the 8-point butterfly scheme described in: Zorin, D., Schroder, P., and Sweldens, W., "Interpolating Subdivisions for Meshes with Arbitrary Topology," Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp.189-192. This scheme improves previous butterfly subdivisions with special treatment of vertices with valence other than 6.

Currently, the filter only operates on triangles. Users should use the vtkTriangleFilter to triangulate meshes that contain polygons or triangle strips.

The filter interpolates point data using the same scheme. New triangles created at a subdivision step will have the cell data of their parent cell.

To create an instance of class vtkButterflySubdivisionFilter, simply invoke its constructor as follows

```
obj = vtkButterflySubdivisionFilter
```

33.19.2 Methods

The class vtkButterflySubdivisionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkButterflySubdivisionFilter class.

- `string = obj.GetClassName ()` - Construct object with NumberOfSubdivisions set to 1.
- `int = obj.IsA (string name)` - Construct object with NumberOfSubdivisions set to 1.
- `vtkButterflySubdivisionFilter = obj.NewInstance ()` - Construct object with NumberOfSubdivisions set to 1.
- `vtkButterflySubdivisionFilter = obj.SafeDownCast (vtkObject o)` - Construct object with NumberOfSubdivisions set to 1.

33.20 vtkButtonSource

33.20.1 Usage

vtkButtonSource is an abstract class that defines an API for creating "button-like" objects in VTK. A button is a geometry with a rectangular region that can be textured. The button is divided into two regions: the texture region and the shoulder region. The points in both regions are assigned texture coordinates. The texture region has texture coordinates consistent with the image to be placed on it. All points in the shoulder regions are assigned a texture coordinate specified by the user. In this way the shoulder region can be colored by the texture.

Creating a vtkButtonSource requires specifying its center point. (Subclasses have other attributes that must be set to control the shape of the button.) You must also specify how to control the shape of the texture region; i.e., wheter to size the texture region proportional to the texture dimensions or whether to size the texture region proportional to the button. Also, buttons can be created single sided are mirrored to create two-sided buttons.

To create an instance of class vtkButtonSource, simply invoke its constructor as follows

```
obj = vtkButtonSource
```

33.20.2 Methods

The class vtkButtonSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkButtonSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkButtonSource = obj.NewInstance ()`
- `vtkButtonSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetCenter (double , double , double )` - Specify a point defining the origin (center) of the button.
- `obj.SetCenter (double a[3])` - Specify a point defining the origin (center) of the button.
- `double = obj. GetCenter ()` - Specify a point defining the origin (center) of the button.
- `obj.SetTextureStyle (int )` - Set/Get the style of the texture region: whether to size it according to the x-y dimensions of the texture, or whether to make the texture region proportional to the width/height of the button.
- `int = obj.GetTextureStyleMinValue ()` - Set/Get the style of the texture region: whether to size it according to the x-y dimensions of the texture, or whether to make the texture region proportional to the width/height of the button.
- `int = obj.GetTextureStyleMaxValue ()` - Set/Get the style of the texture region: whether to size it according to the x-y dimensions of the texture, or whether to make the texture region proportional to the width/height of the button.
- `int = obj.GetTextureStyle ()` - Set/Get the style of the texture region: whether to size it according to the x-y dimensions of the texture, or whether to make the texture region proportional to the width/height of the button.
- `obj.SetTextureStyleToFitImage ()` - Set/Get the style of the texture region: whether to size it according to the x-y dimensions of the texture, or whether to make the texture region proportional to the width/height of the button.
- `obj.SetTextureStyleToProportional ()` - Set/get the texture dimension. This needs to be set if the texture style is set to fit the image.
- `obj.SetTextureDimensions (int , int )` - Set/get the texture dimension. This needs to be set if the texture style is set to fit the image.
- `obj.SetTextureDimensions (int a[2])` - Set/get the texture dimension. This needs to be set if the texture style is set to fit the image.
- `int = obj. GetTextureDimensions ()` - Set/get the texture dimension. This needs to be set if the texture style is set to fit the image.
- `obj.SetShoulderTextureCoordinate (double , double )` - Set/Get the default texture coordinate to set the shoulder region to.
- `obj.SetShoulderTextureCoordinate (double a[2])` - Set/Get the default texture coordinate to set the shoulder region to.
- `double = obj. GetShoulderTextureCoordinate ()` - Set/Get the default texture coordinate to set the shoulder region to.

- `obj.SetTwoSided (int )` - Indicate whether the button is single or double sided. A double sided button can be viewed from two sides...it looks sort of like a "pill." A single-sided button is meant to viewed from a single side; it looks like a "clam-shell."
- `int = obj.GetTwoSided ()` - Indicate whether the button is single or double sided. A double sided button can be viewed from two sides...it looks sort of like a "pill." A single-sided button is meant to viewed from a single side; it looks like a "clam-shell."
- `obj.TwoSidedOn ()` - Indicate whether the button is single or double sided. A double sided button can be viewed from two sides...it looks sort of like a "pill." A single-sided button is meant to viewed from a single side; it looks like a "clam-shell."
- `obj.TwoSidedOff ()` - Indicate whether the button is single or double sided. A double sided button can be viewed from two sides...it looks sort of like a "pill." A single-sided button is meant to viewed from a single side; it looks like a "clam-shell."

33.21 vtkCellCenters

33.21.1 Usage

vtkCellCenters is a filter that takes as input any dataset and generates on output points at the center of the cells in the dataset. These points can be used for placing glyphs (vtkGlyph3D) or labeling (vtkLabeledDataMapper). (The center is the parametric center of the cell, not necessarily the geometric or bounding box center.) The cell attributes will be associated with the points on output.

To create an instance of class vtkCellCenters, simply invoke its constructor as follows

```
obj = vtkCellCenters
```

33.21.2 Methods

The class vtkCellCenters has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellCenters class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCellCenters = obj.NewInstance ()`
- `vtkCellCenters = obj.SafeDownCast (vtkObject o)`
- `obj.SetVertexCells (int )` - Enable/disable the generation of vertex cells. The default is Off.
- `int = obj.GetVertexCells ()` - Enable/disable the generation of vertex cells. The default is Off.
- `obj.VertexCellsOn ()` - Enable/disable the generation of vertex cells. The default is Off.
- `obj.VertexCellsOff ()` - Enable/disable the generation of vertex cells. The default is Off.

33.22 vtkCellDataToPointData

33.22.1 Usage

vtkCellDataToPointData is a filter that transforms cell data (i.e., data specified per cell) into point data (i.e., data specified at cell points). The method of transformation is based on averaging the data values of all cells using a particular point. Optionally, the input cell data can be passed through to the output as well.

To create an instance of class vtkCellDataToPointData, simply invoke its constructor as follows

```
obj = vtkCellDataToPointData
```

33.22.2 Methods

The class vtkCellDataToPointData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellDataToPointData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCellDataToPointData = obj.NewInstance ()`
- `vtkCellDataToPointData = obj.SafeDownCast (vtkObject o)`
- `obj.SetPassCellData (int )` - Control whether the input cell data is to be passed to the output. If on, then the input cell data is passed through to the output; otherwise, only generated point data is placed into the output.
- `int = obj.GetPassCellData ()` - Control whether the input cell data is to be passed to the output. If on, then the input cell data is passed through to the output; otherwise, only generated point data is placed into the output.
- `obj.PassCellDataOn ()` - Control whether the input cell data is to be passed to the output. If on, then the input cell data is passed through to the output; otherwise, only generated point data is placed into the output.
- `obj.PassCellDataOff ()` - Control whether the input cell data is to be passed to the output. If on, then the input cell data is passed through to the output; otherwise, only generated point data is placed into the output.

33.23 vtkCellDerivatives

33.23.1 Usage

vtkCellDerivatives is a filter that computes derivatives of scalars and vectors at the center of cells. You can choose to generate different output including the scalar gradient (a vector), computed tensor vorticity (a vector), gradient of input vectors (a tensor), and strain matrix of the input vectors (a tensor); or you may choose to pass data through to the output.

Note that it is assumed that on input scalars and vector point data is available, which are then used to generate cell vectors and tensors. (The interpolation functions of the cells are used to compute the derivatives which is why point data is required.)

To create an instance of class vtkCellDerivatives, simply invoke its constructor as follows

```
obj = vtkCellDerivatives
```

33.23.2 Methods

The class vtkCellDerivatives has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellDerivatives class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkCellDerivatives = obj.NewInstance ()`
- `vtkCellDerivatives = obj.SafeDownCast (vtkObject o)`
- `obj.SetVectorMode (int )` - Control how the filter works to generate vector cell data. You can choose to pass the input cell vectors, compute the gradient of the input scalars, or extract the vorticity of the computed vector gradient tensor. By default (VectorModeToComputeGradient), the filter will take the gradient of the input scalar data.
- `int = obj.GetVectorMode ()` - Control how the filter works to generate vector cell data. You can choose to pass the input cell vectors, compute the gradient of the input scalars, or extract the vorticity of the computed vector gradient tensor. By default (VectorModeToComputeGradient), the filter will take the gradient of the input scalar data.
- `obj.SetVectorModeToPassVectors ()` - Control how the filter works to generate vector cell data. You can choose to pass the input cell vectors, compute the gradient of the input scalars, or extract the vorticity of the computed vector gradient tensor. By default (VectorModeToComputeGradient), the filter will take the gradient of the input scalar data.
- `obj.SetVectorModeToComputeGradient ()` - Control how the filter works to generate vector cell data. You can choose to pass the input cell vectors, compute the gradient of the input scalars, or extract the vorticity of the computed vector gradient tensor. By default (VectorModeToComputeGradient), the filter will take the gradient of the input scalar data.
- `obj.SetVectorModeToComputeVorticity ()` - Control how the filter works to generate vector cell data. You can choose to pass the input cell vectors, compute the gradient of the input scalars, or extract the vorticity of the computed vector gradient tensor. By default (VectorModeToComputeGradient), the filter will take the gradient of the input scalar data.
- `string = obj.GetVectorModeAsString ()` - Control how the filter works to generate vector cell data. You can choose to pass the input cell vectors, compute the gradient of the input scalars, or extract the vorticity of the computed vector gradient tensor. By default (VectorModeToComputeGradient), the filter will take the gradient of the input scalar data.
- `obj.SetTensorMode (int )` - Control how the filter works to generate tensor cell data. You can choose to pass the input cell tensors, compute the gradient of the input vectors, or compute the strain tensor of the vector gradient tensor. By default (TensorModeToComputeGradient), the filter will take the gradient of the vector data to construct a tensor.
- `int = obj.GetTensorMode ()` - Control how the filter works to generate tensor cell data. You can choose to pass the input cell tensors, compute the gradient of the input vectors, or compute the strain tensor of the vector gradient tensor. By default (TensorModeToComputeGradient), the filter will take the gradient of the vector data to construct a tensor.
- `obj.SetTensorModeToPassTensors ()` - Control how the filter works to generate tensor cell data. You can choose to pass the input cell tensors, compute the gradient of the input vectors, or compute the strain tensor of the vector gradient tensor. By default (TensorModeToComputeGradient), the filter will take the gradient of the vector data to construct a tensor.
- `obj.SetTensorModeToComputeGradient ()` - Control how the filter works to generate tensor cell data. You can choose to pass the input cell tensors, compute the gradient of the input vectors, or compute the strain tensor of the vector gradient tensor. By default (TensorModeToComputeGradient), the filter will take the gradient of the vector data to construct a tensor.
- `obj.SetTensorModeToComputeStrain ()` - Control how the filter works to generate tensor cell data. You can choose to pass the input cell tensors, compute the gradient of the input vectors, or compute the strain tensor of the vector gradient tensor. By default (TensorModeToComputeGradient), the filter will take the gradient of the vector data to construct a tensor.

- `string = obj.GetTensorModeAsString ()` - Control how the filter works to generate tensor cell data. You can choose to pass the input cell tensors, compute the gradient of the input vectors, or compute the strain tensor of the vector gradient tensor. By default (TensorModeToComputeGradient), the filter will take the gradient of the vector data to construct a tensor.

33.24 vtkCleanPolyData

33.24.1 Usage

vtkCleanPolyData is a filter that takes polygonal data as input and generates polygonal data as output. vtkCleanPolyData can merge duplicate points (within specified tolerance and if enabled), eliminate points that are not used, and if enabled, transform degenerate cells into appropriate forms (for example, a triangle is converted into a line if two points of triangle are merged).

Conversion of degenerate cells is controlled by the flags ConvertLinesToPoints, ConvertPolysToLines, ConvertStripsToPolys which act cumulatively such that a degenerate strip may become a poly. The full set is Line with 1 points -¿ Vert (if ConvertLinesToPoints) Poly with 2 points -¿ Line (if ConvertPolysToLines) Poly with 1 points -¿ Vert (if ConvertPolysToLines && ConvertLinesToPoints) Strp with 3 points -¿ Poly (if ConvertStripsToPolys) Strp with 2 points -¿ Line (if ConvertStripsToPolys && ConvertPolysToLines) Strp with 1 points -¿ Vert (if ConvertStripsToPolys && ConvertPolysToLines && ConvertLinesToPoints)

If tolerance is specified precisely=0.0, then vtkCleanPolyData will use the vtkMergePoints object to merge points (which is faster). Otherwise the slower vtkIncrementalPointLocator is used. Before inserting points into the point locator, this class calls a function OperateOnPoint which can be used (in subclasses) to further refine the cleaning process. See vtkQuantizePolyDataPoints.

Note that merging of points can be disabled. In this case, a point locator will not be used, and points that are not used by any cells will be eliminated, but never merged.

To create an instance of class vtkCleanPolyData, simply invoke its constructor as follows

```
obj = vtkCleanPolyData
```

33.24.2 Methods

The class vtkCleanPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCleanPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCleanPolyData = obj.NewInstance ()`
- `vtkCleanPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetToleranceIsAbsolute (int )` - By default ToleranceIsAbsolute is false and Tolerance is a fraction of Bounding box diagonal, if true, AbsoluteTolerance is used when adding points to locator (merging)
- `obj.ToleranceIsAbsoluteOn ()` - By default ToleranceIsAbsolute is false and Tolerance is a fraction of Bounding box diagonal, if true, AbsoluteTolerance is used when adding points to locator (merging)
- `obj.ToleranceIsAbsoluteOff ()` - By default ToleranceIsAbsolute is false and Tolerance is a fraction of Bounding box diagonal, if true, AbsoluteTolerance is used when adding points to locator (merging)
- `int = obj.GetToleranceIsAbsolute ()` - By default ToleranceIsAbsolute is false and Tolerance is a fraction of Bounding box diagonal, if true, AbsoluteTolerance is used when adding points to locator (merging)

- `obj.SetTolerance (double )` - Specify tolerance in terms of fraction of bounding box length.
- `double = obj.GetToleranceMinValue ()` - Specify tolerance in terms of fraction of bounding box length.
- `double = obj.GetToleranceMaxValue ()` - Specify tolerance in terms of fraction of bounding box length.
- `double = obj.GetTolerance ()` - Specify tolerance in terms of fraction of bounding box length.
- `obj.SetAbsoluteTolerance (double )` - Specify tolerance in absolute terms
- `double = obj.GetAbsoluteToleranceMinValue ()` - Specify tolerance in absolute terms
- `double = obj.GetAbsoluteToleranceMaxValue ()` - Specify tolerance in absolute terms
- `double = obj.GetAbsoluteTolerance ()` - Specify tolerance in absolute terms
- `obj.SetConvertLinesToPoints (int )` - Turn on/off conversion of degenerate lines to points
- `obj.ConvertLinesToPointsOn ()` - Turn on/off conversion of degenerate lines to points
- `obj.ConvertLinesToPointsOff ()` - Turn on/off conversion of degenerate lines to points
- `int = obj.GetConvertLinesToPoints ()` - Turn on/off conversion of degenerate lines to points
- `obj.SetConvertPolysToLines (int )` - Turn on/off conversion of degenerate polys to lines
- `obj.ConvertPolysToLinesOn ()` - Turn on/off conversion of degenerate polys to lines
- `obj.ConvertPolysToLinesOff ()` - Turn on/off conversion of degenerate polys to lines
- `int = obj.GetConvertPolysToLines ()` - Turn on/off conversion of degenerate polys to lines
- `obj.SetConvertStripsToPolys (int )` - Turn on/off conversion of degenerate strips to polys
- `obj.ConvertStripsToPolysOn ()` - Turn on/off conversion of degenerate strips to polys
- `obj.ConvertStripsToPolysOff ()` - Turn on/off conversion of degenerate strips to polys
- `int = obj.GetConvertStripsToPolys ()` - Turn on/off conversion of degenerate strips to polys
- `obj.SetPointMerging (int )` - Set/Get a boolean value that controls whether point merging is performed. If on, a locator will be used, and points laying within the appropriate tolerance may be merged. If off, points are never merged. By default, merging is on.
- `int = obj.GetPointMerging ()` - Set/Get a boolean value that controls whether point merging is performed. If on, a locator will be used, and points laying within the appropriate tolerance may be merged. If off, points are never merged. By default, merging is on.
- `obj.PointMergingOn ()` - Set/Get a boolean value that controls whether point merging is performed. If on, a locator will be used, and points laying within the appropriate tolerance may be merged. If off, points are never merged. By default, merging is on.
- `obj.PointMergingOff ()` - Set/Get a boolean value that controls whether point merging is performed. If on, a locator will be used, and points laying within the appropriate tolerance may be merged. If off, points are never merged. By default, merging is on.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set/Get a spatial locator for speeding the search process. By default an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set/Get a spatial locator for speeding the search process. By default an instance of vtkMergePoints is used.

- `obj.CreateDefaultLocator (vtkPolyData input)` - Create default locator. Used to create one when none is specified.
- `obj.ReleaseLocator ()` - Get the MTime of this object also considering the locator.
- `long = obj.GetMTime ()` - Get the MTime of this object also considering the locator.
- `obj.OperateOnPoint (double in[3], double out[3])` - Perform operation on a point
- `obj.OperateOnBounds (double in[6], double out[6])` - Perform operation on bounds
- `obj.SetPieceInvariant (int )`
- `int = obj.GetPieceInvariant ()`
- `obj.PieceInvariantOn ()`
- `obj.PieceInvariantOff ()`

33.25 vtkClipConvexPolyData

33.25.1 Usage

vtkClipConvexPolyData is a filter that clips a convex polydata with a set of planes. Its main usage is for clipping a bounding volume with frustum planes (used later one in volume rendering).

To create an instance of class vtkClipConvexPolyData, simply invoke its constructor as follows

```
obj = vtkClipConvexPolyData
```

33.25.2 Methods

The class vtkClipConvexPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkClipConvexPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkClipConvexPolyData = obj.NewInstance ()`
- `vtkClipConvexPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetPlanes (vtkPlaneCollection planes)` - Set all the planes at once using a vtkPlanes implicit function. This also sets the D value.
- `vtkPlaneCollection = obj.GetPlanes ()` - Set all the planes at once using a vtkPlanes implicit function. This also sets the D value.
- `long = obj.GetMTime ()` - Redefines this method, as this filter depends on time of its components (planes)

33.26 vtkClipDataSet

33.26.1 Usage

vtkClipDataSet is a filter that clips any type of dataset using either any subclass of vtkImplicitFunction, or the input scalar data. Clipping means that it actually "cuts" through the cells of the dataset, returning everything inside of the specified implicit function (or greater than the scalar value) including "pieces" of a cell. (Compare this with vtkExtractGeometry, which pulls out entire, uncut cells.) The output of this filter is an unstructured grid.

To use this filter, you must decide if you will be clipping with an implicit function, or whether you will be using the input scalar data. If you want to clip with an implicit function, you must: 1) define an implicit function 2) set it with the SetClipFunction method 3) apply the GenerateClipScalarsOn method If a ClipFunction is not specified, or GenerateClipScalars is off (the default), then the input's scalar data will be used to clip the polydata.

You can also specify a scalar value, which is used to decide what is inside and outside of the implicit function. You can also reverse the sense of what inside/outside is by setting the InsideOut instance variable. (The clipping algorithm proceeds by computing an implicit function value or using the input scalar data for each point in the dataset. This is compared to the scalar value to determine inside/outside.)

This filter can be configured to compute a second output. The second output is the part of the cell that is clipped away. Set the GenerateClippedData boolean on if you wish to access this output data.

To create an instance of class vtkClipDataSet, simply invoke its constructor as follows

```
obj = vtkClipDataSet
```

33.26.2 Methods

The class vtkClipDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkClipDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkClipDataSet = obj.NewInstance ()`
- `vtkClipDataSet = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (double )` - Set the clipping value of the implicit function (if clipping with implicit function) or scalar value (if clipping with scalars). The default value is 0.0. This value is ignored if UseValueAsOffset is true and a clip function is defined.
- `double = obj.GetValue ()` - Set the clipping value of the implicit function (if clipping with implicit function) or scalar value (if clipping with scalars). The default value is 0.0. This value is ignored if UseValueAsOffset is true and a clip function is defined.
- `obj.SetUseValueAsOffset (bool )` - If UseValueAsOffset is true, Value is used as an offset parameter to the implicit function. Otherwise, Value is used only when clipping using a scalar array. Default is true.
- `bool = obj.GetUseValueAsOffset ()` - If UseValueAsOffset is true, Value is used as an offset parameter to the implicit function. Otherwise, Value is used only when clipping using a scalar array. Default is true.
- `obj.UseValueAsOffsetOn ()` - If UseValueAsOffset is true, Value is used as an offset parameter to the implicit function. Otherwise, Value is used only when clipping using a scalar array. Default is true.

- `obj.UseValueAsOffsetOff ()` - If UseValueAsOffset is true, Value is used as an offset parameter to the implicit function. Otherwise, Value is used only when clipping using a scalar array. Default is true.
- `obj.SetInsideOut (int )` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `int = obj.GetInsideOut ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `obj.InsideOutOn ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `obj.InsideOutOff ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `obj.SetClipFunction (vtkImplicitFunction )`
- `vtkImplicitFunction = obj.GetClipFunction ()`
- `obj.SetGenerateClipScalars (int )` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `int = obj.GetGenerateClipScalars ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `obj.GenerateClipScalarsOn ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `obj.GenerateClipScalarsOff ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `obj.SetGenerateClippedOutput (int )` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `int = obj.GetGenerateClippedOutput ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `obj.GenerateClippedOutputOn ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `obj.GenerateClippedOutputOff ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `obj.SetMergeTolerance (double )` - Set the tolerance for merging clip intersection points that are near the vertices of cells. This tolerance is used to prevent the generation of degenerate primitives. Note that only 3D cells actually use this instance variable.

- `double = obj.GetMergeToleranceMinValue ()` - Set the tolerance for merging clip intersection points that are near the vertices of cells. This tolerance is used to prevent the generation of degenerate primitives. Note that only 3D cells actually use this instance variable.
- `double = obj.GetMergeToleranceMaxValue ()` - Set the tolerance for merging clip intersection points that are near the vertices of cells. This tolerance is used to prevent the generation of degenerate primitives. Note that only 3D cells actually use this instance variable.
- `double = obj.GetMergeTolerance ()` - Set the tolerance for merging clip intersection points that are near the vertices of cells. This tolerance is used to prevent the generation of degenerate primitives. Note that only 3D cells actually use this instance variable.
- `vtkUnstructuredGrid = obj.GetClippedOutput ()` - Return the Clipped output.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.
- `long = obj.GetMTime ()` - Return the mtime also considering the locator and clip function.

33.27 vtkClipHyperOctree

33.27.1 Usage

vtkClipHyperOctree is a filter that clips an hyperoctree using either any subclass of vtkImplicitFunction, or the input scalar data. Clipping means that it actually "cuts" through the leaves (cells) of the hyperoctree, returning everything inside of the specified implicit function (or greater than the scalar value) including "pieces" of a cell. (Compare this with vtkExtractGeometry, which pulls out entire, uncut cells.) The output of this filter is an unstructured grid.

To use this filter, you must decide if you will be clipping with an implicit function, or whether you will be using the input scalar data. If you want to clip with an implicit function, you must: 1) define an implicit function 2) set it with the SetClipFunction method 3) apply the GenerateClipScalarsOn method If a ClipFunction is not specified, or GenerateClipScalars is off (the default), then the input's scalar data will be used to clip the polydata.

You can also specify a scalar value, which is used to decide what is inside and outside of the implicit function. You can also reverse the sense of what inside/outside is by setting the InsideOut instance variable. (The clipping algorithm proceeds by computing an implicit function value or using the input scalar data for each point in the dataset. This is compared to the scalar value to determine inside/outside.)

This filter can be configured to compute a second output. The second output is the part of the cell that is clipped away. Set the GenerateClippedData boolean on if you wish to access this output data.

To create an instance of class vtkClipHyperOctree, simply invoke its constructor as follows

```
obj = vtkClipHyperOctree
```

33.27.2 Methods

The class vtkClipHyperOctree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkClipHyperOctree class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkClipHyperOctree = obj.NewInstance ()`
- `vtkClipHyperOctree = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (double )` - Set the clipping value of the implicit function (if clipping with implicit function) or scalar value (if clipping with scalars). The default value is 0.0.
- `double = obj.GetValue ()` - Set the clipping value of the implicit function (if clipping with implicit function) or scalar value (if clipping with scalars). The default value is 0.0.
- `obj.SetInsideOut (int )` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `int = obj.GetInsideOut ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `obj.InsideOutOn ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `obj.InsideOutOff ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `obj.SetClipFunction (vtkImplicitFunction )`
- `vtkImplicitFunction = obj.GetClipFunction ()`
- `obj.SetGenerateClipScalars (int )` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `int = obj.GetGenerateClipScalars ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `obj.GenerateClipScalarsOn ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `obj.GenerateClipScalarsOff ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `obj.SetGenerateClippedOutput (int )` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `int = obj.GetGenerateClippedOutput ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `obj.GenerateClippedOutputOn ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.

- `obj.GenerateClippedOutputOff ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `vtkUnstructuredGrid = obj.GetClippedOutput ()` - Return the Clipped output.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.
- `long = obj.GetMTime ()` - Return the mtime also considering the locator and clip function.

33.28 vtkClipPolyData

33.28.1 Usage

vtkClipPolyData is a filter that clips polygonal data using either any subclass of vtkImplicitFunction, or the input scalar data. Clipping means that it actually "cuts" through the cells of the dataset, returning everything inside of the specified implicit function (or greater than the scalar value) including "pieces" of a cell. (Compare this with vtkExtractGeometry, which pulls out entire, uncut cells.) The output of this filter is polygonal data.

To use this filter, you must decide if you will be clipping with an implicit function, or whether you will be using the input scalar data. If you want to clip with an implicit function, you must: 1) define an implicit function 2) set it with the SetClipFunction method 3) apply the GenerateClipScalarsOn method If a ClipFunction is not specified, or GenerateClipScalars is off (the default), then the input's scalar data will be used to clip the polydata.

You can also specify a scalar value, which is used to decide what is inside and outside of the implicit function. You can also reverse the sense of what inside/outside is by setting the InsideOut instance variable. (The cutting algorithm proceeds by computing an implicit function value or using the input scalar data for each point in the dataset. This is compared to the scalar value to determine inside/outside.)

This filter can be configured to compute a second output. The second output is the polygonal data that is clipped away. Set the GenerateClippedData boolean on if you wish to access this output data.

To create an instance of class vtkClipPolyData, simply invoke its constructor as follows

```
obj = vtkClipPolyData
```

33.28.2 Methods

The class vtkClipPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkClipPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkClipPolyData = obj.NewInstance ()`
- `vtkClipPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (double )` - Set the clipping value of the implicit function (if clipping with implicit function) or scalar value (if clipping with scalars). The default value is 0.0.

- `double = obj.GetValue ()` - Set the clipping value of the implicit function (if clipping with implicit function) or scalar value (if clipping with scalars). The default value is 0.0.
- `obj.SetInsideOut (int )` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `int = obj.GetInsideOut ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `obj.InsideOutOn ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `obj.InsideOutOff ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.
- `obj.SetClipFunction (vtkImplicitFunction )`
- `vtkImplicitFunction = obj.GetClipFunction ()`
- `obj.SetGenerateClipScalars (int )` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `int = obj.GetGenerateClipScalars ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `obj.GenerateClipScalarsOn ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `obj.GenerateClipScalarsOff ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.
- `obj.SetGenerateClippedOutput (int )` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `int = obj.GetGenerateClippedOutput ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `obj.GenerateClippedOutputOn ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `obj.GenerateClippedOutputOff ()` - Control whether a second output is generated. The second output contains the polygonal data that's been clipped away.
- `vtkPolyData = obj.GetClippedOutput ()` - Return the Clipped output.
- `vtkAlgorithmOutput = obj.GetClippedOutputPort ()` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.

- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.
- `long = obj.GetMTime ()` - Return the mtime also considering the locator and clip function.

33.29 vtkClipVolume

33.29.1 Usage

vtkClipVolume is a filter that clips volume data (i.e., vtkImageData) using either: any subclass of vtkImplicitFunction or the input scalar data. The clipping operation cuts through the cells of the dataset–converting 3D image data into a 3D unstructured grid–returning everything inside of the specified implicit function (or greater than the scalar value). During the clipping the filter will produce pieces of a cell. (Compare this with vtkExtractGeometry or vtkGeometryFilter, which produces entire, uncut cells.) The output of this filter is a 3D unstructured grid (e.g., tetrahedra or other 3D cell types).

To use this filter, you must decide if you will be clipping with an implicit function, or whether you will be using the input scalar data. If you want to clip with an implicit function, you must first define and then set the implicit function with the SetClipFunction() method. Otherwise, you must make sure input scalar data is available. You can also specify a scalar value, which is used to decide what is inside and outside of the implicit function. You can also reverse the sense of what inside/outside is by setting the InsideOut instance variable. (The cutting algorithm proceeds by computing an implicit function value or using the input scalar data for each point in the dataset. This is compared to the scalar value to determine inside/outside.)

This filter can be configured to compute a second output. The second output is the portion of the volume that is clipped away. Set the GenerateClippedData boolean on if you wish to access this output data.

The filter will produce an unstructured grid of entirely tetrahedra or a mixed grid of tetrahedra and other 3D cell types (e.g., wedges). Control this behavior by setting the Mixed3DCellGeneration. By default the Mixed3DCellGeneration is on and a combination of cell types will be produced. Note that producing mixed cell types is a faster than producing only tetrahedra.

To create an instance of class vtkClipVolume, simply invoke its constructor as follows

```
obj = vtkClipVolume
```

33.29.2 Methods

The class vtkClipVolume has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkClipVolume class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkClipVolume = obj.NewInstance ()`
- `vtkClipVolume = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (double )` - Set the clipping value of the implicit function (if clipping with implicit function) or scalar value (if clipping with scalars). The default value is 0.0.
- `double = obj.GetValue ()` - Set the clipping value of the implicit function (if clipping with implicit function) or scalar value (if clipping with scalars). The default value is 0.0.

- `obj.SetInsideOut (int )` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.

- `int = obj.GetInsideOut ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.

- `obj.InsideOutOn ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.

- `obj.InsideOutOff ()` - Set/Get the InsideOut flag. When off, a vertex is considered inside the implicit function if its value is greater than the Value ivar. When InsideOutside is turned on, a vertex is considered inside the implicit function if its implicit function value is less than or equal to the Value ivar. InsideOut is off by default.

- `obj.SetClipFunction (vtkImplicitFunction )`

- `vtkImplicitFunction = obj.GetClipFunction ()`

- `obj.SetGenerateClipScalars (int )` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.

- `int = obj.GetGenerateClipScalars ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.

- `obj.GenerateClipScalarsOn ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.

- `obj.GenerateClipScalarsOff ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data. If you enable this flag but do not provide an implicit function an error will be reported.

- `obj.SetGenerateClippedOutput (int )` - Control whether a second output is generated. The second output contains the unstructured grid that's been clipped away.

- `int = obj.GetGenerateClippedOutput ()` - Control whether a second output is generated. The second output contains the unstructured grid that's been clipped away.

- `obj.GenerateClippedOutputOn ()` - Control whether a second output is generated. The second output contains the unstructured grid that's been clipped away.

- `obj.GenerateClippedOutputOff ()` - Control whether a second output is generated. The second output contains the unstructured grid that's been clipped away.

- `vtkUnstructuredGrid = obj.GetClippedOutput ()` - Return the clipped output.

- `obj.SetMixed3DCellGeneration (int )` - Control whether the filter produces a mix of 3D cell types on output, or whether the output cells are all tetrahedra. By default, a mixed set of cells (e.g., tetrahedra and wedges) is produced. (Note: mixed type generation is faster and less overall data is generated.)

- `int = obj.GetMixed3DCellGeneration ()` - Control whether the filter produces a mix of 3D cell types on output, or whether the output cells are all tetrahedra. By default, a mixed set of cells (e.g., tetrahedra and wedges) is produced. (Note: mixed type generation is faster and less overall data is generated.)
- `obj.Mixed3DCellGenerationOn ()` - Control whether the filter produces a mix of 3D cell types on output, or whether the output cells are all tetrahedra. By default, a mixed set of cells (e.g., tetrahedra and wedges) is produced. (Note: mixed type generation is faster and less overall data is generated.)
- `obj.Mixed3DCellGenerationOff ()` - Control whether the filter produces a mix of 3D cell types on output, or whether the output cells are all tetrahedra. By default, a mixed set of cells (e.g., tetrahedra and wedges) is produced. (Note: mixed type generation is faster and less overall data is generated.)
- `obj.SetMergeTolerance (double )` - Set the tolerance for merging clip intersection points that are near the corners of voxels. This tolerance is used to prevent the generation of degenerate tetrahedra.
- `double = obj.GetMergeToleranceMinValue ()` - Set the tolerance for merging clip intersection points that are near the corners of voxels. This tolerance is used to prevent the generation of degenerate tetrahedra.
- `double = obj.GetMergeToleranceMaxValue ()` - Set the tolerance for merging clip intersection points that are near the corners of voxels. This tolerance is used to prevent the generation of degenerate tetrahedra.
- `double = obj.GetMergeTolerance ()` - Set the tolerance for merging clip intersection points that are near the corners of voxels. This tolerance is used to prevent the generation of degenerate tetrahedra.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / Get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / Get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.
- `long = obj.GetMTime ()` - Return the mtime also considering the locator and clip function.

33.30 vtkCoincidentPoints

33.30.1 Usage

This class provides a collection of points that is organized such that each coordinate is stored with a set of point id's of points that are all coincident.

To create an instance of class vtkCoincidentPoints, simply invoke its constructor as follows

```
obj = vtkCoincidentPoints
```

33.30.2 Methods

The class vtkCoincidentPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCoincidentPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkCoincidentPoints = obj.NewInstance ()`
- `vtkCoincidentPoints = obj.SafeDownCast (vtkObject o)`
- `obj.AddPoint (vtkIdType Id, double point[3])` - Accumulates a set of Ids in a map where the point coordinate is the key. All Ids in a given map entry are thus coincident. @param Id - a unique Id for the given point that will be stored in an vtkIdList. @param[in] point - the point coordinate that we will store in the map to test if any other points are coincident with it.
- `vtkIdList = obj.GetCoincidentPointIds (double point[3])` - Retrieve the list of point Ids that are coincident with the given point. @param[in] point - the coordinate of coincident points we want to retrieve.
- `vtkIdList = obj.GetNextCoincidentPointIds ()` - Used to iterate the sets of coincident points within the map. InitTraversal must be called first or NULL will always be returned.
- `obj.InitTraversal ()`
- `obj.RemoveNonCoincidentPoints ()`
- `obj.Clear ()`

33.31 vtkCompositeDataGeometryFilter

33.31.1 Usage

vtkCompositeDataGeometryFilter applies vtkGeometryFilter to all leaves in vtkCompositeDataSet. Place this filter at the end of a pipeline before a polydata consumer such as a polydata mapper to extract geometry from all blocks and append them to one polydata object.

To create an instance of class vtkCompositeDataGeometryFilter, simply invoke its constructor as follows

```
obj = vtkCompositeDataGeometryFilter
```

33.31.2 Methods

The class vtkCompositeDataGeometryFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositeDataGeometryFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositeDataGeometryFilter = obj.NewInstance ()`
- `vtkCompositeDataGeometryFilter = obj.SafeDownCast (vtkObject o)`

33.32 vtkCompositeDataProbeFilter

33.32.1 Usage

vtkCompositeDataProbeFilter supports probing into multi-group datasets. It sequentially probes through each concrete dataset within the composite probing at only those locations at which there were no hits when probing earlier datasets. For Hierarchical datasets, this traversal through leaf datasets is done in reverse order of levels i.e. highest level first.

When dealing with composite datasets, partial arrays are common i.e. data-arrays that are not available in all of the blocks. By default, this filter only passes those point and cell data-arrays that are available

in all the blocks i.e. partial array are removed. When PassPartialArrays is turned on, this behavior is changed to take a union of all arrays present thus partial arrays are passed as well. However, for composite dataset input, this filter still produces a non-composite output. For all those locations in a block of where a particular data array is missing, this filter uses vtkMath::Nan() for double and float arrays, while 0 for all other types of arrays i.e int, char etc.

To create an instance of class vtkCompositeDataProbeFilter, simply invoke its constructor as follows

```
obj = vtkCompositeDataProbeFilter
```

33.32.2 Methods

The class vtkCompositeDataProbeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositeDataProbeFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositeDataProbeFilter = obj.NewInstance ()`
- `vtkCompositeDataProbeFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetPassPartialArrays (bool )` - When dealing with composite datasets, partial arrays are common i.e. data-arrays that are not available in all of the blocks. By default, this filter only passes those point and cell data-arrays that are available in all the blocks i.e. partial array are removed. When PassPartialArrays is turned on, this behavior is changed to take a union of all arrays present thus partial arrays are passed as well. However, for composite dataset input, this filter still produces a non-composite output. For all those locations in a block of where a particular data array is missing, this filter uses vtkMath::Nan() for double and float arrays, while 0 for all other types of arrays i.e int, char etc.
- `bool = obj.GetPassPartialArrays ()` - When dealing with composite datasets, partial arrays are common i.e. data-arrays that are not available in all of the blocks. By default, this filter only passes those point and cell data-arrays that are available in all the blocks i.e. partial array are removed. When PassPartialArrays is turned on, this behavior is changed to take a union of all arrays present thus partial arrays are passed as well. However, for composite dataset input, this filter still produces a non-composite output. For all those locations in a block of where a particular data array is missing, this filter uses vtkMath::Nan() for double and float arrays, while 0 for all other types of arrays i.e int, char etc.
- `obj.PassPartialArraysOn ()` - When dealing with composite datasets, partial arrays are common i.e. data-arrays that are not available in all of the blocks. By default, this filter only passes those point and cell data-arrays that are available in all the blocks i.e. partial array are removed. When PassPartialArrays is turned on, this behavior is changed to take a union of all arrays present thus partial arrays are passed as well. However, for composite dataset input, this filter still produces a non-composite output. For all those locations in a block of where a particular data array is missing, this filter uses vtkMath::Nan() for double and float arrays, while 0 for all other types of arrays i.e int, char etc.
- `obj.PassPartialArraysOff ()` - When dealing with composite datasets, partial arrays are common i.e. data-arrays that are not available in all of the blocks. By default, this filter only passes those point and cell data-arrays that are available in all the blocks i.e. partial array are removed. When PassPartialArrays is turned on, this behavior is changed to take a union of all arrays present thus partial arrays are passed as well. However, for composite dataset input, this filter still produces a non-composite output. For all those locations in a block of where a particular data array is missing, this filter uses vtkMath::Nan() for double and float arrays, while 0 for all other types of arrays i.e int, char etc.

33.33 vtkConeSource

33.33.1 Usage

vtkConeSource creates a cone centered at a specified point and pointing in a specified direction. (By default, the center is the origin and the direction is the x-axis.) Depending upon the resolution of this object, different representations are created. If resolution=0 a line is created; if resolution=1, a single triangle is created; if resolution=2, two crossed triangles are created. For resolution ¿ 2, a 3D cone (with resolution number of sides) is created. It also is possible to control whether the bottom of the cone is capped with a (resolution-sided) polygon, and to specify the height and radius of the cone.

To create an instance of class vtkConeSource, simply invoke its constructor as follows

```
obj = vtkConeSource
```

33.33.2 Methods

The class vtkConeSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkConeSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkConeSource = obj.NewInstance ()`
- `vtkConeSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetHeight (double )` - Set the height of the cone. This is the height along the cone in its specified direction.
- `double = obj.GetHeightMinValue ()` - Set the height of the cone. This is the height along the cone in its specified direction.
- `double = obj.GetHeightMaxValue ()` - Set the height of the cone. This is the height along the cone in its specified direction.
- `double = obj.GetHeight ()` - Set the height of the cone. This is the height along the cone in its specified direction.
- `obj.SetRadius (double )` - Set the base radius of the cone.
- `double = obj.GetRadiusMinValue ()` - Set the base radius of the cone.
- `double = obj.GetRadiusMaxValue ()` - Set the base radius of the cone.
- `double = obj.GetRadius ()` - Set the base radius of the cone.
- `obj.SetResolution (int )` - Set the number of facets used to represent the cone.
- `int = obj.GetResolutionMinValue ()` - Set the number of facets used to represent the cone.
- `int = obj.GetResolutionMaxValue ()` - Set the number of facets used to represent the cone.
- `int = obj.GetResolution ()` - Set the number of facets used to represent the cone.
- `obj.SetCenter (double , double , double )` - Set the center of the cone. It is located at the middle of the axis of the cone. Warning: this is not the center of the base of the cone! The default is 0,0,0.

- `obj.SetCenter (double a[3])` - Set the center of the cone. It is located at the middle of the axis of the cone. Warning: this is not the center of the base of the cone! The default is 0,0,0.
- `double = obj. GetCenter ()` - Set the center of the cone. It is located at the middle of the axis of the cone. Warning: this is not the center of the base of the cone! The default is 0,0,0.
- `obj.SetDirection (double , double , double )` - Set the orientation vector of the cone. The vector does not have to be normalized. The direction goes from the center of the base toward the apex. The default is (1,0,0).
- `obj.SetDirection (double a[3])` - Set the orientation vector of the cone. The vector does not have to be normalized. The direction goes from the center of the base toward the apex. The default is (1,0,0).
- `double = obj. GetDirection ()` - Set the orientation vector of the cone. The vector does not have to be normalized. The direction goes from the center of the base toward the apex. The default is (1,0,0).
- `obj.SetAngle (double angle)` - Set the angle of the cone. This is the angle between the axis of the cone and a generatrix. Warning: this is not the aperture! The aperture is twice this angle. As a side effect, the angle plus height sets the base radius of the cone. Angle is expressed in degrees.
- `double = obj.GetAngle ()` - Set the angle of the cone. This is the angle between the axis of the cone and a generatrix. Warning: this is not the aperture! The aperture is twice this angle. As a side effect, the angle plus height sets the base radius of the cone. Angle is expressed in degrees.
- `obj.SetCapping (int )` - Turn on/off whether to cap the base of the cone with a polygon.
- `int = obj.GetCapping ()` - Turn on/off whether to cap the base of the cone with a polygon.
- `obj.CappingOn ()` - Turn on/off whether to cap the base of the cone with a polygon.
- `obj.CappingOff ()` - Turn on/off whether to cap the base of the cone with a polygon.

33.34 vtkConnectivityFilter

33.34.1 Usage

vtkConnectivityFilter is a filter that extracts cells that share common points and/or meet other connectivity criterion. (Cells that share vertices and meet other connectivity criterion such as scalar range are known as a region.) The filter works in one of six ways: 1) extract the largest connected region in the dataset; 2) extract specified region numbers; 3) extract all regions sharing specified point ids; 4) extract all regions sharing specified cell ids; 5) extract the region closest to the specified point; or 6) extract all regions (used to color the data by region).

vtkConnectivityFilter is generalized to handle any type of input dataset. It generates output data of type vtkUnstructuredGrid. If you know that your input type is vtkPolyData, you may wish to use vtkPolyDataConnectivityFilter.

The behavior of vtkConnectivityFilter can be modified by turning on the boolean ivar ScalarConnectivity. If this flag is on, the connectivity algorithm is modified so that cells are considered connected only if 1) they are geometrically connected (share a point) and 2) the scalar values of one of the cell's points falls in the scalar range specified. This use of ScalarConnectivity is particularly useful for volume datasets: it can be used as a simple "connected segmentation" algorithm. For example, by using a seed voxel (i.e., cell) on a known anatomical structure, connectivity will pull out all voxels "containing" the anatomical structure. These voxels can then be contoured or processed by other visualization filters.

To create an instance of class vtkConnectivityFilter, simply invoke its constructor as follows

```
obj = vtkConnectivityFilter
```

33.34.2 Methods

The class vtkConnectivityFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkConnectivityFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkConnectivityFilter = obj.NewInstance ()`
- `vtkConnectivityFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetScalarConnectivity (int )` - Turn on/off connectivity based on scalar value. If on, cells are connected only if they share points AND one of the cells scalar values falls in the scalar range specified.
- `int = obj.GetScalarConnectivity ()` - Turn on/off connectivity based on scalar value. If on, cells are connected only if they share points AND one of the cells scalar values falls in the scalar range specified.
- `obj.ScalarConnectivityOn ()` - Turn on/off connectivity based on scalar value. If on, cells are connected only if they share points AND one of the cells scalar values falls in the scalar range specified.
- `obj.ScalarConnectivityOff ()` - Turn on/off connectivity based on scalar value. If on, cells are connected only if they share points AND one of the cells scalar values falls in the scalar range specified.
- `obj.SetScalarRange (double , double )` - Set the scalar range to use to extract cells based on scalar connectivity.
- `obj.SetScalarRange (double  a[2])` - Set the scalar range to use to extract cells based on scalar connectivity.
- `double = obj. GetScalarRange ()` - Set the scalar range to use to extract cells based on scalar connectivity.
- `obj.SetExtractionMode (int )` - Control the extraction of connected surfaces.
- `int = obj.GetExtractionModeMinValue ()` - Control the extraction of connected surfaces.
- `int = obj.GetExtractionModeMaxValue ()` - Control the extraction of connected surfaces.
- `int = obj.GetExtractionMode ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToPointSeededRegions ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToCellSeededRegions ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToLargestRegion ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToSpecifiedRegions ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToClosestPointRegion ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToAllRegions ()` - Control the extraction of connected surfaces.
- `string = obj.GetExtractionModeAsString ()` - Control the extraction of connected surfaces.
- `obj.InitializeSeedList ()` - Initialize list of point ids/cell ids used to seed regions.
- `obj.AddSeed (vtkIdType id)` - Add a seed id (point or cell id). Note: ids are 0-offset.

- `obj.DeleteSeed (vtkIdType id)` - Delete a seed id (point or cell id). Note: ids are 0-offset.
- `obj.InitializeSpecifiedRegionList ()` - Initialize list of region ids to extract.
- `obj.AddSpecifiedRegion (int id)` - Add a region id to extract. Note: ids are 0-offset.
- `obj.DeleteSpecifiedRegion (int id)` - Delete a region id to extract. Note: ids are 0-offset.
- `obj.SetClosestPoint (double , double , double )` - Use to specify x-y-z point coordinates when extracting the region closest to a specified point.
- `obj.SetClosestPoint (double a[3])` - Use to specify x-y-z point coordinates when extracting the region closest to a specified point.
- `double = obj. GetClosestPoint ()` - Use to specify x-y-z point coordinates when extracting the region closest to a specified point.
- `int = obj.GetNumberOfExtractedRegions ()` - Obtain the number of connected regions.
- `obj.SetColorRegions (int )` - Turn on/off the coloring of connected regions.
- `int = obj.GetColorRegions ()` - Turn on/off the coloring of connected regions.
- `obj.ColorRegionsOn ()` - Turn on/off the coloring of connected regions.
- `obj.ColorRegionsOff ()` - Turn on/off the coloring of connected regions.

33.35 vtkContourFilter

33.35.1 Usage

vtkContourFilter is a filter that takes as input any dataset and generates on output isosurfaces and/or isolines. The exact form of the output depends upon the dimensionality of the input data. Data consisting of 3D cells will generate isosurfaces, data consisting of 2D cells will generate isolines, and data with 1D or 0D cells will generate isopoints. Combinations of output type are possible if the input dimension is mixed.

To use this filter you must specify one or more contour values. You can either use the method SetValue() to specify each contour value, or use GenerateValues() to generate a series of evenly spaced contours. It is also possible to accelerate the operation of this filter (at the cost of extra memory) by using a vtkScalarTree. A scalar tree is used to quickly locate cells that contain a contour surface. This is especially effective if multiple contours are being extracted. If you want to use a scalar tree, invoke the method UseScalarTreeOn().

To create an instance of class vtkContourFilter, simply invoke its constructor as follows

```
obj = vtkContourFilter
```

33.35.2 Methods

The class vtkContourFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkContourFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkContourFilter = obj.NewInstance ()`
- `vtkContourFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)` - Methods to set / get contour values.

- `double = obj.GetValue (int i)` - Methods to set / get contour values.
- `obj.GetValues (double contourValues)` - Methods to set / get contour values.
- `obj.SetNumberOfContours (int number)` - Methods to set / get contour values.
- `int = obj.GetNumberOfContours ()` - Methods to set / get contour values.
- `obj.GenerateValues (int numContours, double range[2])` - Methods to set / get contour values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Methods to set / get contour values.
- `long = obj.GetMTime ()` - Modified GetMTime Because we delegate to vtkContourValues
- `obj.SetComputeNormals (int )` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeNormals ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOn ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOff ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeGradients (int )` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeGradients ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOn ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOff ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeScalars (int )` - Set/Get the computation of scalars.
- `int = obj.GetComputeScalars ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOn ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOff ()` - Set/Get the computation of scalars.
- `obj.SetUseScalarTree (int )` - Enable the use of a scalar tree to accelerate contour extraction.

- `int = obj.GetUseScalarTree ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.UseScalarTreeOn ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.UseScalarTreeOff ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.SetScalarTree (vtkScalarTree )` - Enable the use of a scalar tree to accelerate contour extraction.
- `vtkScalarTree = obj.GetScalarTree ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.
- `obj.SetArrayComponent (int )` - Set/get which component of the scalar array to contour on; defaults to 0. Currently this feature only works if the input is a vtkImageData.
- `int = obj.GetArrayComponent ()` - Set/get which component of the scalar array to contour on; defaults to 0. Currently this feature only works if the input is a vtkImageData.

33.36 vtkContourGrid

33.36.1 Usage

vtkContourGrid is a filter that takes as input datasets of type vtkUnstructuredGrid and generates on output isosurfaces and/or isolines. The exact form of the output depends upon the dimensionality of the input data. Data consisting of 3D cells will generate isosurfaces, data consisting of 2D cells will generate isolines, and data with 1D or 0D cells will generate isopoints. Combinations of output type are possible if the input dimension is mixed.

To use this filter you must specify one or more contour values. You can either use the method SetValue() to specify each contour value, or use GenerateValues() to generate a series of evenly spaced contours. It is also possible to accelerate the operation of this filter (at the cost of extra memory) by using a vtkScalarTree. A scalar tree is used to quickly locate cells that contain a contour surface. This is especially effective if multiple contours are being extracted. If you want to use a scalar tree, invoke the method UseScalarTreeOn().

To create an instance of class vtkContourGrid, simply invoke its constructor as follows

```
obj = vtkContourGrid
```

33.36.2 Methods

The class vtkContourGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkContourGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkContourGrid = obj.NewInstance ()`
- `vtkContourGrid = obj.SafeDownCast (vtkObject o)`

- `obj.SetValue (int i, double value)` - Methods to set / get contour values.
- `double = obj.GetValue (int i)` - Methods to set / get contour values.
- `obj.GetValues (double contourValues)` - Methods to set / get contour values.
- `obj.SetNumberOfContours (int number)` - Methods to set / get contour values.
- `int = obj.GetNumberOfContours ()` - Methods to set / get contour values.
- `obj.GenerateValues (int numContours, double range[2])` - Methods to set / get contour values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Methods to set / get contour values.
- `long = obj.GetMTime ()` - Modified GetMTime Because we delegate to vtkContourValues
- `obj.SetComputeNormals (int )` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeNormals ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOn ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOff ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeGradients (int )` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeGradients ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOn ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOff ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeScalars (int )` - Set/Get the computation of scalars.
- `int = obj.GetComputeScalars ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOn ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOff ()` - Set/Get the computation of scalars.

- `obj.SetUseScalarTree (int )` - Enable the use of a scalar tree to accelerate contour extraction.
- `int = obj.GetUseScalarTree ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.UseScalarTreeOn ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.UseScalarTreeOff ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.

33.37 vtkConvertSelection

33.37.1 Usage

vtkConvertSelection converts an input selection from one type to another in the context of a data object being selected. The first input is the selection, while the second input is the data object that the selection relates to.

To create an instance of class vtkConvertSelection, simply invoke its constructor as follows

```
obj = vtkConvertSelection
```

33.37.2 Methods

The class vtkConvertSelection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkConvertSelection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkConvertSelection = obj.NewInstance ()`
- `vtkConvertSelection = obj.SafeDownCast (vtkObject o)`
- `obj.SetDataObjectConnection (vtkAlgorithmOutput in)` - A convenience method for setting the second input (i.e. the data object).
- `obj.SetInputFieldType (int )` - The input field type. If this is set to a number other than -1, ignores the input selection field type and instead assumes that all selection nodes have the field type specified. This should be one of the constants defined in vtkSelectionNode.h. Default is -1.
- `int = obj.GetInputFieldType ()` - The input field type. If this is set to a number other than -1, ignores the input selection field type and instead assumes that all selection nodes have the field type specified. This should be one of the constants defined in vtkSelectionNode.h. Default is -1.
- `obj.SetOutputType (int )` - The output selection content type. This should be one of the constants defined in vtkSelectionNode.h.
- `int = obj.GetOutputType ()` - The output selection content type. This should be one of the constants defined in vtkSelectionNode.h.

- `obj.SetArrayName (string )` - The output array name for value or threshold selections.
- `string = obj.GetArrayName ()` - The output array name for value or threshold selections.
- `obj.SetArrayNames (vtkStringArray )` - The output array names for value selection.
- `vtkStringArray = obj.GetArrayNames ()` - The output array names for value selection.
- `obj.AddArrayName (string )` - Convenience methods used by UI
- `obj.ClearArrayNames ()` - Convenience methods used by UI
- `obj.SetMatchAnyValues (bool )` - When on, creates a separate selection node for each array. Defaults to OFF.
- `bool = obj.GetMatchAnyValues ()` - When on, creates a separate selection node for each array. Defaults to OFF.
- `obj.MatchAnyValuesOn ()` - When on, creates a separate selection node for each array. Defaults to OFF.
- `obj.MatchAnyValuesOff ()` - When on, creates a separate selection node for each array. Defaults to OFF.

33.38 vtkCubeSource

33.38.1 Usage

vtkCubeSource creates a cube centered at origin. The cube is represented with four-sided polygons. It is possible to specify the length, width, and height of the cube independently.

To create an instance of class vtkCubeSource, simply invoke its constructor as follows

```
obj = vtkCubeSource
```

33.38.2 Methods

The class vtkCubeSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCubeSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCubeSource = obj.NewInstance ()`
- `vtkCubeSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetXLength (double )` - Set the length of the cube in the x-direction.
- `double = obj.GetXLengthMinValue ()` - Set the length of the cube in the x-direction.
- `double = obj.GetXLengthMaxValue ()` - Set the length of the cube in the x-direction.
- `double = obj.GetXLength ()` - Set the length of the cube in the x-direction.
- `obj.SetYLength (double )` - Set the length of the cube in the y-direction.
- `double = obj.GetYLengthMinValue ()` - Set the length of the cube in the y-direction.

- `double = obj.GetYLengthMaxValue ()` - Set the length of the cube in the y-direction.
- `double = obj.GetYLength ()` - Set the length of the cube in the y-direction.
- `obj.SetZLength (double )` - Set the length of the cube in the z-direction.
- `double = obj.GetZLengthMinValue ()` - Set the length of the cube in the z-direction.
- `double = obj.GetZLengthMaxValue ()` - Set the length of the cube in the z-direction.
- `double = obj.GetZLength ()` - Set the length of the cube in the z-direction.
- `obj.SetCenter (double , double , double )` - Set the center of the cube.
- `obj.SetCenter (double  a[3])` - Set the center of the cube.
- `double = obj. GetCenter ()` - Set the center of the cube.
- `obj.SetBounds (double xMin, double xMax, double yMin, double yMax, double zMin, double zMax)` - Convenience method allows creation of cube by specifying bounding box.
- `obj.SetBounds (double bounds[6])` - Convenience method allows creation of cube by specifying bounding box.

33.39 vtkCursor2D

33.39.1 Usage

vtkCursor2D is a class that generates a 2D cursor representation. The cursor consists of two intersection axes lines that meet at the cursor focus. Several optional features are available as well. An optional 2D bounding box may be enabled. An inner radius, centered at the focal point, can be set that erases the intersecting lines (e.g., it leaves a clear area under the focal point so you can see what you are selecting). And finally, an optional point can be enabled located at the focal point. All of these features can be turned on and off independently.

To create an instance of class vtkCursor2D, simply invoke its constructor as follows

```
obj = vtkCursor2D
```

33.39.2 Methods

The class vtkCursor2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCursor2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCursor2D = obj.NewInstance ()`
- `vtkCursor2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetModelBounds (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Set / get the bounding box of the 2D cursor. This defines the outline of the cursor, and where the focal point should lie.
- `obj.SetModelBounds (double bounds[6])` - Set / get the bounding box of the 2D cursor. This defines the outline of the cursor, and where the focal point should lie.

- `double = obj. GetModelBounds ()` - Set / get the bounding box of the 2D cursor. This defines the outline of the cursor, and where the focal point should lie.
- `obj.SetFocalPoint (double x[3])` - Set/Get the position of cursor focus. If translation mode is on, then the entire cursor (including bounding box, cursor, and shadows) is translated. Otherwise, the focal point will either be clamped to the bounding box, or wrapped, if Wrap is on. (Note: this behavior requires that the bounding box is set prior to the focal point.) Note that the method takes a 3D point but ignores the z-coordinate value.
- `obj.SetFocalPoint (double x, double y, double z)` - Set/Get the position of cursor focus. If translation mode is on, then the entire cursor (including bounding box, cursor, and shadows) is translated. Otherwise, the focal point will either be clamped to the bounding box, or wrapped, if Wrap is on. (Note: this behavior requires that the bounding box is set prior to the focal point.) Note that the method takes a 3D point but ignores the z-coordinate value.
- `double = obj. GetFocalPoint ()` - Set/Get the position of cursor focus. If translation mode is on, then the entire cursor (including bounding box, cursor, and shadows) is translated. Otherwise, the focal point will either be clamped to the bounding box, or wrapped, if Wrap is on. (Note: this behavior requires that the bounding box is set prior to the focal point.) Note that the method takes a 3D point but ignores the z-coordinate value.
- `obj.SetOutline (int )` - Turn on/off the wireframe bounding box.
- `int = obj.GetOutline ()` - Turn on/off the wireframe bounding box.
- `obj.OutlineOn ()` - Turn on/off the wireframe bounding box.
- `obj.OutlineOff ()` - Turn on/off the wireframe bounding box.
- `obj.SetAxes (int )` - Turn on/off the wireframe axes.
- `int = obj.GetAxes ()` - Turn on/off the wireframe axes.
- `obj.AxesOn ()` - Turn on/off the wireframe axes.
- `obj.AxesOff ()` - Turn on/off the wireframe axes.
- `obj.SetRadius (double )` - Specify a radius for a circle. This erases the cursor lines around the focal point.
- `double = obj.GetRadiusMinValue ()` - Specify a radius for a circle. This erases the cursor lines around the focal point.
- `double = obj.GetRadiusMaxValue ()` - Specify a radius for a circle. This erases the cursor lines around the focal point.
- `double = obj.GetRadius ()` - Specify a radius for a circle. This erases the cursor lines around the focal point.
- `obj.SetPoint (int )` - Turn on/off the point located at the cursor focus.
- `int = obj.GetPoint ()` - Turn on/off the point located at the cursor focus.
- `obj.PointOn ()` - Turn on/off the point located at the cursor focus.
- `obj.PointOff ()` - Turn on/off the point located at the cursor focus.
- `obj.SetTranslationMode (int )` - Enable/disable the translation mode. If on, changes in cursor position cause the entire widget to translate along with the cursor. By default, translation mode is off.
- `int = obj.GetTranslationMode ()` - Enable/disable the translation mode. If on, changes in cursor position cause the entire widget to translate along with the cursor. By default, translation mode is off.

- `obj.TranslationModeOn ()` - Enable/disable the translation mode. If on, changes in cursor position cause the entire widget to translate along with the cursor. By default, translation mode is off.
- `obj.TranslationModeOff ()` - Enable/disable the translation mode. If on, changes in cursor position cause the entire widget to translate along with the cursor. By default, translation mode is off.
- `obj.SetWrap (int )` - Turn on/off cursor wrapping. If the cursor focus moves outside the specified bounds, the cursor will either be restrained against the nearest "wall" (Wrap=off), or it will wrap around (Wrap=on).
- `int = obj.GetWrap ()` - Turn on/off cursor wrapping. If the cursor focus moves outside the specified bounds, the cursor will either be restrained against the nearest "wall" (Wrap=off), or it will wrap around (Wrap=on).
- `obj.WrapOn ()` - Turn on/off cursor wrapping. If the cursor focus moves outside the specified bounds, the cursor will either be restrained against the nearest "wall" (Wrap=off), or it will wrap around (Wrap=on).
- `obj.WrapOff ()` - Turn on/off cursor wrapping. If the cursor focus moves outside the specified bounds, the cursor will either be restrained against the nearest "wall" (Wrap=off), or it will wrap around (Wrap=on).
- `obj.AllOn ()` - Turn every part of the cursor on or off.
- `obj.AllOff ()` - Turn every part of the cursor on or off.

33.40 vtkCursor3D

33.40.1 Usage

vtkCursor3D is an object that generates a 3D representation of a cursor. The cursor consists of a wireframe bounding box, three intersecting axes lines that meet at the cursor focus, and "shadows" or projections of the axes against the sides of the bounding box. Each of these components can be turned on/off.

This filter generates two output datasets. The first (Output) is just the geometric representation of the cursor. The second (Focus) is a single point at the focal point.

To create an instance of class vtkCursor3D, simply invoke its constructor as follows

```
obj = vtkCursor3D
```

33.40.2 Methods

The class vtkCursor3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCursor3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCursor3D = obj.NewInstance ()`
- `vtkCursor3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetModelBounds (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Set / get the boundary of the 3D cursor.
- `obj.SetModelBounds (double bounds[6])` - Set / get the boundary of the 3D cursor.

- `double = obj. GetModelBounds ()` - Set / get the boundary of the 3D cursor.
- `obj.SetFocalPoint (double x[3])` - Set/Get the position of cursor focus. If translation mode is on, then the entire cursor (including bounding box, cursor, and shadows) is translated. Otherwise, the focal point will either be clamped to the bounding box, or wrapped, if Wrap is on. (Note: this behavior requires that the bounding box is set prior to the focal point.)
- `obj.SetFocalPoint (double x, double y, double z)` - Set/Get the position of cursor focus. If translation mode is on, then the entire cursor (including bounding box, cursor, and shadows) is translated. Otherwise, the focal point will either be clamped to the bounding box, or wrapped, if Wrap is on. (Note: this behavior requires that the bounding box is set prior to the focal point.)
- `double = obj. GetFocalPoint ()` - Set/Get the position of cursor focus. If translation mode is on, then the entire cursor (including bounding box, cursor, and shadows) is translated. Otherwise, the focal point will either be clamped to the bounding box, or wrapped, if Wrap is on. (Note: this behavior requires that the bounding box is set prior to the focal point.)
- `obj.SetOutline (int )` - Turn on/off the wireframe bounding box.
- `int = obj.GetOutline ()` - Turn on/off the wireframe bounding box.
- `obj.OutlineOn ()` - Turn on/off the wireframe bounding box.
- `obj.OutlineOff ()` - Turn on/off the wireframe bounding box.
- `obj.SetAxes (int )` - Turn on/off the wireframe axes.
- `int = obj.GetAxes ()` - Turn on/off the wireframe axes.
- `obj.AxesOn ()` - Turn on/off the wireframe axes.
- `obj.AxesOff ()` - Turn on/off the wireframe axes.
- `obj.SetXShadows (int )` - Turn on/off the wireframe x-shadows.
- `int = obj.GetXShadows ()` - Turn on/off the wireframe x-shadows.
- `obj.XShadowsOn ()` - Turn on/off the wireframe x-shadows.
- `obj.XShadowsOff ()` - Turn on/off the wireframe x-shadows.
- `obj.SetYShadows (int )` - Turn on/off the wireframe y-shadows.
- `int = obj.GetYShadows ()` - Turn on/off the wireframe y-shadows.
- `obj.YShadowsOn ()` - Turn on/off the wireframe y-shadows.
- `obj.YShadowsOff ()` - Turn on/off the wireframe y-shadows.
- `obj.SetZShadows (int )` - Turn on/off the wireframe z-shadows.
- `int = obj.GetZShadows ()` - Turn on/off the wireframe z-shadows.
- `obj.ZShadowsOn ()` - Turn on/off the wireframe z-shadows.
- `obj.ZShadowsOff ()` - Turn on/off the wireframe z-shadows.
- `obj.SetTranslationMode (int )` - Enable/disable the translation mode. If on, changes in cursor position cause the entire widget to translate along with the cursor. By default, translation mode is off.
- `int = obj.GetTranslationMode ()` - Enable/disable the translation mode. If on, changes in cursor position cause the entire widget to translate along with the cursor. By default, translation mode is off.

- `obj.TranslationModeOn ()` - Enable/disable the translation mode. If on, changes in cursor position cause the entire widget to translate along with the cursor. By default, translation mode is off.
- `obj.TranslationModeOff ()` - Enable/disable the translation mode. If on, changes in cursor position cause the entire widget to translate along with the cursor. By default, translation mode is off.
- `obj.SetWrap (int )` - Turn on/off cursor wrapping. If the cursor focus moves outside the specified bounds, the cursor will either be restrained against the nearest "wall" (Wrap=off), or it will wrap around (Wrap=on).
- `int = obj.GetWrap ()` - Turn on/off cursor wrapping. If the cursor focus moves outside the specified bounds, the cursor will either be restrained against the nearest "wall" (Wrap=off), or it will wrap around (Wrap=on).
- `obj.WrapOn ()` - Turn on/off cursor wrapping. If the cursor focus moves outside the specified bounds, the cursor will either be restrained against the nearest "wall" (Wrap=off), or it will wrap around (Wrap=on).
- `obj.WrapOff ()` - Turn on/off cursor wrapping. If the cursor focus moves outside the specified bounds, the cursor will either be restrained against the nearest "wall" (Wrap=off), or it will wrap around (Wrap=on).
- `vtkPolyData = obj.GetFocus ()` - Get the focus for this filter.
- `obj.AllOn ()` - Turn every part of the 3D cursor on or off.
- `obj.AllOff ()` - Turn every part of the 3D cursor on or off.

33.41 vtkCurvatures

33.41.1 Usage

vtkCurvatures takes a polydata input and computes the curvature of the mesh at each point. Four possible methods of computation are available :

Gauss Curvature discrete Gauss curvature (K) computation, $K(vertexv) = 2*PI - \sum_{facetneighbsfofv}(angle_fatv)$ The contribution of every facet is for the moment weighted by $Area(facet)/3$ The units of Gaussian Curvature are $[1/m^2]$

Mean Curvature $H(vertexv) = averageoveredgesneighbseofH(e)\ H(edgee) = length(e)*dihedral_angle(e)$ NB: dihedral_angle is the ORIENTED angle between -PI and PI, this means that the surface is assumed to be orientable the computation creates the orientation The units of Mean Curvature are [1/m]

Maximum (k_max) and Minimum (k_min) Principal Curvatures $k_max = H + sqrt(H^2 - K)$ $k_min = H - sqrt(H^2 - K)$ Excepting spherical and planar surfaces which have equal principal curvatures, the curvature at a point on a surface varies with the direction one "sets off" from the point. For all directions, the curvature will pass through two extrema: a minimum (k_min) and a maximum (k_max) which occur at mutually orthogonal directions to each other.

NB. The sign of the Gauss curvature is a geometric ivariant, it should be +ve when the surface looks like a sphere, -ve when it looks like a saddle, however, the sign of the Mean curvature is not, it depends on the convention for normals - This code assumes that normals point outwards (ie from the surface of a sphere outwards). If a given mesh produces curvatures of opposite senses then the flag InvertMeanCurvature can be set and the Curvature reported by the Mean calculation will be inverted.

.SECTION Thanks Philip Batchelor philipp.batchelor@kcl.ac.uk for creating and contributing the class and Andrew Maclean a.maclean@acfr.usyd.edu.au for cleanups and fixes. Thanks also to Goodwin Lawlor for contributing patch to calculate principal curvatures

To create an instance of class vtkCurvatures, simply invoke its constructor as follows

```
obj = vtkCurvatures
```

33.41.2 Methods

The class vtkCurvatures has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCurvatures class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCurvatures = obj.NewInstance ()`
- `vtkCurvatures = obj.SafeDownCast (vtkObject o)`
- `obj.SetCurvatureType (int )` - Set/Get Curvature type VTK_CURVATURE_GAUSS: Gaussian curvature, stored as DataArray "Gauss_Curvature" VTK_CURVATURE_MEAN : Mean curvature, stored as DataArray "Mean_Curvature"
- `int = obj.GetCurvatureType ()` - Set/Get Curvature type VTK_CURVATURE_GAUSS: Gaussian curvature, stored as DataArray "Gauss_Curvature" VTK_CURVATURE_MEAN : Mean curvature, stored as DataArray "Mean_Curvature"
- `obj.SetCurvatureTypeToGaussian ()` - Set/Get Curvature type VTK_CURVATURE_GAUSS: Gaussian curvature, stored as DataArray "Gauss_Curvature" VTK_CURVATURE_MEAN : Mean curvature, stored as DataArray "Mean_Curvature"
- `obj.SetCurvatureTypeToMean ()` - Set/Get Curvature type VTK_CURVATURE_GAUSS: Gaussian curvature, stored as DataArray "Gauss_Curvature" VTK_CURVATURE_MEAN : Mean curvature, stored as DataArray "Mean_Curvature"
- `obj.SetCurvatureTypeToMaximum ()` - Set/Get Curvature type VTK_CURVATURE_GAUSS: Gaussian curvature, stored as DataArray "Gauss_Curvature" VTK_CURVATURE_MEAN : Mean curvature, stored as DataArray "Mean_Curvature"
- `obj.SetCurvatureTypeToMinimum ()` - Set/Get the flag which inverts the mean curvature calculation for meshes with inward pointing normals (default false)
- `obj.SetInvertMeanCurvature (int )` - Set/Get the flag which inverts the mean curvature calculation for meshes with inward pointing normals (default false)
- `int = obj.GetInvertMeanCurvature ()` - Set/Get the flag which inverts the mean curvature calculation for meshes with inward pointing normals (default false)
- `obj.InvertMeanCurvatureOn ()` - Set/Get the flag which inverts the mean curvature calculation for meshes with inward pointing normals (default false)
- `obj.InvertMeanCurvatureOff ()` - Set/Get the flag which inverts the mean curvature calculation for meshes with inward pointing normals (default false)

33.42 vtkCutter

33.42.1 Usage

vtkCutter is a filter to cut through data using any subclass of vtkImplicitFunction. That is, a polygonal surface is created corresponding to the implicit function F(x,y,z) = value(s), where you can specify one or more values used to cut with.

In VTK, cutting means reducing a cell of dimension N to a cut surface of dimension N-1. For example, a tetrahedron when cut by a plane (i.e., vtkPlane implicit function) will generate triangles. (In comparison, clipping takes a N dimensional cell and creates N dimension primitives.)

vtkCutter is generally used to "slice-through" a dataset, generating a surface that can be visualized. It is also possible to use vtkCutter to do a form of volume rendering. vtkCutter does this by generating multiple cut surfaces (usually planes) which are ordered (and rendered) from back-to-front. The surfaces are set translucent to give a volumetric rendering effect.

Note that data can be cut using either 1) the scalar values associated with the dataset or 2) an implicit function associated with this class. By default, if an implicit function is set it is used to clip the data set, otherwise the dataset scalars are used to perform the clipping.

To create an instance of class vtkCutter, simply invoke its constructor as follows

```
obj = vtkCutter
```

33.42.2 Methods

The class vtkCutter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCutter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCutter = obj.NewInstance ()`
- `vtkCutter = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)` - Get the ith contour value.
- `double = obj.GetValue (int i)` - Get a pointer to an array of contour values. There will be GetNumberOfContours() values in the list.
- `obj.GetValues (double contourValues)` - Set the number of contours to place into the list. You only really need to use this method to reduce list size. The method SetValue() will automatically increase list size as needed.
- `obj.SetNumberOfContours (int number)` - Get the number of contours in the list of contour values.
- `int = obj.GetNumberOfContours ()` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double range[2])` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Override GetMTime because we delegate to vtkContourValues and refer to vtkImplicitFunction.
- `long = obj.GetMTime ()` - Override GetMTime because we delegate to vtkContourValues and refer to vtkImplicitFunction.
- `obj.SetCutFunction (vtkImplicitFunction )`
- `vtkImplicitFunction = obj.GetCutFunction ()`
- `obj.SetGenerateCutScalars (int )` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data.
- `int = obj.GetGenerateCutScalars ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data.
- `obj.GenerateCutScalarsOn ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data.

- `obj.GenerateCutScalarsOff ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.SetSortBy (int )` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.
- `int = obj.GetSortByMinValue ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.
- `int = obj.GetSortByMaxValue ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.
- `int = obj.GetSortBy ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.
- `obj.SetSortByToSortByValue ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.
- `obj.SetSortByToSortByCell ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.

- `string = obj.GetSortByAsString ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.

- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.

33.43 vtkCylinderSource

33.43.1 Usage

vtkCylinderSource creates a polygonal cylinder centered at Center; The axis of the cylinder is aligned along the global y-axis. The height and radius of the cylinder can be specified, as well as the number of sides. It is also possible to control whether the cylinder is open-ended or capped.

To create an instance of class vtkCylinderSource, simply invoke its constructor as follows

```
obj = vtkCylinderSource
```

33.43.2 Methods

The class vtkCylinderSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCylinderSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCylinderSource = obj.NewInstance ()`
- `vtkCylinderSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetHeight (double )` - Set the height of the cylinder. Initial value is 1.
- `double = obj.GetHeightMinValue ()` - Set the height of the cylinder. Initial value is 1.
- `double = obj.GetHeightMaxValue ()` - Set the height of the cylinder. Initial value is 1.
- `double = obj.GetHeight ()` - Set the height of the cylinder. Initial value is 1.
- `obj.SetRadius (double )` - Set the radius of the cylinder. Initial value is 0.5
- `double = obj.GetRadiusMinValue ()` - Set the radius of the cylinder. Initial value is 0.5
- `double = obj.GetRadiusMaxValue ()` - Set the radius of the cylinder. Initial value is 0.5
- `double = obj.GetRadius ()` - Set the radius of the cylinder. Initial value is 0.5
- `obj.SetCenter (double , double , double )` - Set/Get cylinder center. Initial value is (0.0,0.0,0.0)
- `obj.SetCenter (double  a[3])` - Set/Get cylinder center. Initial value is (0.0,0.0,0.0)
- `double = obj. GetCenter ()` - Set/Get cylinder center. Initial value is (0.0,0.0,0.0)
- `obj.SetResolution (int )` - Set the number of facets used to define cylinder. Initial value is 6.

- `int = obj.GetResolutionMinValue ()` - Set the number of facets used to define cylinder. Initial value is 6.
- `int = obj.GetResolutionMaxValue ()` - Set the number of facets used to define cylinder. Initial value is 6.
- `int = obj.GetResolution ()` - Set the number of facets used to define cylinder. Initial value is 6.
- `obj.SetCapping (int )` - Turn on/off whether to cap cylinder with polygons. Initial value is true.
- `int = obj.GetCapping ()` - Turn on/off whether to cap cylinder with polygons. Initial value is true.
- `obj.CappingOn ()` - Turn on/off whether to cap cylinder with polygons. Initial value is true.
- `obj.CappingOff ()` - Turn on/off whether to cap cylinder with polygons. Initial value is true.

33.44 vtkDashedStreamLine

33.44.1 Usage

vtkDashedStreamLine is a filter that generates a "dashed" streamline for an arbitrary dataset. The streamline consists of a series of dashes, each of which represents (approximately) a constant time increment. Thus, in the resulting visual representation, relatively long dashes represent areas of high velocity, and small dashes represent areas of low velocity.

vtkDashedStreamLine introduces the instance variable DashFactor. DashFactor interacts with its superclass' instance variable StepLength to create the dashes. DashFactor is the percentage of the StepLength line segment that is visible. Thus, if the DashFactor=0.75, the dashes will be "three-quarters on" and "one-quarter off".

To create an instance of class vtkDashedStreamLine, simply invoke its constructor as follows

```
obj = vtkDashedStreamLine
```

33.44.2 Methods

The class vtkDashedStreamLine has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDashedStreamLine class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDashedStreamLine = obj.NewInstance ()`
- `vtkDashedStreamLine = obj.SafeDownCast (vtkObject o)`
- `obj.SetDashFactor (double )` - For each dash, specify the fraction of the dash that is "on". A factor of 1.0 will result in a continuous line, a factor of 0.5 will result in dashed that are half on and half off.
- `double = obj.GetDashFactorMinValue ()` - For each dash, specify the fraction of the dash that is "on". A factor of 1.0 will result in a continuous line, a factor of 0.5 will result in dashed that are half on and half off.
- `double = obj.GetDashFactorMaxValue ()` - For each dash, specify the fraction of the dash that is "on". A factor of 1.0 will result in a continuous line, a factor of 0.5 will result in dashed that are half on and half off.
- `double = obj.GetDashFactor ()` - For each dash, specify the fraction of the dash that is "on". A factor of 1.0 will result in a continuous line, a factor of 0.5 will result in dashed that are half on and half off.

33.45 vtkDataObjectGenerator

33.45.1 Usage

vtkDataObjectGenerator parses a string and produces dataobjects from the dataobject template names it sees in the string. For example, if the string contains "ID1" the generator will create a vtkImageData. "UF1", "RG1", "SG1", "PD1", and "UG1" will produce vtkUniformGrid, vtkRectilinearGrid, vtkStructuredGrid, vtkPolyData and vtkUnstructuredGrid respectively. "PD2" will produce an alternate vtkPolyData. You can compose composite datasets from the atomic ones listed above by placing them within one of the two composite dataset identifiers - "MB" or "HB[]". "MB ID1 PD1 MB " for example will create a vtkMultiBlockDataSet consisting of three blocks: image data, poly data, multi-block (empty). Hierarchical Box data sets additionally require the notion of groups, declared within "()" braces, to specify AMR depth. "HB[(UF1)(UF1)(UF1)]" will create a vtkHierarchicalBoxDataSet representing an octree that is three levels deep, in which the firstmost cell in each level is refined.

To create an instance of class vtkDataObjectGenerator, simply invoke its constructor as follows

```
obj = vtkDataObjectGenerator
```

33.45.2 Methods

The class vtkDataObjectGenerator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObjectGenerator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataObjectGenerator = obj.NewInstance ()`
- `vtkDataObjectGenerator = obj.SafeDownCast (vtkObject o)`
- `obj.SetProgram (string )` - The string that will be parsed to specify a dataobject structure.
- `string = obj.GetProgram ()` - The string that will be parsed to specify a dataobject structure.

33.46 vtkDataObjectToDataSetFilter

33.46.1 Usage

vtkDataObjectToDataSetFilter is an class that maps a data object (i.e., a field) into a concrete dataset, i.e., gives structure to the field by defining a geometry and topology.

To use this filter you associate components in the input field data with portions of the output dataset. (A component is an array of values from the field.) For example, you would specify x-y-z points by assigning components from the field for the x, then y, then z values of the points. You may also have to specify component ranges (for each z-y-z) to make sure that the number of x, y, and z values is the same. Also, you may want to normalize the components which helps distribute the data uniformly. Once you've setup the filter to combine all the pieces of data into a specified dataset (the geometry, topology, point and cell data attributes), the various output methods (e.g., GetPolyData()) are used to retrieve the final product.

This filter is often used in conjunction with vtkFieldDataToAttributeDataFilter. vtkFieldDataToAttributeDataFilter takes field data and transforms it into attribute data (e.g., point and cell data attributes such as scalars and vectors). To do this, use this filter which constructs a concrete dataset and passes the input data object field data to its output. and then use vtkFieldDataToAttributeDataFilter to generate the attribute data associated with the dataset.

To create an instance of class vtkDataObjectToDataSetFilter, simply invoke its constructor as follows

```
obj = vtkDataObjectToDataSetFilter
```

33.46.2 Methods

The class vtkDataObjectToDataSetFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObjectToDataSetFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataObjectToDataSetFilter = obj.NewInstance ()`
- `vtkDataObjectToDataSetFilter = obj.SafeDownCast (vtkObject o)`
- `vtkDataObject = obj.GetInput ()` - Get the input to the filter.
- `obj.SetDataSetType (int )` - Control what type of data is generated for output.
- `int = obj.GetDataSetType ()` - Control what type of data is generated for output.
- `obj.SetDataSetTypeToPolyData ()` - Control what type of data is generated for output.
- `obj.SetDataSetTypeToStructuredPoints ()` - Control what type of data is generated for output.
- `obj.SetDataSetTypeToStructuredGrid ()` - Control what type of data is generated for output.
- `obj.SetDataSetTypeToRectilinearGrid ()` - Control what type of data is generated for output.
- `obj.SetDataSetTypeToUnstructuredGrid ()` - Control what type of data is generated for output.
- `vtkDataSet = obj.GetOutput ()` - Get the output in different forms. The particular method invoked should be consistent with the SetDataSetType() method. (Note: GetOutput() will always return a type consistent with SetDataSetType(). Also, GetOutput() will return NULL if the filter aborted due to inconsistent data.)
- `vtkDataSet = obj.GetOutput (int idx)` - Get the output in different forms. The particular method invoked should be consistent with the SetDataSetType() method. (Note: GetOutput() will always return a type consistent with SetDataSetType(). Also, GetOutput() will return NULL if the filter aborted due to inconsistent data.)
- `vtkPolyData = obj.GetPolyDataOutput ()` - Get the output in different forms. The particular method invoked should be consistent with the SetDataSetType() method. (Note: GetOutput() will always return a type consistent with SetDataSetType(). Also, GetOutput() will return NULL if the filter aborted due to inconsistent data.)
- `vtkStructuredPoints = obj.GetStructuredPointsOutput ()` - Get the output in different forms. The particular method invoked should be consistent with the SetDataSetType() method. (Note: GetOutput() will always return a type consistent with SetDataSetType(). Also, GetOutput() will return NULL if the filter aborted due to inconsistent data.)
- `vtkStructuredGrid = obj.GetStructuredGridOutput ()` - Get the output in different forms. The particular method invoked should be consistent with the SetDataSetType() method. (Note: GetOutput() will always return a type consistent with SetDataSetType(). Also, GetOutput() will return NULL if the filter aborted due to inconsistent data.)
- `vtkUnstructuredGrid = obj.GetUnstructuredGridOutput ()` - Get the output in different forms. The particular method invoked should be consistent with the SetDataSetType() method. (Note: GetOutput() will always return a type consistent with SetDataSetType(). Also, GetOutput() will return NULL if the filter aborted due to inconsistent data.)

- `vtkRectilinearGrid = obj.GetRectilinearGridOutput ()` - Get the output in different forms. The particular method invoked should be consistent with the SetDataSetType() method. (Note: GetOutput() will always return a type consistent with SetDataSetType(). Also, GetOutput() will return NULL if the filter aborted due to inconsistent data.)

- `obj.SetPointComponent (int comp, string arrayName, int arrayComp, int min, int max, int normalize)` - Define the component of the field to be used for the x, y, and z values of the points. Note that the parameter comp must lie between (0,2) and refers to the x-y-z (i.e., 0,1,2) components of the points. To define the field component to use you can specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract. (This method should be used for vtkPolyData, vtkUnstructuredGrid, vtkStructuredGrid, and vtkRectilinearGrid.) A convenience method, SetPointComponent(),is also provided which does not require setting the (min,max) component range or the normalize flag (normalize is set to DefaulatNormalize value).

- `obj.SetPointComponent (int comp, string arrayName, int arrayComp)` - Define the component of the field to be used for the x, y, and z values of the points. Note that the parameter comp must lie between (0,2) and refers to the x-y-z (i.e., 0,1,2) components of the points. To define the field component to use you can specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract. (This method should be used for vtkPolyData, vtkUnstructuredGrid, vtkStructuredGrid, and vtkRectilinearGrid.) A convenience method, SetPointComponent(),is also provided which does not require setting the (min,max) component range or the normalize flag (normalize is set to DefaulatNormalize value).

- `string = obj.GetPointComponentArrayName (int comp)` - Define the component of the field to be used for the x, y, and z values of the points. Note that the parameter comp must lie between (0,2) and refers to the x-y-z (i.e., 0,1,2) components of the points. To define the field component to use you can specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract. (This method should be used for vtkPolyData, vtkUnstructuredGrid, vtkStructuredGrid, and vtkRectilinearGrid.) A convenience method, SetPointComponent(),is also provided which does not require setting the (min,max) component range or the normalize flag (normalize is set to DefaulatNormalize value).

- `int = obj.GetPointComponentArrayComponent (int comp)` - Define the component of the field to be used for the x, y, and z values of the points. Note that the parameter comp must lie between (0,2) and refers to the x-y-z (i.e., 0,1,2) components of the points. To define the field component to use you can specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract. (This method should be used for vtkPolyData, vtkUnstructuredGrid, vtkStructuredGrid, and vtkRectilinearGrid.) A convenience method, SetPointComponent(),is also provided which does not require setting the (min,max) component range or the normalize flag (normalize is set to DefaulatNormalize value).

- `int = obj.GetPointComponentMinRange (int comp)` - Define the component of the field to be used for the x, y, and z values of the points. Note that the parameter comp must lie between (0,2) and refers to the x-y-z (i.e., 0,1,2) components of the points. To define the field component to use you can specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract. (This method should be used for vtkPolyData, vtkUnstructuredGrid, vtkStructuredGrid, and vtkRectilinearGrid.) A convenience method, SetPointComponent(),is also provided which does not require setting the (min,max) component range or the normalize flag (normalize is set to DefaulatNormalize value).

- `int = obj.GetPointComponentMaxRange (int comp)` - Define the component of the field to be used for the x, y, and z values of the points. Note that the parameter comp must lie between (0,2) and refers to the x-y-z (i.e., 0,1,2) components of the points. To define the field component to use you can specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract. (This method should be used for vtkPolyData, vtkUnstructuredGrid, vtkStructuredGrid, and vtkRectilinearGrid.) A convenience method, SetPointComponent(),is

also provided which does not require setting the (min,max) component range or the normalize flag (normalize is set to DefaulatNormalize value).

- `int = obj.GetPointComponentNormailzeFlag (int comp)` - Define the component of the field to be used for the x, y, and z values of the points. Note that the parameter comp must lie between (0,2) and refers to the x-y-z (i.e., 0,1,2) components of the points. To define the field component to use you can specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract. (This method should be used for vtkPolyData, vtkUnstructuredGrid, vtkStructuredGrid, and vtkRectilinearGrid.) A convenience method, SetPointComponent(),is also provided which does not require setting the (min,max) component range or the normalize flag (normalize is set to DefaulatNormalize value).

- `obj.SetVertsComponent (string arrayName, int arrayComp, int min, int max)` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetVertsComponent (string arrayName, int arrayComp)` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `string = obj.GetVertsComponentArrayName ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetVertsComponentArrayComponent ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetVertsComponentMinRange ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetVertsComponentMaxRange ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetLinesComponent (string arrayName, int arrayComp, int min, int max)` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetLinesComponent (string arrayName, int arrayComp)` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `string = obj.GetLinesComponentArrayName ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetLinesComponentArrayComponent ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetLinesComponentMinRange ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetLinesComponentMaxRange ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetPolysComponent (string arrayName, int arrayComp, int min, int max)` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetPolysComponent (string arrayName, int arrayComp)` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `string = obj.GetPolysComponentArrayName ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetPolysComponentArrayComponent ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetPolysComponentMinRange ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetPolysComponentMaxRange ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetStripsComponent (string arrayName, int arrayComp, int min, int max)` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetStripsComponent (string arrayName, int arrayComp)` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `string = obj.GetStripsComponentArrayName ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetStripsComponentArrayComponent ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetStripsComponentMinRange ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetStripsComponentMaxRange ()` - Define cell connectivity when creating vtkPolyData. You can define vertices, lines, polygons, and/or triangle strips via these methods. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of values that (for each cell) includes the number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetCellTypeComponent (string arrayName, int arrayComp, int min, int max)` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetCellTypeComponent (string arrayName, int arrayComp)` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)
- `string = obj.GetCellTypeComponentArrayName ()` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)
- `int = obj.GetCellTypeComponentArrayComponent ()` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)
- `int = obj.GetCellTypeComponentMinRange ()` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)
- `int = obj.GetCellTypeComponentMaxRange ()` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)
- `obj.SetCellConnectivityComponent (string arrayName, int arrayComp, int min, int max)` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)
- `obj.SetCellConnectivityComponent (string arrayName, int arrayComp)` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)
- `string = obj.GetCellConnectivityComponentArrayName ()` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)
- `int = obj.GetCellConnectivityComponentArrayComponent ()` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetCellConnectivityComponentMinRange ()` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `int = obj.GetCellConnectivityComponentMaxRange ()` - Define cell types and cell connectivity when creating unstructured grid data. These methods are similar to those for defining points, except that no normalization of the data is possible. Basically, you need to define an array of cell types (an integer value per cell), and another array consisting (for each cell) of a number of points per cell, and then the cell connectivity. (This is the vtk file format described in in the textbook or User's Guide.)

- `obj.SetDefaultNormalize (int )` - Set the default Normalize() flag for those methods setting a default Normalize value (e.g., SetPointComponent).

- `int = obj.GetDefaultNormalize ()` - Set the default Normalize() flag for those methods setting a default Normalize value (e.g., SetPointComponent).

- `obj.DefaultNormalizeOn ()` - Set the default Normalize() flag for those methods setting a default Normalize value (e.g., SetPointComponent).

- `obj.DefaultNormalizeOff ()` - Set the default Normalize() flag for those methods setting a default Normalize value (e.g., SetPointComponent).

- `obj.SetDimensions (int , int , int )`

- `obj.SetDimensions (int  a[3])`

- `int = obj. GetDimensions ()`

- `obj.SetOrigin (double , double , double )`

- `obj.SetOrigin (double  a[3])`

- `double = obj. GetOrigin ()`

- `obj.SetSpacing (double , double , double )`

- `obj.SetSpacing (double  a[3])`

- `double = obj. GetSpacing ()`

- `obj.SetDimensionsComponent (string arrayName, int arrayComp, int min, int max)`

- `obj.SetDimensionsComponent (string arrayName, int arrayComp)`

- `obj.SetSpacingComponent (string arrayName, int arrayComp, int min, int max)`

- `obj.SetSpacingComponent (string arrayName, int arrayComp)`

- `obj.SetOriginComponent (string arrayName, int arrayComp, int min, int max)`

- `obj.SetOriginComponent (string arrayName, int arrayComp)`

33.47 vtkDataSetEdgeSubdivisionCriterion

33.47.1 Usage

This is a subclass of vtkEdgeSubdivisionCriterion that is used for tessellating cells of a vtkDataSet, particularly nonlinear cells.

It provides functions for setting the current cell being tessellated and a convenience routine, EvaluateFields() to evaluate field values at a point. You should call EvaluateFields() from inside EvaluateEdge() whenever the result of EvaluateEdge() will be true. Otherwise, do not call EvaluateFields() as the midpoint is about to be discarded. (¡i¿Implementor's note¡/i¿: This isn't true if UGLY_ASPECT_RATIO_HACK has been defined. But in that case, we don't want the exact field values; we need the linearly interpolated ones at the midpoint for continuity.)

To create an instance of class vtkDataSetEdgeSubdivisionCriterion, simply invoke its constructor as follows

```
obj = vtkDataSetEdgeSubdivisionCriterion
```

33.47.2 Methods

The class vtkDataSetEdgeSubdivisionCriterion has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetEdgeSubdivisionCriterion class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetEdgeSubdivisionCriterion = obj.NewInstance ()`
- `vtkDataSetEdgeSubdivisionCriterion = obj.SafeDownCast (vtkObject o)`
- `obj.SetMesh (vtkDataSet )`
- `vtkDataSet = obj.GetMesh ()`
- `obj.SetCellId (vtkIdType cell)`
- `vtkIdType = obj.GetCellId () const`
- `vtkCell = obj.GetCell ()`
- `bool = obj.EvaluateEdge (double p0, double midpt, double p1, int field\_start)`
- `obj.EvaluatePointDataField (double result, double weights, int field)` - Evaluate either a cell or nodal field. This exists because of the funky way that Exodus data will be handled. Sure, it's a hack, but what are ya gonna do?
- `obj.EvaluateCellDataField (double result, double weights, int field)` - Evaluate either a cell or nodal field. This exists because of the funky way that Exodus data will be handled. Sure, it's a hack, but what are ya gonna do?
- `obj.SetChordError2 (double )` - Get/Set the square of the allowable chord error at any edge's midpoint. This value is used by EvaluateEdge.
- `double = obj.GetChordError2 ()` - Get/Set the square of the allowable chord error at any edge's midpoint. This value is used by EvaluateEdge.
- `obj.SetFieldError2 (int s, double err)` - Get/Set the square of the allowable error magnitude for the scalar field s at any edge's midpoint. A value less than or equal to 0 indicates that the field should not be used as a criterion for subdivision.

- `double = obj.GetFieldError2 (int s) const` - Get/Set the square of the allowable error magnitude for the scalar field s at any edge's midpoint. A value less than or equal to 0 indicates that the field should not be used as a criterion for subdivision.
- `obj.ResetFieldError2 ()` - Tell the subdivider not to use any field values as subdivision criteria. Effectively calls SetFieldError2(a, -1.) for all fields.
- `int = obj.GetActiveFieldCriteria ()` - Return a bitfield specifying which FieldError2 criteria are positive (i.e., actively used to decide edge subdivisions). This is stored as separate state to make subdivisions go faster.
- `int = obj.GetActiveFieldCriteria () const`

33.48 vtkDataSetGradient

33.48.1 Usage

vtkDataSetGradient Computes per cell gradient of point scalar field or per point gradient of cell scalar field.

.SECTION Thanks This file is part of the generalized Youngs material interface reconstruction algorithm contributed by CEA/DIF - Commissariat a l'Energie Atomique, Centre DAM Ile-De-France ¡br¿ BP12, F-91297 Arpajon, France. ¡br¿ Implementation by Thierry Carrard (CEA)

To create an instance of class vtkDataSetGradient, simply invoke its constructor as follows

```
obj = vtkDataSetGradient
```

33.48.2 Methods

The class vtkDataSetGradient has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetGradient class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetGradient = obj.NewInstance ()`
- `vtkDataSetGradient = obj.SafeDownCast (vtkObject o)`
- `obj.SetResultArrayName (string )` - Set/Get the name of computed vector array.
- `string = obj.GetResultArrayName ()` - Set/Get the name of computed vector array.

33.49 vtkDataSetGradientPrecompute

33.49.1 Usage

Computes a geometry based vector field that the DataSetGradient filter uses to accelerate gradient computation. This vector field is added to FieldData since it has a different value for each vertex of each cell (a vertex shared by two cell has two differents values).

.SECTION Thanks This file is part of the generalized Youngs material interface reconstruction algorithm contributed by CEA/DIF - Commissariat a l'Energie Atomique, Centre DAM Ile-De-France ¡br¿ BP12, F-91297 Arpajon, France. ¡br¿ Implementation by Thierry Carrard (CEA)

To create an instance of class vtkDataSetGradientPrecompute, simply invoke its constructor as follows

```
obj = vtkDataSetGradientPrecompute
```

33.49.2 Methods

The class vtkDataSetGradientPrecompute has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetGradientPrecompute class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetGradientPrecompute = obj.NewInstance ()`
- `vtkDataSetGradientPrecompute = obj.SafeDownCast (vtkObject o)`

33.50 vtkDataSetSurfaceFilter

33.50.1 Usage

vtkDataSetSurfaceFilter is a faster version of vtkGeometry filter, but it does not have an option to select bounds. It may use more memory than vtkGeometryFilter. It only has one option: whether to use triangle strips when the input type is structured.

To create an instance of class vtkDataSetSurfaceFilter, simply invoke its constructor as follows

```
obj = vtkDataSetSurfaceFilter
```

33.50.2 Methods

The class vtkDataSetSurfaceFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetSurfaceFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetSurfaceFilter = obj.NewInstance ()`
- `vtkDataSetSurfaceFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetUseStrips (int )` - When input is structured data, this flag will generate faces with triangle strips. This should render faster and use less memory, but no cell data is copied. By default, UseStrips is Off.
- `int = obj.GetUseStrips ()` - When input is structured data, this flag will generate faces with triangle strips. This should render faster and use less memory, but no cell data is copied. By default, UseStrips is Off.
- `obj.UseStripsOn ()` - When input is structured data, this flag will generate faces with triangle strips. This should render faster and use less memory, but no cell data is copied. By default, UseStrips is Off.
- `obj.UseStripsOff ()` - When input is structured data, this flag will generate faces with triangle strips. This should render faster and use less memory, but no cell data is copied. By default, UseStrips is Off.
- `obj.SetPieceInvariant (int )` - If PieceInvariant is true, vtkDataSetSurfaceFilter requests 1 ghost level from input in order to remove internal surface that are between processes. False by default.

- `int = obj.GetPieceInvariant ()` - If PieceInvariant is true, vtkDataSetSurfaceFilter requests 1 ghost level from input in order to remove internal surface that are between processes. False by default.

- `obj.SetPassThroughCellIds (int )` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory. Note that PassThroughCellIds will be ignored if UseStrips is on, since in that case each tringle strip can represent more than on of the input cells.

- `int = obj.GetPassThroughCellIds ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory. Note that PassThroughCellIds will be ignored if UseStrips is on, since in that case each tringle strip can represent more than on of the input cells.

- `obj.PassThroughCellIdsOn ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory. Note that PassThroughCellIds will be ignored if UseStrips is on, since in that case each tringle strip can represent more than on of the input cells.

- `obj.PassThroughCellIdsOff ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory. Note that PassThroughCellIds will be ignored if UseStrips is on, since in that case each tringle strip can represent more than on of the input cells.

- `obj.SetPassThroughPointIds (int )` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory. Note that PassThroughCellIds will be ignored if UseStrips is on, since in that case each tringle strip can represent more than on of the input cells.

- `int = obj.GetPassThroughPointIds ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory. Note that PassThroughCellIds will be ignored if UseStrips is on, since in that case each tringle strip can represent more than on of the input cells.

- `obj.PassThroughPointIdsOn ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory. Note that PassThroughCellIds will be ignored if UseStrips is on, since in that case each tringle strip can represent more than on of the input cells.

- `obj.PassThroughPointIdsOff ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory. Note that PassThroughCellIds will be ignored if UseStrips is on, since in that case each tringle strip can represent more than on of the input cells.

- `int = obj.StructuredExecute (vtkDataSet input, vtkPolyData output, int ext32, int wholeExt32)` - Direct access methods that can be used to use the this class as an algorithm without using it as a filter.

- `int = obj.UnstructuredGridExecute (vtkDataSet input, vtkPolyData output)` - Direct access methods that can be used to use the this class as an algorithm without using it as a filter.

- `int = obj.DataSetExecute (vtkDataSet input, vtkPolyData output)` - Direct access methods that can be used to use the this class as an algorithm without using it as a filter.

33.51 vtkDataSetToDataObjectFilter

33.51.1 Usage

vtkDataSetToDataObjectFilter is an class that transforms a dataset into data object (i.e., a field). The field will have labeled data arrays corresponding to the topology, geometry, field data, and point and cell attribute data.

You can control what portions of the dataset are converted into the output data object's field data. The instance variables Geometry, Topology, FieldData, PointData, and CellData are flags that control whether the dataset's geometry (e.g., points, spacing, origin); topology (e.g., cell connectivity, dimensions); the field data associated with the dataset's superclass data object; the dataset's point data attributes; and the dataset's cell data attributes. (Note: the data attributes include scalars, vectors, tensors, normals, texture coordinates, and field data.)

The names used to create the field data are as follows. For vtkPolyData, "Points", "Verts", "Lines", "Polys", and "Strips". For vtkUnstructuredGrid, "Cells" and "CellTypes". For vtkStructuredPoints, "Dimensions", "Spacing", and "Origin". For vtkStructuredGrid, "Points" and "Dimensions". For vtkRectilinearGrid, "XCoordinates", "YCoordinates", and "ZCoordinates". for point attribute data, "PointScalars", "PointVectors", etc. For cell attribute data, "CellScalars", "CellVectors", etc. Field data arrays retain their original name.

To create an instance of class vtkDataSetToDataObjectFilter, simply invoke its constructor as follows

```
obj = vtkDataSetToDataObjectFilter
```

33.51.2 Methods

The class vtkDataSetToDataObjectFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetToDataObjectFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetToDataObjectFilter = obj.NewInstance ()`
- `vtkDataSetToDataObjectFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetGeometry (int )` - Turn on/off the conversion of dataset geometry to a data object.
- `int = obj.GetGeometry ()` - Turn on/off the conversion of dataset geometry to a data object.
- `obj.GeometryOn ()` - Turn on/off the conversion of dataset geometry to a data object.
- `obj.GeometryOff ()` - Turn on/off the conversion of dataset geometry to a data object.
- `obj.SetTopology (int )` - Turn on/off the conversion of dataset topology to a data object.
- `int = obj.GetTopology ()` - Turn on/off the conversion of dataset topology to a data object.
- `obj.TopologyOn ()` - Turn on/off the conversion of dataset topology to a data object.
- `obj.TopologyOff ()` - Turn on/off the conversion of dataset topology to a data object.
- `obj.SetFieldData (int )` - Turn on/off the conversion of dataset field data to a data object.
- `int = obj.GetFieldData ()` - Turn on/off the conversion of dataset field data to a data object.
- `obj.FieldDataOn ()` - Turn on/off the conversion of dataset field data to a data object.
- `obj.FieldDataOff ()` - Turn on/off the conversion of dataset field data to a data object.

- `obj.SetPointData (int )` - Turn on/off the conversion of dataset point data to a data object.
- `int = obj.GetPointData ()` - Turn on/off the conversion of dataset point data to a data object.
- `obj.PointDataOn ()` - Turn on/off the conversion of dataset point data to a data object.
- `obj.PointDataOff ()` - Turn on/off the conversion of dataset point data to a data object.
- `obj.SetCellData (int )` - Turn on/off the conversion of dataset cell data to a data object.
- `int = obj.GetCellData ()` - Turn on/off the conversion of dataset cell data to a data object.
- `obj.CellDataOn ()` - Turn on/off the conversion of dataset cell data to a data object.
- `obj.CellDataOff ()` - Turn on/off the conversion of dataset cell data to a data object.

33.52 vtkDataSetTriangleFilter

33.52.1 Usage

vtkDataSetTriangleFilter generates n-dimensional simplices from any input dataset. That is, 3D cells are converted to tetrahedral meshes, 2D cells to triangles, and so on. The triangulation is guaranteed to be compatible.

This filter uses simple 1D and 2D triangulation techniques for cells that are of topological dimension 2 or less. For 3D cells–due to the issue of face compatibility across quadrilateral faces (which way to orient the diagonal?)–a fancier ordered Delaunay triangulation is used. This approach produces templates on the fly for triangulating the cells. The templates are then used to do the actual triangulation.

To create an instance of class vtkDataSetTriangleFilter, simply invoke its constructor as follows

```
obj = vtkDataSetTriangleFilter
```

33.52.2 Methods

The class vtkDataSetTriangleFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetTriangleFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetTriangleFilter = obj.NewInstance ()`
- `vtkDataSetTriangleFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetTetrahedraOnly (int )` - When On this filter will cull all 1D and 2D cells from the output. The default is Off.
- `int = obj.GetTetrahedraOnly ()` - When On this filter will cull all 1D and 2D cells from the output. The default is Off.
- `obj.TetrahedraOnlyOn ()` - When On this filter will cull all 1D and 2D cells from the output. The default is Off.
- `obj.TetrahedraOnlyOff ()` - When On this filter will cull all 1D and 2D cells from the output. The default is Off.

33.53 vtkDecimatePolylineFilter

33.53.1 Usage

vtkDecimatePolylineFilter is a filter to reduce the number of lines in a polyline. The algorithm functions by evaluating an error metric for each vertex (i.e., the distance of the vertex to a line defined from the two vertices on either side of the vertex). Then, these vertices are placed into a priority queue, and those with larger errors are deleted first. The decimation continues until the target reduction is reached.

To create an instance of class vtkDecimatePolylineFilter, simply invoke its constructor as follows

```
obj = vtkDecimatePolylineFilter
```

33.53.2 Methods

The class vtkDecimatePolylineFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDecimatePolylineFilter class.

- `string = obj.GetClassName ()` - Standard methods for type information and printing.
- `int = obj.IsA (string name)` - Standard methods for type information and printing.
- `vtkDecimatePolylineFilter = obj.NewInstance ()` - Standard methods for type information and printing.
- `vtkDecimatePolylineFilter = obj.SafeDownCast (vtkObject o)` - Standard methods for type information and printing.
- `obj.SetTargetReduction (double )` - Specify the desired reduction in the total number of polygons (e.g., if TargetReduction is set to 0.9, this filter will try to reduce the data set to 10
- `double = obj.GetTargetReductionMinValue ()` - Specify the desired reduction in the total number of polygons (e.g., if TargetReduction is set to 0.9, this filter will try to reduce the data set to 10
- `double = obj.GetTargetReductionMaxValue ()` - Specify the desired reduction in the total number of polygons (e.g., if TargetReduction is set to 0.9, this filter will try to reduce the data set to 10
- `double = obj.GetTargetReduction ()` - Specify the desired reduction in the total number of polygons (e.g., if TargetReduction is set to 0.9, this filter will try to reduce the data set to 10

33.54 vtkDecimatePro

33.54.1 Usage

vtkDecimatePro is a filter to reduce the number of triangles in a triangle mesh, forming a good approximation to the original geometry. The input to vtkDecimatePro is a vtkPolyData object, and only triangles are treated. If you desire to decimate polygonal meshes, first triangulate the polygons with vtkTriangleFilter object.

The implementation of vtkDecimatePro is similar to the algorithm originally described in "Decimation of Triangle Meshes", Proc Siggraph '92, with three major differences. First, this algorithm does not necessarily preserve the topology of the mesh. Second, it is guaranteed to give the a mesh reduction factor specified by the user (as long as certain constraints are not set - see Caveats). Third, it is set up generate progressive meshes, that is a stream of operations that can be easily transmitted and incrementally updated (see Hugues Hoppe's Siggraph '96 paper on progressive meshes).

The algorithm proceeds as follows. Each vertex in the mesh is classified and inserted into a priority queue. The priority is based on the error to delete the vertex and retriangulate the hole. Vertices that cannot be deleted or triangulated (at this point in the algorithm) are skipped. Then, each vertex in the priority queue

is processed (i.e., deleted followed by hole triangulation using edge collapse). This continues until the priority queue is empty. Next, all remaining vertices are processed, and the mesh is split into separate pieces along sharp edges or at non-manifold attachment points and reinserted into the priority queue. Again, the priority queue is processed until empty. If the desired reduction is still not achieved, the remaining vertices are split as necessary (in a recursive fashion) so that it is possible to eliminate every triangle as necessary.

To use this object, at a minimum you need to specify the ivar TargetReduction. The algorithm is guaranteed to generate a reduced mesh at this level as long as the following four conditions are met: 1) topology modification is allowed (i.e., the ivar PreserveTopology is off); 2) mesh splitting is enabled (i.e., the ivar Splitting is on); 3) the algorithm is allowed to modify the boundary of the mesh (i.e., the ivar BoundaryVertexDeletion is on); and 4) the maximum allowable error (i.e., the ivar MaximumError) is set to VTK_DOUBLE_MAX. Other important parameters to adjust include the FeatureAngle and SplitAngle ivars, since these can impact the quality of the final mesh. Also, you can set the ivar AccumulateError to force incremental error update and distribution to surrounding vertices as each vertex is deleted. The accumulated error is a conservative global error bounds and decimation error, but requires additional memory and time to compute.

To create an instance of class vtkDecimatePro, simply invoke its constructor as follows

```
obj = vtkDecimatePro
```

33.54.2 Methods

The class vtkDecimatePro has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDecimatePro class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDecimatePro = obj.NewInstance ()`
- `vtkDecimatePro = obj.SafeDownCast (vtkObject o)`
- `obj.SetTargetReduction (double )` - Specify the desired reduction in the total number of polygons (e.g., if TargetReduction is set to 0.9, this filter will try to reduce the data set to 10reduction may not be realized. If you want to guarantee a particular reduction, you must turn off PreserveTopology, turn on SplitEdges and BoundaryVertexDeletion, and set the MaximumError to VTK_DOUBLE_MAX (these ivars are initialized this way when the object is instantiated).
- `double = obj.GetTargetReductionMinValue ()` - Specify the desired reduction in the total number of polygons (e.g., if TargetReduction is set to 0.9, this filter will try to reduce the data set to 10reduction may not be realized. If you want to guarantee a particular reduction, you must turn off PreserveTopology, turn on SplitEdges and BoundaryVertexDeletion, and set the MaximumError to VTK_DOUBLE_MAX (these ivars are initialized this way when the object is instantiated).
- `double = obj.GetTargetReductionMaxValue ()` - Specify the desired reduction in the total number of polygons (e.g., if TargetReduction is set to 0.9, this filter will try to reduce the data set to 10reduction may not be realized. If you want to guarantee a particular reduction, you must turn off PreserveTopology, turn on SplitEdges and BoundaryVertexDeletion, and set the MaximumError to VTK_DOUBLE_MAX (these ivars are initialized this way when the object is instantiated).
- `double = obj.GetTargetReduction ()` - Specify the desired reduction in the total number of polygons (e.g., if TargetReduction is set to 0.9, this filter will try to reduce the data set to 10reduction may not be realized. If you want to guarantee a particular reduction, you must turn off PreserveTopology, turn on SplitEdges and BoundaryVertexDeletion, and set the MaximumError to VTK_DOUBLE_MAX (these ivars are initialized this way when the object is instantiated).

- `obj.SetPreserveTopology (int )` - Turn on/off whether to preserve the topology of the original mesh. If on, mesh splitting and hole elimination will not occur. This may limit the maximum reduction that may be achieved.
- `int = obj.GetPreserveTopology ()` - Turn on/off whether to preserve the topology of the original mesh. If on, mesh splitting and hole elimination will not occur. This may limit the maximum reduction that may be achieved.
- `obj.PreserveTopologyOn ()` - Turn on/off whether to preserve the topology of the original mesh. If on, mesh splitting and hole elimination will not occur. This may limit the maximum reduction that may be achieved.
- `obj.PreserveTopologyOff ()` - Turn on/off whether to preserve the topology of the original mesh. If on, mesh splitting and hole elimination will not occur. This may limit the maximum reduction that may be achieved.
- `obj.SetFeatureAngle (double )` - Specify the mesh feature angle. This angle is used to define what an edge is (i.e., if the surface normal between two adjacent triangles is ¿= FeatureAngle, an edge exists).
- `double = obj.GetFeatureAngleMinValue ()` - Specify the mesh feature angle. This angle is used to define what an edge is (i.e., if the surface normal between two adjacent triangles is ¿= FeatureAngle, an edge exists).
- `double = obj.GetFeatureAngleMaxValue ()` - Specify the mesh feature angle. This angle is used to define what an edge is (i.e., if the surface normal between two adjacent triangles is ¿= FeatureAngle, an edge exists).
- `double = obj.GetFeatureAngle ()` - Specify the mesh feature angle. This angle is used to define what an edge is (i.e., if the surface normal between two adjacent triangles is ¿= FeatureAngle, an edge exists).
- `obj.SetSplitting (int )` - Turn on/off the splitting of the mesh at corners, along edges, at non-manifold points, or anywhere else a split is required. Turning splitting off will better preserve the original topology of the mesh, but you may not obtain the requested reduction.
- `int = obj.GetSplitting ()` - Turn on/off the splitting of the mesh at corners, along edges, at non-manifold points, or anywhere else a split is required. Turning splitting off will better preserve the original topology of the mesh, but you may not obtain the requested reduction.
- `obj.SplittingOn ()` - Turn on/off the splitting of the mesh at corners, along edges, at non-manifold points, or anywhere else a split is required. Turning splitting off will better preserve the original topology of the mesh, but you may not obtain the requested reduction.
- `obj.SplittingOff ()` - Turn on/off the splitting of the mesh at corners, along edges, at non-manifold points, or anywhere else a split is required. Turning splitting off will better preserve the original topology of the mesh, but you may not obtain the requested reduction.
- `obj.SetSplitAngle (double )` - Specify the mesh split angle. This angle is used to control the splitting of the mesh. A split line exists when the surface normals between two edge connected triangles are ¿= SplitAngle.
- `double = obj.GetSplitAngleMinValue ()` - Specify the mesh split angle. This angle is used to control the splitting of the mesh. A split line exists when the surface normals between two edge connected triangles are ¿= SplitAngle.
- `double = obj.GetSplitAngleMaxValue ()` - Specify the mesh split angle. This angle is used to control the splitting of the mesh. A split line exists when the surface normals between two edge connected triangles are ¿= SplitAngle.

- `double = obj.GetSplitAngle ()` - Specify the mesh split angle. This angle is used to control the splitting of the mesh. A split line exists when the surface normals between two edge connected triangles are ¿= SplitAngle.

- `obj.SetPreSplitMesh (int )` - In some cases you may wish to split the mesh prior to algorithm execution. This separates the mesh into semi-planar patches, which are disconnected from each other. This can give superior results in some cases. If the ivar PreSplitMesh ivar is enabled, the mesh is split with the specified SplitAngle. Otherwise mesh splitting is deferred as long as possible.

- `int = obj.GetPreSplitMesh ()` - In some cases you may wish to split the mesh prior to algorithm execution. This separates the mesh into semi-planar patches, which are disconnected from each other. This can give superior results in some cases. If the ivar PreSplitMesh ivar is enabled, the mesh is split with the specified SplitAngle. Otherwise mesh splitting is deferred as long as possible.

- `obj.PreSplitMeshOn ()` - In some cases you may wish to split the mesh prior to algorithm execution. This separates the mesh into semi-planar patches, which are disconnected from each other. This can give superior results in some cases. If the ivar PreSplitMesh ivar is enabled, the mesh is split with the specified SplitAngle. Otherwise mesh splitting is deferred as long as possible.

- `obj.PreSplitMeshOff ()` - In some cases you may wish to split the mesh prior to algorithm execution. This separates the mesh into semi-planar patches, which are disconnected from each other. This can give superior results in some cases. If the ivar PreSplitMesh ivar is enabled, the mesh is split with the specified SplitAngle. Otherwise mesh splitting is deferred as long as possible.

- `obj.SetMaximumError (double )` - Set the largest decimation error that is allowed during the decimation process. This may limit the maximum reduction that may be achieved. The maximum error is specified as a fraction of the maximum length of the input data bounding box.

- `double = obj.GetMaximumErrorMinValue ()` - Set the largest decimation error that is allowed during the decimation process. This may limit the maximum reduction that may be achieved. The maximum error is specified as a fraction of the maximum length of the input data bounding box.

- `double = obj.GetMaximumErrorMaxValue ()` - Set the largest decimation error that is allowed during the decimation process. This may limit the maximum reduction that may be achieved. The maximum error is specified as a fraction of the maximum length of the input data bounding box.

- `double = obj.GetMaximumError ()` - Set the largest decimation error that is allowed during the decimation process. This may limit the maximum reduction that may be achieved. The maximum error is specified as a fraction of the maximum length of the input data bounding box.

- `obj.SetAccumulateError (int )` - The computed error can either be computed directly from the mesh or the error may be accumulated as the mesh is modified. If the error is accumulated, then it represents a global error bounds, and the ivar MaximumError becomes a global bounds on mesh error. Accumulating the error requires extra memory proportional to the number of vertices in the mesh. If AccumulateError is off, then the error is not accumulated.

- `int = obj.GetAccumulateError ()` - The computed error can either be computed directly from the mesh or the error may be accumulated as the mesh is modified. If the error is accumulated, then it represents a global error bounds, and the ivar MaximumError becomes a global bounds on mesh error. Accumulating the error requires extra memory proportional to the number of vertices in the mesh. If AccumulateError is off, then the error is not accumulated.

- `obj.AccumulateErrorOn ()` - The computed error can either be computed directly from the mesh or the error may be accumulated as the mesh is modified. If the error is accumulated, then it represents a global error bounds, and the ivar MaximumError becomes a global bounds on mesh error. Accumulating the error requires extra memory proportional to the number of vertices in the mesh. If AccumulateError is off, then the error is not accumulated.

- `obj.AccumulateErrorOff ()` - The computed error can either be computed directly from the mesh or the error may be accumulated as the mesh is modified. If the error is accumulated, then it represents a global error bounds, and the ivar MaximumError becomes a global bounds on mesh error. Accumulating the error requires extra memory proportional to the number of vertices in the mesh. If AccumulateError is off, then the error is not accumulated.

- `obj.SetErrorIsAbsolute (int )` - The MaximumError is normally defined as a fraction of the dataset bounding diagonal. By setting ErrorIsAbsolute to 1, the error is instead defined as that specified by AbsoluteError. By default ErrorIsAbsolute=0.

- `int = obj.GetErrorIsAbsolute ()` - The MaximumError is normally defined as a fraction of the dataset bounding diagonal. By setting ErrorIsAbsolute to 1, the error is instead defined as that specified by AbsoluteError. By default ErrorIsAbsolute=0.

- `obj.SetAbsoluteError (double )` - Same as MaximumError, but to be used when ErrorIsAbsolute is 1

- `double = obj.GetAbsoluteErrorMinValue ()` - Same as MaximumError, but to be used when ErrorIsAbsolute is 1

- `double = obj.GetAbsoluteErrorMaxValue ()` - Same as MaximumError, but to be used when ErrorIsAbsolute is 1

- `double = obj.GetAbsoluteError ()` - Same as MaximumError, but to be used when ErrorIsAbsolute is 1

- `obj.SetBoundaryVertexDeletion (int )` - Turn on/off the deletion of vertices on the boundary of a mesh. This may limit the maximum reduction that may be achieved.

- `int = obj.GetBoundaryVertexDeletion ()` - Turn on/off the deletion of vertices on the boundary of a mesh. This may limit the maximum reduction that may be achieved.

- `obj.BoundaryVertexDeletionOn ()` - Turn on/off the deletion of vertices on the boundary of a mesh. This may limit the maximum reduction that may be achieved.

- `obj.BoundaryVertexDeletionOff ()` - Turn on/off the deletion of vertices on the boundary of a mesh. This may limit the maximum reduction that may be achieved.

- `obj.SetDegree (int )` - If the number of triangles connected to a vertex exceeds "Degree", then the vertex will be split. (NOTE: the complexity of the triangulation algorithm is proportional to Degree$\hat{}$2. Setting degree small can improve the performance of the algorithm.)

- `int = obj.GetDegreeMinValue ()` - If the number of triangles connected to a vertex exceeds "Degree", then the vertex will be split. (NOTE: the complexity of the triangulation algorithm is proportional to Degree$\hat{}$2. Setting degree small can improve the performance of the algorithm.)

- `int = obj.GetDegreeMaxValue ()` - If the number of triangles connected to a vertex exceeds "Degree", then the vertex will be split. (NOTE: the complexity of the triangulation algorithm is proportional to Degree$\hat{}$2. Setting degree small can improve the performance of the algorithm.)

- `int = obj.GetDegree ()` - If the number of triangles connected to a vertex exceeds "Degree", then the vertex will be split. (NOTE: the complexity of the triangulation algorithm is proportional to Degree$\hat{}$2. Setting degree small can improve the performance of the algorithm.)

- `obj.SetInflectionPointRatio (double )` - Specify the inflection point ratio. An inflection point occurs when the ratio of reduction error between two iterations is greater than or equal to the InflectionPointRatio.

- `double = obj.GetInflectionPointRatioMinValue ()` - Specify the inflection point ratio. An inflection point occurs when the ratio of reduction error between two iterations is greater than or equal to the InflectionPointRatio.
- `double = obj.GetInflectionPointRatioMaxValue ()` - Specify the inflection point ratio. An inflection point occurs when the ratio of reduction error between two iterations is greater than or equal to the InflectionPointRatio.
- `double = obj.GetInflectionPointRatio ()` - Specify the inflection point ratio. An inflection point occurs when the ratio of reduction error between two iterations is greater than or equal to the InflectionPointRatio.
- `vtkIdType = obj.GetNumberOfInflectionPoints ()` - Get the number of inflection points. Only returns a valid value after the filter has executed. The values in the list are mesh reduction values at each inflection point. Note: the first inflection point always occurs right before non-planar triangles are decimated (i.e., as the error becomes non-zero).
- `obj.GetInflectionPoints (double inflectionPoints)` - Get a list of inflection points. These are double values 0 ¡ r ¡= 1.0 corresponding to reduction level, and there are a total of NumberOfInflectionPoints() values. You must provide an array (of the correct size) into which the inflection points are written.

33.55 vtkDelaunay2D

33.55.1 Usage

vtkDelaunay2D is a filter that constructs a 2D Delaunay triangulation from a list of input points. These points may be represented by any dataset of type vtkPointSet and subclasses. The output of the filter is a polygonal dataset. Usually the output is a triangle mesh, but if a non-zero alpha distance value is specified (called the "alpha" value), then only triangles, edges, and vertices lying within the alpha radius are output. In other words, non-zero alpha values may result in arbitrary combinations of triangles, lines, and vertices. (The notion of alpha value is derived from Edelsbrunner's work on "alpha shapes".) Also, it is possible to generate "constrained triangulations" using this filter. A constrained triangulation is one where edges and loops (i.e., polygons) can be defined and the triangulation will preserve them (read on for more information).

The 2D Delaunay triangulation is defined as the triangulation that satisfies the Delaunay criterion for n-dimensional simplexes (in this case n=2 and the simplexes are triangles). This criterion states that a circumsphere of each simplex in a triangulation contains only the n+1 defining points of the simplex. (See "The Visualization Toolkit" text for more information.) In two dimensions, this translates into an optimal triangulation. That is, the maximum interior angle of any triangle is less than or equal to that of any possible triangulation.

Delaunay triangulations are used to build topological structures from unorganized (or unstructured) points. The input to this filter is a list of points specified in 3D, even though the triangulation is 2D. Thus the triangulation is constructed in the x-y plane, and the z coordinate is ignored (although carried through to the output). If you desire to triangulate in a different plane, you can use the vtkTransformFilter to transform the points into and out of the x-y plane or you can specify a transform to the Delaunay2D directly. In the latter case, the input points are transformed, the transformed points are triangulated, and the output will use the triangulated topology for the original (non-transformed) points. This avoids transforming the data back as would be required when using the vtkTransformFilter method. Specifying a transform directly also allows any transform to be used: rigid, non-rigid, non-invertible, etc.

If an input transform is used, then alpha values are applied (for the most part) in the original data space. The exception is when BoundingTriangulation is on. In this case, alpha values are applied in the original data space unless a cell uses a bounding vertex.

The Delaunay triangulation can be numerically sensitive in some cases. To prevent problems, try to avoid injecting points that will result in triangles with bad aspect ratios (1000:1 or greater). In practice this means inserting points that are "widely dispersed", and enables smooth transition of triangle sizes throughout the

mesh. (You may even want to add extra points to create a better point distribution.) If numerical problems are present, you will see a warning message to this effect at the end of the triangulation process.

To create constrained meshes, you must define an additional input. This input is an instance of vtkPolyData which contains lines, polylines, and/or polygons that define constrained edges and loops. Only the topology of (lines and polygons) from this second input are used. The topology is assumed to reference points in the input point set (the one to be triangulated). In other words, the lines and polygons use point ids from the first input point set. Lines and polylines found in the input will be mesh edges in the output. Polygons define a loop with inside and outside regions. The inside of the polygon is determined by using the right-hand-rule, i.e., looking down the z-axis a polygon should be ordered counter-clockwise. Holes in a polygon should be ordered clockwise. If you choose to create a constrained triangulation, the final mesh may not satisfy the Delaunay criterion. (Noted: the lines/polygon edges must not intersect when projected onto the 2D plane. It may not be possible to recover all edges due to not enough points in the triangulation, or poorly defined edges (coincident or excessively long). The form of the lines or polygons is a list of point ids that correspond to the input point ids used to generate the triangulation.)

If an input transform is used, constraints are defined in the "transformed" space. So when the right hand rule is used for a polygon constraint, that operation is applied using the transformed points. Since the input transform can be any transformation (rigid or non-rigid), care must be taken in constructing constraints when an input transform is used.

To create an instance of class vtkDelaunay2D, simply invoke its constructor as follows

```
obj = vtkDelaunay2D
```

33.55.2 Methods

The class vtkDelaunay2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDelaunay2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDelaunay2D = obj.NewInstance ()`
- `vtkDelaunay2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetSource (vtkPolyData )` - Specify the source object used to specify constrained edges and loops. (This is optional.) If set, and lines/polygons are defined, a constrained triangulation is created. The lines/polygons are assumed to reference points in the input point set (i.e. point ids are identical in the input and source). Old style. See SetSourceConnection.
- `obj.SetSourceConnection (vtkAlgorithmOutput algOutput)` - Specify the source object used to specify constrained edges and loops. (This is optional.) If set, and lines/polygons are defined, a constrained triangulation is created. The lines/polygons are assumed to reference points in the input point set (i.e. point ids are identical in the input and source). New style. This method is equivalent to SetInputConnection(1, algOutput).
- `vtkPolyData = obj.GetSource ()` - Get a pointer to the source object.
- `obj.SetAlpha (double )` - Specify alpha (or distance) value to control output of this filter. For a non-zero alpha value, only edges or triangles contained within a sphere centered at mesh vertices will be output. Otherwise, only triangles will be output.
- `double = obj.GetAlphaMinValue ()` - Specify alpha (or distance) value to control output of this filter. For a non-zero alpha value, only edges or triangles contained within a sphere centered at mesh vertices will be output. Otherwise, only triangles will be output.

- `double = obj.GetAlphaMaxValue ()` - Specify alpha (or distance) value to control output of this filter. For a non-zero alpha value, only edges or triangles contained within a sphere centered at mesh vertices will be output. Otherwise, only triangles will be output.
- `double = obj.GetAlpha ()` - Specify alpha (or distance) value to control output of this filter. For a non-zero alpha value, only edges or triangles contained within a sphere centered at mesh vertices will be output. Otherwise, only triangles will be output.
- `obj.SetTolerance (double )` - Specify a tolerance to control discarding of closely spaced points. This tolerance is specified as a fraction of the diagonal length of the bounding box of the points.
- `double = obj.GetToleranceMinValue ()` - Specify a tolerance to control discarding of closely spaced points. This tolerance is specified as a fraction of the diagonal length of the bounding box of the points.
- `double = obj.GetToleranceMaxValue ()` - Specify a tolerance to control discarding of closely spaced points. This tolerance is specified as a fraction of the diagonal length of the bounding box of the points.
- `double = obj.GetTolerance ()` - Specify a tolerance to control discarding of closely spaced points. This tolerance is specified as a fraction of the diagonal length of the bounding box of the points.
- `obj.SetOffset (double )` - Specify a multiplier to control the size of the initial, bounding Delaunay triangulation.
- `double = obj.GetOffsetMinValue ()` - Specify a multiplier to control the size of the initial, bounding Delaunay triangulation.
- `double = obj.GetOffsetMaxValue ()` - Specify a multiplier to control the size of the initial, bounding Delaunay triangulation.
- `double = obj.GetOffset ()` - Specify a multiplier to control the size of the initial, bounding Delaunay triangulation.
- `obj.SetBoundingTriangulation (int )` - Boolean controls whether bounding triangulation points (and associated triangles) are included in the output. (These are introduced as an initial triangulation to begin the triangulation process. This feature is nice for debugging output.)
- `int = obj.GetBoundingTriangulation ()` - Boolean controls whether bounding triangulation points (and associated triangles) are included in the output. (These are introduced as an initial triangulation to begin the triangulation process. This feature is nice for debugging output.)
- `obj.BoundingTriangulationOn ()` - Boolean controls whether bounding triangulation points (and associated triangles) are included in the output. (These are introduced as an initial triangulation to begin the triangulation process. This feature is nice for debugging output.)
- `obj.BoundingTriangulationOff ()` - Boolean controls whether bounding triangulation points (and associated triangles) are included in the output. (These are introduced as an initial triangulation to begin the triangulation process. This feature is nice for debugging output.)
- `obj.SetTransform (vtkAbstractTransform )` - Set / get the transform which is applied to points to generate a 2D problem. This maps a 3D dataset into a 2D dataset where triangulation can be done on the XY plane. The points are transformed and triangulated. The topology of triangulated points is used as the output topology. The output points are the original (untransformed) points. The transform can be any subclass of vtkAbstractTransform (thus it does not need to be a linear or invertible transform).
- `vtkAbstractTransform = obj.GetTransform ()` - Set / get the transform which is applied to points to generate a 2D problem. This maps a 3D dataset into a 2D dataset where triangulation can be done on the XY plane. The points are transformed and triangulated. The topology of triangulated points is used as the output topology. The output points are the original (untransformed) points. The transform can be any subclass of vtkAbstractTransform (thus it does not need to be a linear or invertible transform).

- `obj.SetProjectionPlaneMode (int )` - Define
- `int = obj.GetProjectionPlaneModeMinValue ()` - Define
- `int = obj.GetProjectionPlaneModeMaxValue ()` - Define
- `int = obj.GetProjectionPlaneMode ()` - Define

33.56 vtkDelaunay3D

33.56.1 Usage

vtkDelaunay3D is a filter that constructs a 3D Delaunay triangulation from a list of input points. These points may be represented by any dataset of type vtkPointSet and subclasses. The output of the filter is an unstructured grid dataset. Usually the output is a tetrahedral mesh, but if a non-zero alpha distance value is specified (called the "alpha" value), then only tetrahedra, triangles, edges, and vertices lying within the alpha radius are output. In other words, non-zero alpha values may result in arbitrary combinations of tetrahedra, triangles, lines, and vertices. (The notion of alpha value is derived from Edelsbrunner's work on "alpha shapes".)

The 3D Delaunay triangulation is defined as the triangulation that satisfies the Delaunay criterion for n-dimensional simplexes (in this case n=3 and the simplexes are tetrahedra). This criterion states that a circumsphere of each simplex in a triangulation contains only the n+1 defining points of the simplex. (See text for more information.) While in two dimensions this translates into an "optimal" triangulation, this is not true in 3D, since a measurement for optimality in 3D is not agreed on.

Delaunay triangulations are used to build topological structures from unorganized (or unstructured) points. The input to this filter is a list of points specified in 3D. (If you wish to create 2D triangulations see vtkDelaunay2D.) The output is an unstructured grid.

The Delaunay triangulation can be numerically sensitive. To prevent problems, try to avoid injecting points that will result in triangles with bad aspect ratios (1000:1 or greater). In practice this means inserting points that are "widely dispersed", and enables smooth transition of triangle sizes throughout the mesh. (You may even want to add extra points to create a better point distribution.) If numerical problems are present, you will see a warning message to this effect at the end of the triangulation process.

To create an instance of class vtkDelaunay3D, simply invoke its constructor as follows

```
obj = vtkDelaunay3D
```

33.56.2 Methods

The class vtkDelaunay3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDelaunay3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDelaunay3D = obj.NewInstance ()`
- `vtkDelaunay3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetAlpha (double )` - Specify alpha (or distance) value to control output of this filter. For a non-zero alpha value, only edges, faces, or tetra contained within the circumsphere (of radius alpha) will be output. Otherwise, only tetrahedra will be output.
- `double = obj.GetAlphaMinValue ()` - Specify alpha (or distance) value to control output of this filter. For a non-zero alpha value, only edges, faces, or tetra contained within the circumsphere (of radius alpha) will be output. Otherwise, only tetrahedra will be output.

- `double = obj.GetAlphaMaxValue ()` - Specify alpha (or distance) value to control output of this filter. For a non-zero alpha value, only edges, faces, or tetra contained within the circumsphere (of radius alpha) will be output. Otherwise, only tetrahedra will be output.

- `double = obj.GetAlpha ()` - Specify alpha (or distance) value to control output of this filter. For a non-zero alpha value, only edges, faces, or tetra contained within the circumsphere (of radius alpha) will be output. Otherwise, only tetrahedra will be output.

- `obj.SetTolerance (double )` - Specify a tolerance to control discarding of closely spaced points. This tolerance is specified as a fraction of the diagonal length of the bounding box of the points.

- `double = obj.GetToleranceMinValue ()` - Specify a tolerance to control discarding of closely spaced points. This tolerance is specified as a fraction of the diagonal length of the bounding box of the points.

- `double = obj.GetToleranceMaxValue ()` - Specify a tolerance to control discarding of closely spaced points. This tolerance is specified as a fraction of the diagonal length of the bounding box of the points.

- `double = obj.GetTolerance ()` - Specify a tolerance to control discarding of closely spaced points. This tolerance is specified as a fraction of the diagonal length of the bounding box of the points.

- `obj.SetOffset (double )` - Specify a multiplier to control the size of the initial, bounding Delaunay triangulation.

- `double = obj.GetOffsetMinValue ()` - Specify a multiplier to control the size of the initial, bounding Delaunay triangulation.

- `double = obj.GetOffsetMaxValue ()` - Specify a multiplier to control the size of the initial, bounding Delaunay triangulation.

- `double = obj.GetOffset ()` - Specify a multiplier to control the size of the initial, bounding Delaunay triangulation.

- `obj.SetBoundingTriangulation (int )` - Boolean controls whether bounding triangulation points (and associated triangles) are included in the output. (These are introduced as an initial triangulation to begin the triangulation process. This feature is nice for debugging output.)

- `int = obj.GetBoundingTriangulation ()` - Boolean controls whether bounding triangulation points (and associated triangles) are included in the output. (These are introduced as an initial triangulation to begin the triangulation process. This feature is nice for debugging output.)

- `obj.BoundingTriangulationOn ()` - Boolean controls whether bounding triangulation points (and associated triangles) are included in the output. (These are introduced as an initial triangulation to begin the triangulation process. This feature is nice for debugging output.)

- `obj.BoundingTriangulationOff ()` - Boolean controls whether bounding triangulation points (and associated triangles) are included in the output. (These are introduced as an initial triangulation to begin the triangulation process. This feature is nice for debugging output.)

- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default, an instance of vtkPointLocator is used.

- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default, an instance of vtkPointLocator is used.

- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to eliminate "coincident" points.

- `obj.InsertPoint (vtkUnstructuredGrid Mesh, vtkPoints points, vtkIdType id, double x[3], vtkIdList h` - This is a helper method used with InitPointInsertion() to create tetrahedronalizations of points. Its purpose is to inject point at coordinates specified into tetrahedronalization. The point id is an index into the list of points in the mesh structure. (See vtkDelaunay3D::InitPointInsertion() for more information.) When you have completed inserting points, traverse the mesh structure to extract desired tetrahedra (or tetra faces and edges).The holeTetras id list lists all the tetrahedra that are deleted (invalid) in the mesh structure.
- `obj.EndPointInsertion ()` - Invoke this method after all points have been inserted. The purpose of the method is to clean up internal data structures. Note that the (vtkUnstructuredGrid *)Mesh returned from InitPointInsertion() is NOT deleted, you still are responsible for cleaning that up.
- `long = obj.GetMTime ()` - Return the MTime also considering the locator.

33.57 vtkDensifyPolyData

33.57.1 Usage

The filter takes any polygonal data as input and will tessellate cells that are planar polygons present by fanning out triangles from its centroid. Other cells are simply passed through to the output. PointData, if present, is interpolated via linear interpolation. CellData for any tessellated cell is simply copied over from its parent cell. Planar polygons are assumed to be convex. Funny things will happen if they are not.

The number of subdivisions can be controlled by the parameter NumberOfSubdivisions.

To create an instance of class vtkDensifyPolyData, simply invoke its constructor as follows

```
obj = vtkDensifyPolyData
```

33.57.2 Methods

The class vtkDensifyPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDensifyPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDensifyPolyData = obj.NewInstance ()`
- `vtkDensifyPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfSubdivisions (int )` - Number of recursive subdivisions. Initial value is 1.
- `int = obj.GetNumberOfSubdivisions ()` - Number of recursive subdivisions. Initial value is 1.

33.58 vtkDicer

33.58.1 Usage

Subclasses of vtkDicer divides the input dataset into separate pieces. These pieces can then be operated on by other filters (e.g., vtkThreshold). One application is to break very large polygonal models into pieces and performing viewing and occlusion culling on the pieces. Multiple pieces can also be streamed through the visualization pipeline.

To use this filter, you must specify the execution mode of the filter; i.e., set the way that the piece size is controlled (do this by setting the DiceMode ivar). The filter does not change the geometry or topology of the input dataset, rather it generates integer numbers that indicate which piece a particular point belongs

to (i.e., it modifies the point and cell attribute data). The integer number can be placed into the output scalar data, or the output field data.

To create an instance of class vtkDicer, simply invoke its constructor as follows

```
obj = vtkDicer
```

33.58.2 Methods

The class vtkDicer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDicer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDicer = obj.NewInstance ()`
- `vtkDicer = obj.SafeDownCast (vtkObject o)`
- `obj.SetFieldData (int )` - Set/Get the flag which controls whether to generate point scalar data or point field data. If this flag is off, scalar data is generated. Otherwise, field data is generated. Note that the generated the data are integer numbers indicating which piece a particular point belongs to.
- `int = obj.GetFieldData ()` - Set/Get the flag which controls whether to generate point scalar data or point field data. If this flag is off, scalar data is generated. Otherwise, field data is generated. Note that the generated the data are integer numbers indicating which piece a particular point belongs to.
- `obj.FieldDataOn ()` - Set/Get the flag which controls whether to generate point scalar data or point field data. If this flag is off, scalar data is generated. Otherwise, field data is generated. Note that the generated the data are integer numbers indicating which piece a particular point belongs to.
- `obj.FieldDataOff ()` - Set/Get the flag which controls whether to generate point scalar data or point field data. If this flag is off, scalar data is generated. Otherwise, field data is generated. Note that the generated the data are integer numbers indicating which piece a particular point belongs to.
- `obj.SetDiceMode (int )` - Specify the method to determine how many pieces the data should be broken into. By default, the number of points per piece is used.
- `int = obj.GetDiceModeMinValue ()` - Specify the method to determine how many pieces the data should be broken into. By default, the number of points per piece is used.
- `int = obj.GetDiceModeMaxValue ()` - Specify the method to determine how many pieces the data should be broken into. By default, the number of points per piece is used.
- `int = obj.GetDiceMode ()` - Specify the method to determine how many pieces the data should be broken into. By default, the number of points per piece is used.
- `obj.SetDiceModeToNumberOfPointsPerPiece ()` - Specify the method to determine how many pieces the data should be broken into. By default, the number of points per piece is used.
- `obj.SetDiceModeToSpecifiedNumberOfPieces ()` - Specify the method to determine how many pieces the data should be broken into. By default, the number of points per piece is used.
- `obj.SetDiceModeToMemoryLimitPerPiece ()` - Specify the method to determine how many pieces the data should be broken into. By default, the number of points per piece is used.
- `int = obj.GetNumberOfActualPieces ()` - Use the following method after the filter has updated to determine the actual number of pieces the data was separated into.

- `obj.SetNumberOfPointsPerPiece (int )` - Control piece size based on the maximum number of points per piece. (This ivar has effect only when the DiceMode is set to SetDiceModeToNumberOfPoints().)
- `int = obj.GetNumberOfPointsPerPieceMinValue ()` - Control piece size based on the maximum number of points per piece. (This ivar has effect only when the DiceMode is set to SetDiceModeToNumberOfPoints().)
- `int = obj.GetNumberOfPointsPerPieceMaxValue ()` - Control piece size based on the maximum number of points per piece. (This ivar has effect only when the DiceMode is set to SetDiceModeToNumberOfPoints().)
- `int = obj.GetNumberOfPointsPerPiece ()` - Control piece size based on the maximum number of points per piece. (This ivar has effect only when the DiceMode is set to SetDiceModeToNumberOfPoints().)
- `obj.SetNumberOfPieces (int )` - Set/Get the number of pieces the object is to be separated into. (This ivar has effect only when the DiceMode is set to SetDiceModeToSpecifiedNumber()). Note that the ivar NumberOfPieces is a target - depending on the particulars of the data, more or less number of pieces than the target value may be created.
- `int = obj.GetNumberOfPiecesMinValue ()` - Set/Get the number of pieces the object is to be separated into. (This ivar has effect only when the DiceMode is set to SetDiceModeToSpecifiedNumber()). Note that the ivar NumberOfPieces is a target - depending on the particulars of the data, more or less number of pieces than the target value may be created.
- `int = obj.GetNumberOfPiecesMaxValue ()` - Set/Get the number of pieces the object is to be separated into. (This ivar has effect only when the DiceMode is set to SetDiceModeToSpecifiedNumber()). Note that the ivar NumberOfPieces is a target - depending on the particulars of the data, more or less number of pieces than the target value may be created.
- `int = obj.GetNumberOfPieces ()` - Set/Get the number of pieces the object is to be separated into. (This ivar has effect only when the DiceMode is set to SetDiceModeToSpecifiedNumber()). Note that the ivar NumberOfPieces is a target - depending on the particulars of the data, more or less number of pieces than the target value may be created.
- `obj.SetMemoryLimit (long )` - Control piece size based on a memory limit. (This ivar has effect only when the DiceMode is set to SetDiceModeToMemoryLimit()). The memory limit should be set in kilobytes.
- `GetMemoryLimitMinValue = obj.()` - Control piece size based on a memory limit. (This ivar has effect only when the DiceMode is set to SetDiceModeToMemoryLimit()). The memory limit should be set in kilobytes.
- `GetMemoryLimitMaxValue = obj.()` - Control piece size based on a memory limit. (This ivar has effect only when the DiceMode is set to SetDiceModeToMemoryLimit()). The memory limit should be set in kilobytes.
- `long = obj.GetMemoryLimit ()` - Control piece size based on a memory limit. (This ivar has effect only when the DiceMode is set to SetDiceModeToMemoryLimit()). The memory limit should be set in kilobytes.

33.59 vtkDijkstraGraphGeodesicPath

33.59.1 Usage

Takes as input a polygonal mesh and performs a single source shortest path calculation. Dijkstra's algorithm is used. The implementation is similar to the one described in Introduction to Algorithms (Second Edition)

by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein, published by MIT Press and McGraw-Hill. Some minor enhancement are added though. All vertices are not pushed on the heap at start, instead a front set is maintained. The heap is implemented as a binary heap. The output of the filter is a set of lines describing the shortest path from StartVertex to EndVertex.

To create an instance of class vtkDijkstraGraphGeodesicPath, simply invoke its constructor as follows

```
obj = vtkDijkstraGraphGeodesicPath
```

33.59.2 Methods

The class vtkDijkstraGraphGeodesicPath has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDijkstraGraphGeodesicPath class.

- `string = obj.GetClassName ()` - Standard methids for printing and determining type information.
- `int = obj.IsA (string name)` - Standard methids for printing and determining type information.
- `vtkDijkstraGraphGeodesicPath = obj.NewInstance ()` - Standard methids for printing and determining type information.
- `vtkDijkstraGraphGeodesicPath = obj.SafeDownCast (vtkObject o)` - Standard methids for printing and determining type information.
- `vtkIdList = obj.GetIdList ()` - The vertex ids (of the input polydata) on the shortest path
- `obj.SetStopWhenEndReached (int )` - Stop when the end vertex is reached or calculate shortest path to all vertices
- `int = obj.GetStopWhenEndReached ()` - Stop when the end vertex is reached or calculate shortest path to all vertices
- `obj.StopWhenEndReachedOn ()` - Stop when the end vertex is reached or calculate shortest path to all vertices
- `obj.StopWhenEndReachedOff ()` - Stop when the end vertex is reached or calculate shortest path to all vertices
- `obj.SetUseScalarWeights (int )` - Use scalar values in the edge weight (experimental)
- `int = obj.GetUseScalarWeights ()` - Use scalar values in the edge weight (experimental)
- `obj.UseScalarWeightsOn ()` - Use scalar values in the edge weight (experimental)
- `obj.UseScalarWeightsOff ()` - Use scalar values in the edge weight (experimental)
- `obj.SetRepelPathFromVertices (int )` - Use the input point to repel the path by assigning high costs.
- `int = obj.GetRepelPathFromVertices ()` - Use the input point to repel the path by assigning high costs.
- `obj.RepelPathFromVerticesOn ()` - Use the input point to repel the path by assigning high costs.
- `obj.RepelPathFromVerticesOff ()` - Use the input point to repel the path by assigning high costs.
- `obj.SetRepelVertices (vtkPoints )` - Specify vtkPoints to use to repel the path from.
- `vtkPoints = obj.GetRepelVertices ()` - Specify vtkPoints to use to repel the path from.
- `double = obj.GetGeodesicLength ()`
- `obj.GetCumulativeWeights (vtkDoubleArray weights)`

33.60 vtkDijkstraImageGeodesicPath

33.60.1 Usage

Takes as input a polyline and an image representing a 2D cost function and performs a single source shortest path calculation. Dijkstra's algorithm is used. The implementation is similar to the one described in Introduction to Algorithms (Second Edition) by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein, published by MIT Press and McGraw-Hill. Some minor enhancement are added though. All vertices are not pushed on the heap at start, instead a front set is maintained. The heap is implemented as a binary heap. The output of the filter is a set of lines describing the shortest path from StartVertex to EndVertex. See parent class vtkDijkstraGraphGeodesicPath for the implementation.

To create an instance of class vtkDijkstraImageGeodesicPath, simply invoke its constructor as follows

```
obj = vtkDijkstraImageGeodesicPath
```

33.60.2 Methods

The class vtkDijkstraImageGeodesicPath has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDijkstraImageGeodesicPath class.

- `string = obj.GetClassName ()` - Standard methids for printing and determining type information.
- `int = obj.IsA (string name)` - Standard methids for printing and determining type information.
- `vtkDijkstraImageGeodesicPath = obj.NewInstance ()` - Standard methids for printing and determining type information.
- `vtkDijkstraImageGeodesicPath = obj.SafeDownCast (vtkObject o)` - Standard methids for printing and determining type information.
- `obj.SetInput (vtkDataObject )` - Specify the image object which is used as a cost function.
- `vtkImageData = obj.GetInputAsImageData ()` - Specify the image object which is used as a cost function.
- `obj.SetImageWeight (double )` - Image cost weight.
- `double = obj.GetImageWeight ()` - Image cost weight.
- `obj.SetEdgeLengthWeight (double )` - Edge length cost weight.
- `double = obj.GetEdgeLengthWeight ()` - Edge length cost weight.
- `obj.SetCurvatureWeight (double )` - Curvature cost weight.
- `double = obj.GetCurvatureWeightMinValue ()` - Curvature cost weight.
- `double = obj.GetCurvatureWeightMaxValue ()` - Curvature cost weight.
- `double = obj.GetCurvatureWeight ()` - Curvature cost weight.

33.61 vtkDiscreteMarchingCubes

33.61.1 Usage

takes as input a volume (e.g., 3D structured point set) of segmentation labels and generates on output one or more models representing the boundaries between the specified label and the adjacent structures. One or more label values must be specified to generate the models. The boundary positions are always defined to be half-way between adjacent voxels. This filter works best with integral scalar values. If ComputeScalars is on (the default), each output cell will have cell data that corresponds to the scalar value (segmentation label) of the corresponding cube. Note that this differs from vtkMarchingCubes, which stores the scalar value as point data. The rationale for this difference is that cell vertices may be shared between multiple cells. This also means that the resultant polydata may be non-manifold (cell faces may be coincident). To further process the polydata, users should either: 1) extract cells that have a common scalar value using vtkThreshold, or 2) process the data with filters that can handle non-manifold polydata (e.g. vtkWindowedSincPolyDataFilter). Also note, Normals and Gradients are not computed.

To create an instance of class vtkDiscreteMarchingCubes, simply invoke its constructor as follows

```
obj = vtkDiscreteMarchingCubes
```

33.61.2 Methods

The class vtkDiscreteMarchingCubes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDiscreteMarchingCubes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDiscreteMarchingCubes = obj.NewInstance ()`
- `vtkDiscreteMarchingCubes = obj.SafeDownCast (vtkObject o)`

33.62 vtkDiskSource

33.62.1 Usage

vtkDiskSource creates a polygonal disk with a hole in the center. The disk has zero height. The user can specify the inner and outer radius of the disk, and the radial and circumferential resolution of the polygonal representation.

To create an instance of class vtkDiskSource, simply invoke its constructor as follows

```
obj = vtkDiskSource
```

33.62.2 Methods

The class vtkDiskSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDiskSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDiskSource = obj.NewInstance ()`

- `vtkDiskSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetInnerRadius (double )` - Specify inner radius of hole in disc.
- `double = obj.GetInnerRadiusMinValue ()` - Specify inner radius of hole in disc.
- `double = obj.GetInnerRadiusMaxValue ()` - Specify inner radius of hole in disc.
- `double = obj.GetInnerRadius ()` - Specify inner radius of hole in disc.
- `obj.SetOuterRadius (double )` - Specify outer radius of disc.
- `double = obj.GetOuterRadiusMinValue ()` - Specify outer radius of disc.
- `double = obj.GetOuterRadiusMaxValue ()` - Specify outer radius of disc.
- `double = obj.GetOuterRadius ()` - Specify outer radius of disc.
- `obj.SetRadialResolution (int )` - Set the number of points in radius direction.
- `int = obj.GetRadialResolutionMinValue ()` - Set the number of points in radius direction.
- `int = obj.GetRadialResolutionMaxValue ()` - Set the number of points in radius direction.
- `int = obj.GetRadialResolution ()` - Set the number of points in radius direction.
- `obj.SetCircumferentialResolution (int )` - Set the number of points in circumferential direction.
- `int = obj.GetCircumferentialResolutionMinValue ()` - Set the number of points in circumferential direction.
- `int = obj.GetCircumferentialResolutionMaxValue ()` - Set the number of points in circumferential direction.
- `int = obj.GetCircumferentialResolution ()` - Set the number of points in circumferential direction.

33.63 vtkEdgePoints

33.63.1 Usage

vtkEdgePoints is a filter that takes as input any dataset and generates for output a set of points that lie on an isosurface. The points are created by interpolation along cells edges whose end-points are below and above the contour value.

To create an instance of class vtkEdgePoints, simply invoke its constructor as follows

```
obj = vtkEdgePoints
```

33.63.2 Methods

The class vtkEdgePoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEdgePoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEdgePoints = obj.NewInstance ()`

- `vtkEdgePoints = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (double )` - Set/get the contour value.
- `double = obj.GetValue ()` - Set/get the contour value.

33.64 vtkEdgeSubdivisionCriterion

33.64.1 Usage

Descendants of this abstract class are used to decide whether a piecewise linear approximation (triangles, lines, ...) to some nonlinear geometry should be subdivided. This decision may be based on an absolute error metric (chord error) or on some view-dependent metric (chord error compared to device resolution) or on some abstract metric (color error). Or anything else, really. Just so long as you implement the EvaluateEdge member, all will be well.

To create an instance of class vtkEdgeSubdivisionCriterion, simply invoke its constructor as follows

```
obj = vtkEdgeSubdivisionCriterion
```

33.64.2 Methods

The class vtkEdgeSubdivisionCriterion has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEdgeSubdivisionCriterion class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEdgeSubdivisionCriterion = obj.NewInstance ()`
- `vtkEdgeSubdivisionCriterion = obj.SafeDownCast (vtkObject o)`
- `bool = obj.EvaluateEdge (double p0, double p1, double p2, int field\_start)` - You must implement this member function in a subclass. It will be called by vtkStreamingTessellator for each edge in each primitive that vtkStreamingTessellator generates.
- `int = obj.PassField (int sourceId, int sourceSize, vtkStreamingTessellator t)` - This is a helper routine called by PassFields() which you may also call directly; it adds sourceSize to the size of the output vertex field values. The offset of the sourceId field in the output vertex array is returned. -1 is returned if sourceSize would force the output to have more than vtkStreamingTessellator::MaxFieldSize field values per vertex.
- `obj.ResetFieldList ()` - Don't pass any field values in the vertex pointer. This is used to reset the list of fields to pass after a successful run of vtkStreamingTessellator.
- `bool = obj.DontPassField (int sourceId, vtkStreamingTessellator t)` - This does the opposite of PassField(); it removes a field from the output (assuming the field was set to be passed). Returns true if any action was taken, false otherwise.
- `int = obj.GetOutputField (int fieldId) const` - Return the output ID of an input field. Returns -1 if fieldId is not set to be passed to the output.
- `int = obj.GetNumberOfFields () const` - Return the number of fields being evaluated at each output vertex. This is the length of the arrays returned by GetFieldIds() and GetFieldOffsets().

33.65 vtkElevationFilter

33.65.1 Usage

vtkElevationFilter is a filter to generate scalar values from a dataset. The scalar values lie within a user specified range, and are generated by computing a projection of each dataset point onto a line. The line can be oriented arbitrarily. A typical example is to generate scalars based on elevation or height above a plane.

To create an instance of class vtkElevationFilter, simply invoke its constructor as follows

```
obj = vtkElevationFilter
```

33.65.2 Methods

The class vtkElevationFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkElevationFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkElevationFilter = obj.NewInstance ()`
- `vtkElevationFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetLowPoint (double , double , double )` - Define one end of the line (small scalar values). Default is (0,0,0).
- `obj.SetLowPoint (double  a[3])` - Define one end of the line (small scalar values). Default is (0,0,0).
- `double = obj. GetLowPoint ()` - Define one end of the line (small scalar values). Default is (0,0,0).
- `obj.SetHighPoint (double , double , double )` - Define other end of the line (large scalar values). Default is (0,0,1).
- `obj.SetHighPoint (double  a[3])` - Define other end of the line (large scalar values). Default is (0,0,1).
- `double = obj. GetHighPoint ()` - Define other end of the line (large scalar values). Default is (0,0,1).
- `obj.SetScalarRange (double , double )` - Specify range to map scalars into. Default is [0, 1].
- `obj.SetScalarRange (double  a[2])` - Specify range to map scalars into. Default is [0, 1].
- `double = obj. GetScalarRange ()` - Specify range to map scalars into. Default is [0, 1].

33.66 vtkEllipticalButtonSource

33.66.1 Usage

vtkEllipticalButtonSource creates a ellipsoidal shaped button with texture coordinates suitable for application of a texture map. This provides a way to make nice looking 3D buttons. The buttons are represented as vtkPolyData that includes texture coordinates and normals. The button lies in the x-y plane.

To use this class you must define the major and minor axes lengths of an ellipsoid (expressed as width (x), height (y) and depth (z)). The button has a rectangular mesh region in the center with texture coordinates that range smoothly from (0,1). (This flat region is called the texture region.) The outer, curved portion of the button (called the shoulder) has texture coordinates set to a user specified value (by default (0,0). (This results in coloring the button curve the same color as the (s,t) location of the texture map.) The resolution

in the radial direction, the texture region, and the shoulder region must also be set. The button can be moved by specifying an origin.

To create an instance of class vtkEllipticalButtonSource, simply invoke its constructor as follows

```
obj = vtkEllipticalButtonSource
```

33.66.2 Methods

The class vtkEllipticalButtonSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEllipticalButtonSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEllipticalButtonSource = obj.NewInstance ()`
- `vtkEllipticalButtonSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetWidth (double )` - Set/Get the width of the button (the x-ellipsoid axis length * 2).
- `double = obj.GetWidthMinValue ()` - Set/Get the width of the button (the x-ellipsoid axis length * 2).
- `double = obj.GetWidthMaxValue ()` - Set/Get the width of the button (the x-ellipsoid axis length * 2).
- `double = obj.GetWidth ()` - Set/Get the width of the button (the x-ellipsoid axis length * 2).
- `obj.SetHeight (double )` - Set/Get the height of the button (the y-ellipsoid axis length * 2).
- `double = obj.GetHeightMinValue ()` - Set/Get the height of the button (the y-ellipsoid axis length * 2).
- `double = obj.GetHeightMaxValue ()` - Set/Get the height of the button (the y-ellipsoid axis length * 2).
- `double = obj.GetHeight ()` - Set/Get the height of the button (the y-ellipsoid axis length * 2).
- `obj.SetDepth (double )` - Set/Get the depth of the button (the z-eliipsoid axis length).
- `double = obj.GetDepthMinValue ()` - Set/Get the depth of the button (the z-eliipsoid axis length).
- `double = obj.GetDepthMaxValue ()` - Set/Get the depth of the button (the z-eliipsoid axis length).
- `double = obj.GetDepth ()` - Set/Get the depth of the button (the z-eliipsoid axis length).
- `obj.SetCircumferentialResolution (int )` - Specify the resolution of the button in the circumferential direction.
- `int = obj.GetCircumferentialResolutionMinValue ()` - Specify the resolution of the button in the circumferential direction.
- `int = obj.GetCircumferentialResolutionMaxValue ()` - Specify the resolution of the button in the circumferential direction.
- `int = obj.GetCircumferentialResolution ()` - Specify the resolution of the button in the circumferential direction.

- `obj.SetTextureResolution (int )` - Specify the resolution of the texture in the radial direction in the texture region.
- `int = obj.GetTextureResolutionMinValue ()` - Specify the resolution of the texture in the radial direction in the texture region.
- `int = obj.GetTextureResolutionMaxValue ()` - Specify the resolution of the texture in the radial direction in the texture region.
- `int = obj.GetTextureResolution ()` - Specify the resolution of the texture in the radial direction in the texture region.
- `obj.SetShoulderResolution (int )` - Specify the resolution of the texture in the radial direction in the shoulder region.
- `int = obj.GetShoulderResolutionMinValue ()` - Specify the resolution of the texture in the radial direction in the shoulder region.
- `int = obj.GetShoulderResolutionMaxValue ()` - Specify the resolution of the texture in the radial direction in the shoulder region.
- `int = obj.GetShoulderResolution ()` - Specify the resolution of the texture in the radial direction in the shoulder region.
- `obj.SetRadialRatio (double )` - Set/Get the radial ratio. This is the measure of the radius of the outer ellipsoid to the inner ellipsoid of the button. The outer ellipsoid is the boundary of the button defined by the height and width. The inner ellipsoid circumscribes the texture region. Larger RadialRatio's cause the button to be more rounded (and the texture region to be smaller); smaller ratios produce sharply curved shoulders with a larger texture region.
- `double = obj.GetRadialRatioMinValue ()` - Set/Get the radial ratio. This is the measure of the radius of the outer ellipsoid to the inner ellipsoid of the button. The outer ellipsoid is the boundary of the button defined by the height and width. The inner ellipsoid circumscribes the texture region. Larger RadialRatio's cause the button to be more rounded (and the texture region to be smaller); smaller ratios produce sharply curved shoulders with a larger texture region.
- `double = obj.GetRadialRatioMaxValue ()` - Set/Get the radial ratio. This is the measure of the radius of the outer ellipsoid to the inner ellipsoid of the button. The outer ellipsoid is the boundary of the button defined by the height and width. The inner ellipsoid circumscribes the texture region. Larger RadialRatio's cause the button to be more rounded (and the texture region to be smaller); smaller ratios produce sharply curved shoulders with a larger texture region.
- `double = obj.GetRadialRatio ()` - Set/Get the radial ratio. This is the measure of the radius of the outer ellipsoid to the inner ellipsoid of the button. The outer ellipsoid is the boundary of the button defined by the height and width. The inner ellipsoid circumscribes the texture region. Larger RadialRatio's cause the button to be more rounded (and the texture region to be smaller); smaller ratios produce sharply curved shoulders with a larger texture region.

33.67 vtkExtractArraysOverTime

33.67.1 Usage

vtkExtractArraysOverTime extracts a selection over time. The output is a multiblock dataset. If selection content type is vtkSelection::Locations, then each output block corresponds to each probed location. Otherwise, each output block corresponds to an extracted cell/point depending on whether the selection field type is CELL or POINT. Each block is a vtkTable with a column named Time (or TimeData if Time exists in the input). When extracting point data, the input point coordinates are copied to a column named Point Coordinates or Points (if Point Coordinates exists in the input). This algorithm does not produce a

TIME_STEPS or TIME_RANGE information because it works across time. .Section Caveat This algorithm works only with source that produce TIME_STEPS(). Continuous time range is not yet supported.

To create an instance of class vtkExtractArraysOverTime, simply invoke its constructor as follows

```
obj = vtkExtractArraysOverTime
```

33.67.2 Methods

The class vtkExtractArraysOverTime has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractArraysOverTime class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractArraysOverTime = obj.NewInstance ()`
- `vtkExtractArraysOverTime = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfTimeSteps ()` - Get the number of time steps
- `obj.SetSelectionConnection (vtkAlgorithmOutput algOutput)`

33.68 vtkExtractBlock

33.68.1 Usage

vtkExtractBlock is a filter that extracts blocks from a multiblock dataset. Each node in the multi-block tree is identified by an įndex. The index can be obtained by performing a preorder traversal of the tree (including empty nodes). eg. A(B (D, E), C(F, G)). Inorder traversal yields: A, B, D, E, C, F, G Index of A is 0, while index of C is 4.

To create an instance of class vtkExtractBlock, simply invoke its constructor as follows

```
obj = vtkExtractBlock
```

33.68.2 Methods

The class vtkExtractBlock has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractBlock class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractBlock = obj.NewInstance ()`
- `vtkExtractBlock = obj.SafeDownCast (vtkObject o)`
- `obj.AddIndex (int index)` - Select the block indices to extract. Each node in the multi-block tree is identified by an įndex. The index can be obtained by performing a preorder traversal of the tree (including empty nodes). eg. A(B (D, E), C(F, G)). Inorder traversal yields: A, B, D, E, C, F, G Index of A is 0, while index of C is 4.

- `obj.RemoveIndex (int index)` - Select the block indices to extract. Each node in the multi-block tree is identified by an ţndex. The index can be obtained by performing a preorder traversal of the tree (including empty nodes). eg. A(B (D, E), C(F, G)). Inorder traversal yields: A, B, D, E, C, F, G Index of A is 0, while index of C is 4.

- `obj.RemoveAllIndices ()` - Select the block indices to extract. Each node in the multi-block tree is identified by an ţndex. The index can be obtained by performing a preorder traversal of the tree (including empty nodes). eg. A(B (D, E), C(F, G)). Inorder traversal yields: A, B, D, E, C, F, G Index of A is 0, while index of C is 4.

- `obj.SetPruneOutput (int )` - When set, the output mutliblock dataset will be pruned to remove empty nodes. On by default.

- `int = obj.GetPruneOutput ()` - When set, the output mutliblock dataset will be pruned to remove empty nodes. On by default.

- `obj.PruneOutputOn ()` - When set, the output mutliblock dataset will be pruned to remove empty nodes. On by default.

- `obj.PruneOutputOff ()` - When set, the output mutliblock dataset will be pruned to remove empty nodes. On by default.

- `obj.SetMaintainStructure (int )` - This is used only when PruneOutput is ON. By default, when pruning the output i.e. remove empty blocks, if node has only 1 non-null child block, then that node is removed. To preserve these parent nodes, set this flag to true. Off by default.

- `int = obj.GetMaintainStructure ()` - This is used only when PruneOutput is ON. By default, when pruning the output i.e. remove empty blocks, if node has only 1 non-null child block, then that node is removed. To preserve these parent nodes, set this flag to true. Off by default.

- `obj.MaintainStructureOn ()` - This is used only when PruneOutput is ON. By default, when pruning the output i.e. remove empty blocks, if node has only 1 non-null child block, then that node is removed. To preserve these parent nodes, set this flag to true. Off by default.

- `obj.MaintainStructureOff ()` - This is used only when PruneOutput is ON. By default, when pruning the output i.e. remove empty blocks, if node has only 1 non-null child block, then that node is removed. To preserve these parent nodes, set this flag to true. Off by default.

33.69 vtkExtractCells

33.69.1 Usage

Given a vtkDataSet and a list of cell Ids, create a vtkUnstructuredGrid composed of these cells. If the cell list is empty when vtkExtractCells executes, it will set up the ugrid, point and cell arrays, with no points, cells or data.

To create an instance of class vtkExtractCells, simply invoke its constructor as follows

```
obj = vtkExtractCells
```

33.69.2 Methods

The class vtkExtractCells has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractCells class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkExtractCells = obj.NewInstance ()`
- `vtkExtractCells = obj.SafeDownCast (vtkObject o)`
- `obj.SetCellList (vtkIdList l)`
- `obj.AddCellList (vtkIdList l)`
- `obj.AddCellRange (vtkIdType from, vtkIdType to)`

33.70 vtkExtractDataOverTime

33.70.1 Usage

This filter extracts the point data from a time sequence and specified index and creates an output of the same type as the input but with Points containing "number of time steps" points; the point and PointData corresponding to the PointIndex are extracted at each time step and added to the output. A PointData array is added called "Time" (or "TimeData" if there is already an array called "Time"), which is the time at each index.

To create an instance of class vtkExtractDataOverTime, simply invoke its constructor as follows

```
obj = vtkExtractDataOverTime
```

33.70.2 Methods

The class vtkExtractDataOverTime has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractDataOverTime class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractDataOverTime = obj.NewInstance ()`
- `vtkExtractDataOverTime = obj.SafeDownCast (vtkObject o)`
- `obj.SetPointIndex (int )` - Index of point to extract at each time step
- `int = obj.GetPointIndex ()` - Index of point to extract at each time step
- `int = obj.GetNumberOfTimeSteps ()` - Get the number of time steps

33.71 vtkExtractDataSets

33.71.1 Usage

vtkExtractDataSets accepts a vtkHierarchicalBoxDataSet as input and extracts different datasets from different levels. The output is vtkHierarchicalBoxDataSet with same structure as the input with only the selected datasets passed through.

To create an instance of class vtkExtractDataSets, simply invoke its constructor as follows

```
obj = vtkExtractDataSets
```

33.71.2 Methods

The class vtkExtractDataSets has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractDataSets class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractDataSets = obj.NewInstance ()`
- `vtkExtractDataSets = obj.SafeDownCast (vtkObject o)`
- `obj.AddDataSet (int level, int idx)` - Add a dataset to be extracted.
- `obj.ClearDataSetList ()` - Remove all entries from the list of datasets to be extracted.

33.72 vtkExtractEdges

33.72.1 Usage

vtkExtractEdges is a filter to extract edges from a dataset. Edges are extracted as lines or polylines.

To create an instance of class vtkExtractEdges, simply invoke its constructor as follows

```
obj = vtkExtractEdges
```

33.72.2 Methods

The class vtkExtractEdges has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractEdges class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractEdges = obj.NewInstance ()`
- `vtkExtractEdges = obj.SafeDownCast (vtkObject o)`
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified.
- `long = obj.GetMTime ()` - Return MTime also considering the locator.

33.73 vtkExtractGeometry

33.73.1 Usage

vtkExtractGeometry extracts from its input dataset all cells that are either completely inside or outside of a specified implicit function. Any type of dataset can be input to this filter. On output the filter generates an unstructured grid.

To use this filter you must specify an implicit function. You must also specify whethter to extract cells lying inside or outside of the implicit function. (The inside of an implicit function is the negative values region.) An option exists to extract cells that are neither inside or outside (i.e., boundary).

A more efficient version of this filter is available for vtkPolyData input. See vtkExtractPolyDataGeometry.

To create an instance of class vtkExtractGeometry, simply invoke its constructor as follows

```
obj = vtkExtractGeometry
```

33.73.2 Methods

The class vtkExtractGeometry has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractGeometry class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractGeometry = obj.NewInstance ()`
- `vtkExtractGeometry = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Return the MTime taking into account changes to the implicit function
- `obj.SetImplicitFunction (vtkImplicitFunction )` - Specify the implicit function for inside/outside checks.
- `vtkImplicitFunction = obj.GetImplicitFunction ()` - Specify the implicit function for inside/outside checks.
- `obj.SetExtractInside (int )` - Boolean controls whether to extract cells that are inside of implicit function (ExtractInside == 1) or outside of implicit function (ExtractInside == 0).
- `int = obj.GetExtractInside ()` - Boolean controls whether to extract cells that are inside of implicit function (ExtractInside == 1) or outside of implicit function (ExtractInside == 0).
- `obj.ExtractInsideOn ()` - Boolean controls whether to extract cells that are inside of implicit function (ExtractInside == 1) or outside of implicit function (ExtractInside == 0).
- `obj.ExtractInsideOff ()` - Boolean controls whether to extract cells that are inside of implicit function (ExtractInside == 1) or outside of implicit function (ExtractInside == 0).
- `obj.SetExtractBoundaryCells (int )` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.
- `int = obj.GetExtractBoundaryCells ()` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.
- `obj.ExtractBoundaryCellsOn ()` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.

- `obj.ExtractBoundaryCellsOff ()` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.
- `obj.SetExtractOnlyBoundaryCells (int )` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.
- `int = obj.GetExtractOnlyBoundaryCells ()` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.
- `obj.ExtractOnlyBoundaryCellsOn ()` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.
- `obj.ExtractOnlyBoundaryCellsOff ()` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.

33.74 vtkExtractGrid

33.74.1 Usage

vtkExtractGrid is a filter that selects a portion of an input structured grid dataset, or subsamples an input dataset. (The selected portion of interested is referred to as the Volume Of Interest, or VOI.) The output of this filter is a structured grid dataset. The filter treats input data of any topological dimension (i.e., point, line, image, or volume) and can generate output data of any topological dimension.

To use this filter set the VOI ivar which are i-j-k min/max indices that specify a rectangular region in the data. (Note that these are 0-offset.) You can also specify a sampling rate to subsample the data.

Typical applications of this filter are to extract a plane from a grid for contouring, subsampling large grids to reduce data size, or extracting regions of a grid with interesting data.

To create an instance of class vtkExtractGrid, simply invoke its constructor as follows

```
obj = vtkExtractGrid
```

33.74.2 Methods

The class vtkExtractGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractGrid = obj.NewInstance ()`
- `vtkExtractGrid = obj.SafeDownCast (vtkObject o)`
- `obj.SetVOI (int , int , int , int , int , int )` - Specify i-j-k (min,max) pairs to extract. The resulting structured grid dataset can be of any topological dimension (i.e., point, line, plane, or 3D grid).
- `obj.SetVOI (int  a[6])` - Specify i-j-k (min,max) pairs to extract. The resulting structured grid dataset can be of any topological dimension (i.e., point, line, plane, or 3D grid).
- `int = obj. GetVOI ()` - Specify i-j-k (min,max) pairs to extract. The resulting structured grid dataset can be of any topological dimension (i.e., point, line, plane, or 3D grid).
- `obj.SetSampleRate (int , int , int )` - Set the sampling rate in the i, j, and k directions. If the rate is ¿ 1, then the resulting VOI will be subsampled representation of the input. For example, if the SampleRate=(2,2,2), every other point will be selected, resulting in a volume 1/8th the original size.

- `obj.SetSampleRate (int  a[3])` - Set the sampling rate in the i, j, and k directions. If the rate is ¿ 1, then the resulting VOI will be subsampled representation of the input. For example, if the SampleRate=(2,2,2), every other point will be selected, resulting in a volume 1/8th the original size.
- `int = obj. GetSampleRate ()` - Set the sampling rate in the i, j, and k directions. If the rate is ¿ 1, then the resulting VOI will be subsampled representation of the input. For example, if the SampleRate=(2,2,2), every other point will be selected, resulting in a volume 1/8th the original size.
- `obj.SetIncludeBoundary (int )` - Control whether to enforce that the "boundary" of the grid is output in the subsampling process. (This ivar only has effect when the SampleRate in any direction is not equal to 1.) When this ivar IncludeBoundary is on, the subsampling will always include the boundary of the grid even though the sample rate is not an even multiple of the grid dimensions. (By default IncludeBoundary is off.)
- `int = obj.GetIncludeBoundary ()` - Control whether to enforce that the "boundary" of the grid is output in the subsampling process. (This ivar only has effect when the SampleRate in any direction is not equal to 1.) When this ivar IncludeBoundary is on, the subsampling will always include the boundary of the grid even though the sample rate is not an even multiple of the grid dimensions. (By default IncludeBoundary is off.)
- `obj.IncludeBoundaryOn ()` - Control whether to enforce that the "boundary" of the grid is output in the subsampling process. (This ivar only has effect when the SampleRate in any direction is not equal to 1.) When this ivar IncludeBoundary is on, the subsampling will always include the boundary of the grid even though the sample rate is not an even multiple of the grid dimensions. (By default IncludeBoundary is off.)
- `obj.IncludeBoundaryOff ()` - Control whether to enforce that the "boundary" of the grid is output in the subsampling process. (This ivar only has effect when the SampleRate in any direction is not equal to 1.) When this ivar IncludeBoundary is on, the subsampling will always include the boundary of the grid even though the sample rate is not an even multiple of the grid dimensions. (By default IncludeBoundary is off.)

33.75 vtkExtractLevel

33.75.1 Usage

vtkExtractLevel filter extracts the levels between (and including) the user specified min and max levels.

To create an instance of class vtkExtractLevel, simply invoke its constructor as follows

```
obj = vtkExtractLevel
```

33.75.2 Methods

The class vtkExtractLevel has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractLevel class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractLevel = obj.NewInstance ()`
- `vtkExtractLevel = obj.SafeDownCast (vtkObject o)`
- `obj.AddLevel (int level)` - Select the levels that should be extracted. All other levels will have no datasets in them.

- `obj.RemoveLevel (int level)` - Select the levels that should be extracted. All other levels will have no datasets in them.
- `obj.RemoveAllLevels ()` - Select the levels that should be extracted. All other levels will have no datasets in them.

33.76 vtkExtractPolyDataGeometry

33.76.1 Usage

vtkExtractPolyDataGeometry extracts from its input vtkPolyData all cells that are either completely inside or outside of a specified implicit function. This filter is specialized to vtkPolyData. On output the filter generates vtkPolyData.

To use this filter you must specify an implicit function. You must also specify whether to extract cells lying inside or outside of the implicit function. (The inside of an implicit function is the negative values region.) An option exists to extract cells that are neither inside nor outside (i.e., boundary).

A more general version of this filter is available for arbitrary vtkDataSet input (see vtkExtractGeometry).

To create an instance of class vtkExtractPolyDataGeometry, simply invoke its constructor as follows

```
obj = vtkExtractPolyDataGeometry
```

33.76.2 Methods

The class vtkExtractPolyDataGeometry has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractPolyDataGeometry class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractPolyDataGeometry = obj.NewInstance ()`
- `vtkExtractPolyDataGeometry = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Return the MTime taking into account changes to the implicit function
- `obj.SetImplicitFunction (vtkImplicitFunction )` - Specify the implicit function for inside/outside checks.
- `vtkImplicitFunction = obj.GetImplicitFunction ()` - Specify the implicit function for inside/outside checks.
- `obj.SetExtractInside (int )` - Boolean controls whether to extract cells that are inside of implicit function (ExtractInside == 1) or outside of implicit function (ExtractInside == 0).
- `int = obj.GetExtractInside ()` - Boolean controls whether to extract cells that are inside of implicit function (ExtractInside == 1) or outside of implicit function (ExtractInside == 0).
- `obj.ExtractInsideOn ()` - Boolean controls whether to extract cells that are inside of implicit function (ExtractInside == 1) or outside of implicit function (ExtractInside == 0).
- `obj.ExtractInsideOff ()` - Boolean controls whether to extract cells that are inside of implicit function (ExtractInside == 1) or outside of implicit function (ExtractInside == 0).
- `obj.SetExtractBoundaryCells (int )` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.

- `int = obj.GetExtractBoundaryCells ()` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.
- `obj.ExtractBoundaryCellsOn ()` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.
- `obj.ExtractBoundaryCellsOff ()` - Boolean controls whether to extract cells that are partially inside. By default, ExtractBoundaryCells is off.

33.77 vtkExtractRectilinearGrid

33.77.1 Usage

vtkExtractRectilinearGrid rounds out the set of filters that extract a subgrid out of a larger structured data set. RIght now, this filter only supports extracting a VOI. In the future, it might support strides like the vtkExtract grid filter.

To create an instance of class vtkExtractRectilinearGrid, simply invoke its constructor as follows

```
obj = vtkExtractRectilinearGrid
```

33.77.2 Methods

The class vtkExtractRectilinearGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractRectilinearGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractRectilinearGrid = obj.NewInstance ()`
- `vtkExtractRectilinearGrid = obj.SafeDownCast (vtkObject o)`
- `obj.SetVOI (int , int , int , int , int , int )` - Specify i-j-k (min,max) pairs to extract. The resulting structured grid dataset can be of any topological dimension (i.e., point, line, plane, or 3D grid).
- `obj.SetVOI (int a[6])` - Specify i-j-k (min,max) pairs to extract. The resulting structured grid dataset can be of any topological dimension (i.e., point, line, plane, or 3D grid).
- `int = obj. GetVOI ()` - Specify i-j-k (min,max) pairs to extract. The resulting structured grid dataset can be of any topological dimension (i.e., point, line, plane, or 3D grid).
- `obj.SetSampleRate (int , int , int )` - Set the sampling rate in the i, j, and k directions. If the rate is ¿ 1, then the resulting VOI will be subsampled representation of the input. For example, if the SampleRate=(2,2,2), every other point will be selected, resulting in a volume 1/8th the original size.
- `obj.SetSampleRate (int a[3])` - Set the sampling rate in the i, j, and k directions. If the rate is ¿ 1, then the resulting VOI will be subsampled representation of the input. For example, if the SampleRate=(2,2,2), every other point will be selected, resulting in a volume 1/8th the original size.
- `int = obj. GetSampleRate ()` - Set the sampling rate in the i, j, and k directions. If the rate is ¿ 1, then the resulting VOI will be subsampled representation of the input. For example, if the SampleRate=(2,2,2), every other point will be selected, resulting in a volume 1/8th the original size.

- `obj.SetIncludeBoundary (int )` - Control whether to enforce that the "boundary" of the grid is output in the subsampling process. (This ivar only has effect when the SampleRate in any direction is not equal to 1.) When this ivar IncludeBoundary is on, the subsampling will always include the boundary of the grid even though the sample rate is not an even multiple of the grid dimensions. (By default IncludeBoundary is off.)
- `int = obj.GetIncludeBoundary ()` - Control whether to enforce that the "boundary" of the grid is output in the subsampling process. (This ivar only has effect when the SampleRate in any direction is not equal to 1.) When this ivar IncludeBoundary is on, the subsampling will always include the boundary of the grid even though the sample rate is not an even multiple of the grid dimensions. (By default IncludeBoundary is off.)
- `obj.IncludeBoundaryOn ()` - Control whether to enforce that the "boundary" of the grid is output in the subsampling process. (This ivar only has effect when the SampleRate in any direction is not equal to 1.) When this ivar IncludeBoundary is on, the subsampling will always include the boundary of the grid even though the sample rate is not an even multiple of the grid dimensions. (By default IncludeBoundary is off.)
- `obj.IncludeBoundaryOff ()` - Control whether to enforce that the "boundary" of the grid is output in the subsampling process. (This ivar only has effect when the SampleRate in any direction is not equal to 1.) When this ivar IncludeBoundary is on, the subsampling will always include the boundary of the grid even though the sample rate is not an even multiple of the grid dimensions. (By default IncludeBoundary is off.)

33.78 vtkExtractSelectedBlock

33.78.1 Usage

To create an instance of class vtkExtractSelectedBlock, simply invoke its constructor as follows

```
obj = vtkExtractSelectedBlock
```

33.78.2 Methods

The class vtkExtractSelectedBlock has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelectedBlock class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelectedBlock = obj.NewInstance ()`
- `vtkExtractSelectedBlock = obj.SafeDownCast (vtkObject o)`

33.79 vtkExtractSelectedFrustum

33.79.1 Usage

This class intersects the input DataSet with a frustum and determines which cells and points lie within the frustum. The frustum is defined with a vtkPlanes containing six cutting planes. The output is a DataSet that is either a shallow copy of the input dataset with two new "vtkInsidedness" attribute arrays, or a completely new UnstructuredGrid that contains only the cells and points of the input that are inside the frustum. The PreserveTopology flag controls which occurs. When PreserveTopology is off this filter adds

a scalar array called vtkOriginalCellIds that says what input cell produced each output cell. This is an example of a Pedigree ID which helps to trace back results.

To create an instance of class vtkExtractSelectedFrustum, simply invoke its constructor as follows

```
obj = vtkExtractSelectedFrustum
```

33.79.2 Methods

The class vtkExtractSelectedFrustum has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelectedFrustum class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelectedFrustum = obj.NewInstance ()`
- `vtkExtractSelectedFrustum = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Return the MTime taking into account changes to the Frustum
- `obj.SetFrustum (vtkPlanes )` - Set the selection frustum. The planes object must contain six planes.
- `vtkPlanes = obj.GetFrustum ()` - Set the selection frustum. The planes object must contain six planes.
- `obj.CreateFrustum (double vertices[32])` - Given eight vertices, creates a frustum. each pt is x,y,z,1 in the following order near lower left, far lower left near upper left, far upper left near lower right, far lower right near upper right, far upper right
- `vtkPoints = obj.GetClipPoints ()` - Return eight points that define the selection frustum. Valid if create Frustum was used, invalid if SetFrustum was.
- `obj.SetFieldType (int )` - Sets/gets the intersection test type.
- `int = obj.GetFieldType ()` - Sets/gets the intersection test type.
- `obj.SetContainingCells (int )` - Sets/gets the intersection test type. Only meaningful when fieldType is vtkSelection::POINT
- `int = obj.GetContainingCells ()` - Sets/gets the intersection test type. Only meaningful when fieldType is vtkSelection::POINT
- `int = obj.OverallBoundsTest (double bounds)` - Does a quick test on the AABBox defined by the bounds.
- `obj.SetShowBounds (int )` - When On, this returns an unstructured grid that outlines selection area. Off is the default.
- `int = obj.GetShowBounds ()` - When On, this returns an unstructured grid that outlines selection area. Off is the default.
- `obj.ShowBoundsOn ()` - When On, this returns an unstructured grid that outlines selection area. Off is the default.
- `obj.ShowBoundsOff ()` - When On, this returns an unstructured grid that outlines selection area. Off is the default.
- `obj.SetInsideOut (int )` - When on, extracts cells outside the frustum instead of inside.

- `int = obj.GetInsideOut ()` - When on, extracts cells outside the frustum instead of inside.
- `obj.InsideOutOn ()` - When on, extracts cells outside the frustum instead of inside.
- `obj.InsideOutOff ()` - When on, extracts cells outside the frustum instead of inside.

33.80 vtkExtractSelectedIds

33.80.1 Usage

vtkExtractSelectedIds extracts a set of cells and points from within a vtkDataSet. The set of ids to extract are listed within a vtkSelection. This filter adds a scalar array called vtkOriginalCellIds that says what input cell produced each output cell. This is an example of a Pedigree ID which helps to trace back results. Depending on whether the selection has GLOBALIDS, VALUES or INDICES, the selection will use the contents of the array named in the GLOBALIDS DataSetAttribute, and arbitrary array, or the position (tuple id or number) within the cell or point array.

To create an instance of class vtkExtractSelectedIds, simply invoke its constructor as follows

```
obj = vtkExtractSelectedIds
```

33.80.2 Methods

The class vtkExtractSelectedIds has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelectedIds class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelectedIds = obj.NewInstance ()`
- `vtkExtractSelectedIds = obj.SafeDownCast (vtkObject o)`

33.81 vtkExtractSelectedLocations

33.81.1 Usage

vtkExtractSelectedLocations extracts all cells whose volume contain at least one point listed in the LOCATIONS content of the vtkSelection. This filter adds a scalar array called vtkOriginalCellIds that says what input cell produced each output cell. This is an example of a Pedigree ID which helps to trace back results.

To create an instance of class vtkExtractSelectedLocations, simply invoke its constructor as follows

```
obj = vtkExtractSelectedLocations
```

33.81.2 Methods

The class vtkExtractSelectedLocations has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelectedLocations class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelectedLocations = obj.NewInstance ()`
- `vtkExtractSelectedLocations = obj.SafeDownCast (vtkObject o)`

33.82 vtkExtractSelectedPolyDataIds

33.82.1 Usage

vtkExtractSelectedPolyDataIds extracts all cells in vtkSelection from a vtkPolyData.

To create an instance of class vtkExtractSelectedPolyDataIds, simply invoke its constructor as follows

```
obj = vtkExtractSelectedPolyDataIds
```

33.82.2 Methods

The class vtkExtractSelectedPolyDataIds has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelectedPolyDataIds class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelectedPolyDataIds = obj.NewInstance ()`
- `vtkExtractSelectedPolyDataIds = obj.SafeDownCast (vtkObject o)`

33.83 vtkExtractSelectedRows

33.83.1 Usage

The first input is a vtkTable to extract rows from. The second input is a vtkSelection containing the selected indices. The third input is a vtkAnnotationLayers containing selected indices. The field type of the input selection is ignored when converted to row indices.

To create an instance of class vtkExtractSelectedRows, simply invoke its constructor as follows

```
obj = vtkExtractSelectedRows
```

33.83.2 Methods

The class vtkExtractSelectedRows has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelectedRows class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelectedRows = obj.NewInstance ()`
- `vtkExtractSelectedRows = obj.SafeDownCast (vtkObject o)`
- `obj.SetSelectionConnection (vtkAlgorithmOutput in)` - A convenience method for setting the second input (i.e. the selection).
- `obj.SetAnnotationLayersConnection (vtkAlgorithmOutput in)` - A convenience method for setting the third input (i.e. the annotation layers).
- `int = obj.FillInputPortInformation (int port, vtkInformation info)` - Specify the first vtkGraph input and the second vtkSelection input.

- `obj.SetAddOriginalRowIdsArray (bool )` - When set, a column named vtkOriginalRowIds will be added to the output. False by default.
- `bool = obj.GetAddOriginalRowIdsArray ()` - When set, a column named vtkOriginalRowIds will be added to the output. False by default.
- `obj.AddOriginalRowIdsArrayOn ()` - When set, a column named vtkOriginalRowIds will be added to the output. False by default.
- `obj.AddOriginalRowIdsArrayOff ()` - When set, a column named vtkOriginalRowIds will be added to the output. False by default.

33.84 vtkExtractSelectedThresholds

33.84.1 Usage

vtkExtractSelectedThresholds extracts all cells and points with attribute values that lie within a vtkSelection's THRESHOLD contents. The selecion can specify to threshold a particular array within either the point or cell attribute data of the input. This is similar to vtkThreshold but allows mutliple thresholds ranges. This filter adds a scalar array called vtkOriginalCellIds that says what input cell produced each output cell. This is an example of a Pedigree ID which helps to trace back results.

To create an instance of class vtkExtractSelectedThresholds, simply invoke its constructor as follows

```
obj = vtkExtractSelectedThresholds
```

33.84.2 Methods

The class vtkExtractSelectedThresholds has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelectedThresholds class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelectedThresholds = obj.NewInstance ()`
- `vtkExtractSelectedThresholds = obj.SafeDownCast (vtkObject o)`

33.85 vtkExtractSelection

33.85.1 Usage

vtkExtractSelection extracts some subset of cells and points from its input dataset. The dataset is given on its first input port. The subset is described by the contents of the vtkSelection on its second input port. Depending on the content of the vtkSelection, this will use either a vtkExtractSelectedIds, vtkExtractSelectedFrustum vtkExtractSelectedLocations or a vtkExtractSelectedThreshold to perform the extraction.

To create an instance of class vtkExtractSelection, simply invoke its constructor as follows

```
obj = vtkExtractSelection
```

33.85.2 Methods

The class vtkExtractSelection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelection = obj.NewInstance ()`
- `vtkExtractSelection = obj.SafeDownCast (vtkObject o)`
- `obj.SetShowBounds (int )` - When On, this returns an unstructured grid that outlines selection area. Off is the default. Applicable only to Frustum selection extraction.
- `int = obj.GetShowBounds ()` - When On, this returns an unstructured grid that outlines selection area. Off is the default. Applicable only to Frustum selection extraction.
- `obj.ShowBoundsOn ()` - When On, this returns an unstructured grid that outlines selection area. Off is the default. Applicable only to Frustum selection extraction.
- `obj.ShowBoundsOff ()` - When On, this returns an unstructured grid that outlines selection area. Off is the default. Applicable only to Frustum selection extraction.
- `obj.SetUseProbeForLocations (int )` - When On, vtkProbeSelectedLocations is used for extracting selections of content type vtkSelection::LOCATIONS. Default is off and then vtkExtractSelectedLocations is used.
- `int = obj.GetUseProbeForLocations ()` - When On, vtkProbeSelectedLocations is used for extracting selections of content type vtkSelection::LOCATIONS. Default is off and then vtkExtractSelectedLocations is used.
- `obj.UseProbeForLocationsOn ()` - When On, vtkProbeSelectedLocations is used for extracting selections of content type vtkSelection::LOCATIONS. Default is off and then vtkExtractSelectedLocations is used.
- `obj.UseProbeForLocationsOff ()` - When On, vtkProbeSelectedLocations is used for extracting selections of content type vtkSelection::LOCATIONS. Default is off and then vtkExtractSelectedLocations is used.

33.86 vtkExtractSelectionBase

33.86.1 Usage

vtkExtractSelectionBase is an abstract base class for all extract selection filters. It defines some properties common to all extract selection filters.

To create an instance of class vtkExtractSelectionBase, simply invoke its constructor as follows

```
obj = vtkExtractSelectionBase
```

33.86.2 Methods

The class vtkExtractSelectionBase has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelectionBase class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelectionBase = obj.NewInstance ()`
- `vtkExtractSelectionBase = obj.SafeDownCast (vtkObject o)`
- `obj.SetSelectionConnection (vtkAlgorithmOutput algOutput)` - This flag tells the extraction filter not to convert the selected output into an unstructured grid, but instead to produce a vtkInsidedness array and add it to the input dataset. Default value is false(0).
- `obj.SetPreserveTopology (int )` - This flag tells the extraction filter not to convert the selected output into an unstructured grid, but instead to produce a vtkInsidedness array and add it to the input dataset. Default value is false(0).
- `int = obj.GetPreserveTopology ()` - This flag tells the extraction filter not to convert the selected output into an unstructured grid, but instead to produce a vtkInsidedness array and add it to the input dataset. Default value is false(0).
- `obj.PreserveTopologyOn ()` - This flag tells the extraction filter not to convert the selected output into an unstructured grid, but instead to produce a vtkInsidedness array and add it to the input dataset. Default value is false(0).
- `obj.PreserveTopologyOff ()` - This flag tells the extraction filter not to convert the selected output into an unstructured grid, but instead to produce a vtkInsidedness array and add it to the input dataset. Default value is false(0).

33.87 vtkExtractTemporalFieldData

33.87.1 Usage

vtkExtractTemporalFieldData extracts arrays from the input vtkFieldData. These arrays are assumed to contain temporal data, where the nth tuple contains the value for the nth timestep. The output is a 1D rectilinear grid where the XCoordinates correspond to time (the same array is also copied to a point array named Time or TimeData (if Time exists in the input). This algorithm does not produce a TIME_STEPS or TIME_RANGE information because it works across time. .Section Caveat vtkExtractTemporalFieldData puts a vtkOnePieceExtentTranslator in the output during RequestInformation(). As a result, the same whole extented is produced independent of the piece request. This algorithm works only with source that produce TIME_STEPS(). Continuous time range is not yet supported.

To create an instance of class vtkExtractTemporalFieldData, simply invoke its constructor as follows

```
obj = vtkExtractTemporalFieldData
```

33.87.2 Methods

The class vtkExtractTemporalFieldData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractTemporalFieldData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractTemporalFieldData = obj.NewInstance ()`
- `vtkExtractTemporalFieldData = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfTimeSteps ()` - Get the number of time steps

33.88 vtkExtractTensorComponents

33.88.1 Usage

vtkExtractTensorComponents is a filter that extracts components of a tensor to create a scalar, vector, normal, or texture coords. For example, if the tensor contains components of stress, then you could extract the normal stress in the x-direction as a scalar (i.e., tensor component (0,0).

To use this filter, you must set some boolean flags to control which data is extracted from the tensors, and whether you want to pass the tensor data through to the output. Also, you must specify the tensor component(s) for each type of data you want to extract. The tensor component(s) is(are) specified using matrix notation into a 3x3 matrix. That is, use the (row,column) address to specify a particular tensor component; and if the data you are extracting requires more than one component, use a list of addresses. (Note that the addresses are 0-offset -¿ (0,0) specifies upper left corner of the tensor.)

There are two optional methods to extract scalar data. You can extract the determinant of the tensor, or you can extract the effective stress of the tensor. These require that the ivar ExtractScalars is on, and the appropriate scalar extraction mode is set.

To create an instance of class vtkExtractTensorComponents, simply invoke its constructor as follows

```
obj = vtkExtractTensorComponents
```

33.88.2 Methods

The class vtkExtractTensorComponents has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractTensorComponents class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractTensorComponents = obj.NewInstance ()`
- `vtkExtractTensorComponents = obj.SafeDownCast (vtkObject o)`
- `obj.SetPassTensorsToOutput (int )` - Boolean controls whether tensor data is passed through to output.
- `int = obj.GetPassTensorsToOutput ()` - Boolean controls whether tensor data is passed through to output.
- `obj.PassTensorsToOutputOn ()` - Boolean controls whether tensor data is passed through to output.
- `obj.PassTensorsToOutputOff ()` - Boolean controls whether tensor data is passed through to output.
- `obj.SetExtractScalars (int )` - Boolean controls whether scalar data is extracted from tensor.
- `int = obj.GetExtractScalars ()` - Boolean controls whether scalar data is extracted from tensor.
- `obj.ExtractScalarsOn ()` - Boolean controls whether scalar data is extracted from tensor.

- `obj.ExtractScalarsOff ()` - Boolean controls whether scalar data is extracted from tensor.
- `obj.SetScalarComponents (int , int )` - Specify the (row,column) tensor component to extract as a scalar.
- `obj.SetScalarComponents (int a[2])` - Specify the (row,column) tensor component to extract as a scalar.
- `int = obj. GetScalarComponents ()` - Specify the (row,column) tensor component to extract as a scalar.
- `obj.SetScalarMode (int )` - Specify how to extract the scalar. You can extract it as one of the components of the tensor, as effective stress, or as the determinant of the tensor. If you extract a component make sure that you set the ScalarComponents ivar.
- `int = obj.GetScalarMode ()` - Specify how to extract the scalar. You can extract it as one of the components of the tensor, as effective stress, or as the determinant of the tensor. If you extract a component make sure that you set the ScalarComponents ivar.
- `obj.SetScalarModeToComponent ()` - Specify how to extract the scalar. You can extract it as one of the components of the tensor, as effective stress, or as the determinant of the tensor. If you extract a component make sure that you set the ScalarComponents ivar.
- `obj.SetScalarModeToEffectiveStress ()` - Specify how to extract the scalar. You can extract it as one of the components of the tensor, as effective stress, or as the determinant of the tensor. If you extract a component make sure that you set the ScalarComponents ivar.
- `obj.SetScalarModeToDeterminant ()` - Specify how to extract the scalar. You can extract it as one of the components of the tensor, as effective stress, or as the determinant of the tensor. If you extract a component make sure that you set the ScalarComponents ivar.
- `obj.ScalarIsComponent ()` - Specify how to extract the scalar. You can extract it as one of the components of the tensor, as effective stress, or as the determinant of the tensor. If you extract a component make sure that you set the ScalarComponents ivar.
- `obj.ScalarIsEffectiveStress ()` - Specify how to extract the scalar. You can extract it as one of the components of the tensor, as effective stress, or as the determinant of the tensor. If you extract a component make sure that you set the ScalarComponents ivar.
- `obj.ScalarIsDeterminant ()` - Specify how to extract the scalar. You can extract it as one of the components of the tensor, as effective stress, or as the determinant of the tensor. If you extract a component make sure that you set the ScalarComponents ivar.
- `obj.SetExtractVectors (int )` - Boolean controls whether vector data is extracted from tensor.
- `int = obj.GetExtractVectors ()` - Boolean controls whether vector data is extracted from tensor.
- `obj.ExtractVectorsOn ()` - Boolean controls whether vector data is extracted from tensor.
- `obj.ExtractVectorsOff ()` - Boolean controls whether vector data is extracted from tensor.
- `obj.SetVectorComponents (int , int , int , int , int , int )` - Specify the ((row,column)0,(row,column)1,(r tensor components to extract as a vector.
- `obj.SetVectorComponents (int a[6])` - Specify the ((row,column)0,(row,column)1,(row,column)2) tensor components to extract as a vector.
- `int = obj. GetVectorComponents ()` - Specify the ((row,column)0,(row,column)1,(row,column)2) tensor components to extract as a vector.
- `obj.SetExtractNormals (int )` - Boolean controls whether normal data is extracted from tensor.

- `int = obj.GetExtractNormals ()` - Boolean controls whether normal data is extracted from tensor.
- `obj.ExtractNormalsOn ()` - Boolean controls whether normal data is extracted from tensor.
- `obj.ExtractNormalsOff ()` - Boolean controls whether normal data is extracted from tensor.
- `obj.SetNormalizeNormals (int )` - Boolean controls whether normal vector is converted to unit normal after extraction.
- `int = obj.GetNormalizeNormals ()` - Boolean controls whether normal vector is converted to unit normal after extraction.
- `obj.NormalizeNormalsOn ()` - Boolean controls whether normal vector is converted to unit normal after extraction.
- `obj.NormalizeNormalsOff ()` - Boolean controls whether normal vector is converted to unit normal after extraction.
- `obj.SetNormalComponents (int , int , int , int , int , int )` - Specify the ((row,column)0,(row,column)1,(r tensor components to extract as a vector.
- `obj.SetNormalComponents (int a[6])` - Specify the ((row,column)0,(row,column)1,(row,column)2) tensor components to extract as a vector.
- `int = obj. GetNormalComponents ()` - Specify the ((row,column)0,(row,column)1,(row,column)2) tensor components to extract as a vector.
- `obj.SetExtractTCoords (int )` - Boolean controls whether texture coordinates are extracted from tensor.
- `int = obj.GetExtractTCoords ()` - Boolean controls whether texture coordinates are extracted from tensor.
- `obj.ExtractTCoordsOn ()` - Boolean controls whether texture coordinates are extracted from tensor.
- `obj.ExtractTCoordsOff ()` - Boolean controls whether texture coordinates are extracted from tensor.
- `obj.SetNumberOfTCoords (int )` - Set the dimension of the texture coordinates to extract.
- `int = obj.GetNumberOfTCoordsMinValue ()` - Set the dimension of the texture coordinates to extract.
- `int = obj.GetNumberOfTCoordsMaxValue ()` - Set the dimension of the texture coordinates to extract.
- `int = obj.GetNumberOfTCoords ()` - Set the dimension of the texture coordinates to extract.
- `obj.SetTCoordComponents (int , int , int , int , int , int )` - Specify the ((row,column)0,(row,column)1,(r tensor components to extract as a vector. Up to NumberOfTCoords components are extracted.
- `obj.SetTCoordComponents (int a[6])` - Specify the ((row,column)0,(row,column)1,(row,column)2) tensor components to extract as a vector. Up to NumberOfTCoords components are extracted.
- `int = obj. GetTCoordComponents ()` - Specify the ((row,column)0,(row,column)1,(row,column)2) tensor components to extract as a vector. Up to NumberOfTCoords components are extracted.

33.89 vtkExtractUnstructuredGrid

33.89.1 Usage

vtkExtractUnstructuredGrid is a general-purpose filter to extract geometry (and associated data) from an unstructured grid dataset. The extraction process is controlled by specifying a range of point ids, cell ids, or a bounding box (referred to as "Extent"). Those cells lying within these regions are sent to the output. The user has the choice of merging coincident points (Merging is on) or using the original point set (Merging is off).

To create an instance of class vtkExtractUnstructuredGrid, simply invoke its constructor as follows

```
obj = vtkExtractUnstructuredGrid
```

33.89.2 Methods

The class vtkExtractUnstructuredGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractUnstructuredGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractUnstructuredGrid = obj.NewInstance ()`
- `vtkExtractUnstructuredGrid = obj.SafeDownCast (vtkObject o)`
- `obj.SetPointClipping (int )` - Turn on/off selection of geometry by point id.
- `int = obj.GetPointClipping ()` - Turn on/off selection of geometry by point id.
- `obj.PointClippingOn ()` - Turn on/off selection of geometry by point id.
- `obj.PointClippingOff ()` - Turn on/off selection of geometry by point id.
- `obj.SetCellClipping (int )` - Turn on/off selection of geometry by cell id.
- `int = obj.GetCellClipping ()` - Turn on/off selection of geometry by cell id.
- `obj.CellClippingOn ()` - Turn on/off selection of geometry by cell id.
- `obj.CellClippingOff ()` - Turn on/off selection of geometry by cell id.
- `obj.SetExtentClipping (int )` - Turn on/off selection of geometry via bounding box.
- `int = obj.GetExtentClipping ()` - Turn on/off selection of geometry via bounding box.
- `obj.ExtentClippingOn ()` - Turn on/off selection of geometry via bounding box.
- `obj.ExtentClippingOff ()` - Turn on/off selection of geometry via bounding box.
- `obj.SetPointMinimum (vtkIdType )` - Specify the minimum point id for point id selection.
- `vtkIdType = obj.GetPointMinimumMinValue ()` - Specify the minimum point id for point id selection.
- `vtkIdType = obj.GetPointMinimumMaxValue ()` - Specify the minimum point id for point id selection.
- `vtkIdType = obj.GetPointMinimum ()` - Specify the minimum point id for point id selection.
- `obj.SetPointMaximum (vtkIdType )` - Specify the maximum point id for point id selection.

- `vtkIdType = obj.GetPointMaximumMinValue ()` - Specify the maximum point id for point id selection.
- `vtkIdType = obj.GetPointMaximumMaxValue ()` - Specify the maximum point id for point id selection.
- `vtkIdType = obj.GetPointMaximum ()` - Specify the maximum point id for point id selection.
- `obj.SetCellMinimum (vtkIdType )` - Specify the minimum cell id for point id selection.
- `vtkIdType = obj.GetCellMinimumMinValue ()` - Specify the minimum cell id for point id selection.
- `vtkIdType = obj.GetCellMinimumMaxValue ()` - Specify the minimum cell id for point id selection.
- `vtkIdType = obj.GetCellMinimum ()` - Specify the minimum cell id for point id selection.
- `obj.SetCellMaximum (vtkIdType )` - Specify the maximum cell id for point id selection.
- `vtkIdType = obj.GetCellMaximumMinValue ()` - Specify the maximum cell id for point id selection.
- `vtkIdType = obj.GetCellMaximumMaxValue ()` - Specify the maximum cell id for point id selection.
- `vtkIdType = obj.GetCellMaximum ()` - Specify the maximum cell id for point id selection.
- `obj.SetExtent (double xMin, double xMax, double yMin, double yMax, double zMin, double zMax)` - Specify a (xmin,xmax, ymin,ymax, zmin,zmax) bounding box to clip data.
- `obj.SetExtent (double extent[6])` - Set / get a (xmin,xmax, ymin,ymax, zmin,zmax) bounding box to clip data.
- `double = obj.GetExtent ()` - Set / get a (xmin,xmax, ymin,ymax, zmin,zmax) bounding box to clip data.
- `obj.SetMerging (int )` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `int = obj.GetMerging ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.MergingOn ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.MergingOff ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified.
- `long = obj.GetMTime ()` - Return the MTime also considering the locator.

33.90 vtkExtractVectorComponents

33.90.1 Usage

vtkExtractVectorComponents is a filter that extracts vector components as separate scalars. This is accomplished by creating three different outputs. Each output is the same as the input, except that the scalar values will be one of the three components of the vector. These can be found in the VxComponent, VyComponent, and VzComponent. Alternatively, if the ExtractToFieldData flag is set, the filter will put all the components in the field data. The first component will be the scalar and the others will be non-attribute arrays.

To create an instance of class vtkExtractVectorComponents, simply invoke its constructor as follows

```
obj = vtkExtractVectorComponents
```

33.90.2 Methods

The class vtkExtractVectorComponents has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractVectorComponents class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractVectorComponents = obj.NewInstance ()`
- `vtkExtractVectorComponents = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkDataSet input)` - Specify the input data or filter.
- `vtkDataSet = obj.GetVxComponent ()` - Get the output dataset representing velocity x-component. If output is NULL then input hasn't been set, which is necessary for abstract objects. (Note: this method returns the same information as the GetOutput() method with an index of 0.)
- `vtkDataSet = obj.GetVyComponent ()` - Get the output dataset representing velocity y-component. If output is NULL then input hasn't been set, which is necessary for abstract objects. (Note: this method returns the same information as the GetOutput() method with an index of 1.) Note that if ExtractToFieldData is true, this output will be empty.
- `vtkDataSet = obj.GetVzComponent ()` - Get the output dataset representing velocity z-component. If output is NULL then input hasn't been set, which is necessary for abstract objects. (Note: this method returns the same information as the GetOutput() method with an index of 2.) Note that if ExtractToFieldData is true, this output will be empty.
- `obj.SetExtractToFieldData (int )` - Determines whether the vector components will be put in separate outputs or in the first output's field data
- `int = obj.GetExtractToFieldData ()` - Determines whether the vector components will be put in separate outputs or in the first output's field data
- `obj.ExtractToFieldDataOn ()` - Determines whether the vector components will be put in separate outputs or in the first output's field data
- `obj.ExtractToFieldDataOff ()` - Determines whether the vector components will be put in separate outputs or in the first output's field data

33.91 vtkFeatureEdges

33.91.1 Usage

vtkFeatureEdges is a filter to extract special types of edges from input polygonal data. These edges are either 1) boundary (used by one polygon) or a line cell; 2) non-manifold (used by three or more polygons); 3) feature edges (edges used by two triangles and whose dihedral angle ¿ FeatureAngle); or 4) manifold edges (edges used by exactly two polygons). These edges may be extracted in any combination. Edges may also be "colored" (i.e., scalar values assigned) based on edge type. The cell coloring is assigned to the cell data of the extracted edges.

To create an instance of class vtkFeatureEdges, simply invoke its constructor as follows

```
obj = vtkFeatureEdges
```

33.91.2 Methods

The class vtkFeatureEdges has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFeatureEdges class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFeatureEdges = obj.NewInstance ()`
- `vtkFeatureEdges = obj.SafeDownCast (vtkObject o)`
- `obj.SetBoundaryEdges (int )` - Turn on/off the extraction of boundary edges.
- `int = obj.GetBoundaryEdges ()` - Turn on/off the extraction of boundary edges.
- `obj.BoundaryEdgesOn ()` - Turn on/off the extraction of boundary edges.
- `obj.BoundaryEdgesOff ()` - Turn on/off the extraction of boundary edges.
- `obj.SetFeatureEdges (int )` - Turn on/off the extraction of feature edges.
- `int = obj.GetFeatureEdges ()` - Turn on/off the extraction of feature edges.
- `obj.FeatureEdgesOn ()` - Turn on/off the extraction of feature edges.
- `obj.FeatureEdgesOff ()` - Turn on/off the extraction of feature edges.
- `obj.SetFeatureAngle (double )` - Specify the feature angle for extracting feature edges.
- `double = obj.GetFeatureAngleMinValue ()` - Specify the feature angle for extracting feature edges.
- `double = obj.GetFeatureAngleMaxValue ()` - Specify the feature angle for extracting feature edges.
- `double = obj.GetFeatureAngle ()` - Specify the feature angle for extracting feature edges.
- `obj.SetNonManifoldEdges (int )` - Turn on/off the extraction of non-manifold edges.
- `int = obj.GetNonManifoldEdges ()` - Turn on/off the extraction of non-manifold edges.
- `obj.NonManifoldEdgesOn ()` - Turn on/off the extraction of non-manifold edges.
- `obj.NonManifoldEdgesOff ()` - Turn on/off the extraction of non-manifold edges.
- `obj.SetManifoldEdges (int )` - Turn on/off the extraction of manifold edges.

- `int = obj.GetManifoldEdges ()` - Turn on/off the extraction of manifold edges.
- `obj.ManifoldEdgesOn ()` - Turn on/off the extraction of manifold edges.
- `obj.ManifoldEdgesOff ()` - Turn on/off the extraction of manifold edges.
- `obj.SetColoring (int )` - Turn on/off the coloring of edges by type.
- `int = obj.GetColoring ()` - Turn on/off the coloring of edges by type.
- `obj.ColoringOn ()` - Turn on/off the coloring of edges by type.
- `obj.ColoringOff ()` - Turn on/off the coloring of edges by type.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified.
- `long = obj.GetMTime ()` - Return MTime also considering the locator.

33.92 vtkFieldDataToAttributeDataFilter

33.92.1 Usage

vtkFieldDataToAttributeDataFilter is a class that maps field data into dataset attributes. The input to this filter is any type of dataset and the output is the same dataset (geometry/topology) with new attribute data (attribute data is passed through if not replaced during filter execution).

To use this filter you must specify which field data from the input dataset to use. There are three possibilities: the cell field data, the point field data, or the field data associated with the data object superclass. Then you specify which attribute data to create: either cell attribute data or point attribute data. Finally, you must define how to construct the various attribute data types (e.g., scalars, vectors, normals, etc.) from the arrays and the components of the arrays from the field data. This is done by associating components in the input field with components making up the attribute data. For example, you would specify a scalar with three components (RGB) by assigning components from the field for the R, then G, then B values of the scalars. You may also have to specify component ranges (for each R-G-B) to make sure that the number of R, G, and B values is the same. Also, you may want to normalize the components which helps distribute the data uniformly.

This filter is often used in conjunction with vtkDataObjectToDataSetFilter. vtkDataObjectToDataSet-Filter filter generates dataset topology and geometry and passes its input field data along to its output. Then this filter is used to generate the attribute data to go along with the dataset.

To create an instance of class vtkFieldDataToAttributeDataFilter, simply invoke its constructor as follows

```
obj = vtkFieldDataToAttributeDataFilter
```

33.92.2 Methods

The class vtkFieldDataToAttributeDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFieldDataToAttributeDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkFieldDataToAttributeDataFilter = obj.NewInstance ()`
- `vtkFieldDataToAttributeDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInputField (int )` - Specify which field data to use to generate the output attribute data. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.
- `int = obj.GetInputField ()` - Specify which field data to use to generate the output attribute data. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.
- `obj.SetInputFieldToDataObjectField ()` - Specify which field data to use to generate the output attribute data. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.
- `obj.SetInputFieldToPointDataField ()` - Specify which field data to use to generate the output attribute data. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.
- `obj.SetInputFieldToCellDataField ()` - Specify which field data to use to generate the output attribute data. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.
- `obj.SetOutputAttributeData (int )` - Specify which attribute data to output: point or cell data attributes.
- `int = obj.GetOutputAttributeData ()` - Specify which attribute data to output: point or cell data attributes.
- `obj.SetOutputAttributeDataToCellData ()` - Specify which attribute data to output: point or cell data attributes.
- `obj.SetOutputAttributeDataToPointData ()` - Specify which attribute data to output: point or cell data attributes.
- `obj.SetScalarComponent (int comp, string arrayName, int arrayComp, int min, int max, int normalize)` - Define the component(s) of the field to be used for the scalar components. Note that the parameter comp must lie between (0,4). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `obj.SetScalarComponent (int comp, string arrayName, int arrayComp)` - Define the component(s) of the field to be used for the scalar components. Note that the parameter comp must lie between (0,4). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `string = obj.GetScalarComponentArrayName (int comp)` - Define the component(s) of the field to be used for the scalar components. Note that the parameter comp must lie between (0,4). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetScalarComponentArrayComponent (int comp)` - Define the component(s) of the field to be used for the scalar components. Note that the parameter comp must lie between (0,4). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetScalarComponentMinRange (int comp)` - Define the component(s) of the field to be used for the scalar components. Note that the parameter comp must lie between (0,4). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `int = obj.GetScalarComponentMaxRange (int comp)` - Define the component(s) of the field to be used for the scalar components. Note that the parameter comp must lie between (0,4). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `int = obj.GetScalarComponentNormalizeFlag (int comp)` - Define the component(s) of the field to be used for the scalar components. Note that the parameter comp must lie between (0,4). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `obj.SetVectorComponent (int comp, string arrayName, int arrayComp, int min, int max, int normalize)` - Define the component(s) of the field to be used for the vector components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `obj.SetVectorComponent (int comp, string arrayName, int arrayComp)` - Define the component(s) of the field to be used for the vector components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `string = obj.GetVectorComponentArrayName (int comp)` - Define the component(s) of the field to be used for the vector components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `int = obj.GetVectorComponentArrayComponent (int comp)` - Define the component(s) of the field to be used for the vector components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `int = obj.GetVectorComponentMinRange (int comp)` - Define the component(s) of the field to be used for the vector components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `int = obj.GetVectorComponentMaxRange (int comp)` - Define the component(s) of the field to be used for the vector components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `int = obj.GetVectorComponentNormalizeFlag (int comp)` - Define the component(s) of the field to be used for the vector components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `obj.SetNormalComponent (int comp, string arrayName, int arrayComp, int min, int max, int normalize)` - Define the component(s) of the field to be used for the normal components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `obj.SetNormalComponent (int comp, string arrayName, int arrayComp)` - Define the component(s) of the field to be used for the normal components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `string = obj.GetNormalComponentArrayName (int comp)` - Define the component(s) of the field to be used for the normal components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetNormalComponentArrayComponent (int comp)` - Define the component(s) of the field to be used for the normal components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetNormalComponentMinRange (int comp)` - Define the component(s) of the field to be used for the normal components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetNormalComponentMaxRange (int comp)` - Define the component(s) of the field to be used for the normal components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetNormalComponentNormalizeFlag (int comp)` - Define the component(s) of the field to be used for the normal components. Note that the parameter comp must lie between (0,3). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `obj.SetTensorComponent (int comp, string arrayName, int arrayComp, int min, int max, int normalize)` - Define the components of the field to be used for the tensor components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `obj.SetTensorComponent (int comp, string arrayName, int arrayComp)` - Define the components of the field to be used for the tensor components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `string = obj.GetTensorComponentArrayName (int comp)` - Define the components of the field to be used for the tensor components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetTensorComponentArrayComponent (int comp)` - Define the components of the field to be used for the tensor components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetTensorComponentMinRange (int comp)` - Define the components of the field to be used for the tensor components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetTensorComponentMaxRange (int comp)` - Define the components of the field to be used for the tensor components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetTensorComponentNormalizeFlag (int comp)` - Define the components of the field to be used for the tensor components. Note that the parameter comp must lie between (0,9). To define

the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.

- `obj.SetTCoordComponent (int comp, string arrayName, int arrayComp, int min, int max, int normalize)` - Define the components of the field to be used for the cell texture coord components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `obj.SetTCoordComponent (int comp, string arrayName, int arrayComp)` - Define the components of the field to be used for the cell texture coord components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `string = obj.GetTCoordComponentArrayName (int comp)` - Define the components of the field to be used for the cell texture coord components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetTCoordComponentArrayComponent (int comp)` - Define the components of the field to be used for the cell texture coord components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetTCoordComponentMinRange (int comp)` - Define the components of the field to be used for the cell texture coord components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetTCoordComponentMaxRange (int comp)` - Define the components of the field to be used for the cell texture coord components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `int = obj.GetTCoordComponentNormalizeFlag (int comp)` - Define the components of the field to be used for the cell texture coord components. Note that the parameter comp must lie between (0,9). To define the field component to use you specify an array name and the component in that array. The (min,max) values are the range of data in the component you wish to extract.
- `obj.SetDefaultNormalize (int )` - Set the default Normalize() flag for those methods setting a default Normalize value (e.g., SetScalarComponents).
- `int = obj.GetDefaultNormalize ()` - Set the default Normalize() flag for those methods setting a default Normalize value (e.g., SetScalarComponents).
- `obj.DefaultNormalizeOn ()` - Set the default Normalize() flag for those methods setting a default Normalize value (e.g., SetScalarComponents).
- `obj.DefaultNormalizeOff ()` - Set the default Normalize() flag for those methods setting a default Normalize value (e.g., SetScalarComponents).

33.93 vtkFillHolesFilter

33.93.1 Usage

vtkFillHolesFilter is a filter that identifies and fills holes in input vtkPolyData meshes. Holes are identified by locating boundary edges, linking them together into loops, and then triangulating the resulting loops. Note that you can specify an approximate limit to the size of the hole that can be filled.

To create an instance of class vtkFillHolesFilter, simply invoke its constructor as follows

```
obj = vtkFillHolesFilter
```

33.93.2 Methods

The class vtkFillHolesFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFillHolesFilter class.

- `string = obj.GetClassName ()` - Standard methods for instantiation, type information and printing.
- `int = obj.IsA (string name)` - Standard methods for instantiation, type information and printing.
- `vtkFillHolesFilter = obj.NewInstance ()` - Standard methods for instantiation, type information and printing.
- `vtkFillHolesFilter = obj.SafeDownCast (vtkObject o)` - Standard methods for instantiation, type information and printing.
- `obj.SetHoleSize (double )` - Specify the maximum hole size to fill. This is represented as a radius to the bounding circumsphere containing the hole. Note that this is an approximate area; the actual area cannot be computed without first triangulating the hole.
- `double = obj.GetHoleSizeMinValue ()` - Specify the maximum hole size to fill. This is represented as a radius to the bounding circumsphere containing the hole. Note that this is an approximate area; the actual area cannot be computed without first triangulating the hole.
- `double = obj.GetHoleSizeMaxValue ()` - Specify the maximum hole size to fill. This is represented as a radius to the bounding circumsphere containing the hole. Note that this is an approximate area; the actual area cannot be computed without first triangulating the hole.
- `double = obj.GetHoleSize ()` - Specify the maximum hole size to fill. This is represented as a radius to the bounding circumsphere containing the hole. Note that this is an approximate area; the actual area cannot be computed without first triangulating the hole.

33.94 vtkFrustumSource

33.94.1 Usage

vtkFrustumSource creates a frustum defines by a set of planes. The frustum is represented with four-sided polygons. It is possible to specify extra lines to better visualize the field of view.

.SECTION Usage Typical use consists of 3 steps: 1. get the planes coefficients from a vtkCamera with vtkCamera::GetFrustumPlanes() 2. initialize the planes with vtkPlanes::SetFrustumPlanes() with the planes coefficients 3. pass the vtkPlanes to a vtkFrustumSource.

To create an instance of class vtkFrustumSource, simply invoke its constructor as follows

```
obj = vtkFrustumSource
```

33.94.2 Methods

The class vtkFrustumSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFrustumSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkFrustumSource = obj.NewInstance ()`
- `vtkFrustumSource = obj.SafeDownCast (vtkObject o)`
- `vtkPlanes = obj.GetPlanes ()` - Return the 6 planes defining the frustum. Initial value is NULL. The 6 planes are defined in this order: left,right,bottom,top,far,near. If Planes==NULL or if Planes-¿GetNumberOfPlanes()!=6 when RequestData() is called, an error message will be emitted and RequestData() will return right away.
- `obj.SetPlanes (vtkPlanes planes)` - Set the 6 planes defining the frustum.
- `bool = obj.GetShowLines ()` - Tells if some extra lines will be generated. Initial value is true.
- `obj.SetShowLines (bool )` - Tells if some extra lines will be generated. Initial value is true.
- `obj.ShowLinesOn ()` - Tells if some extra lines will be generated. Initial value is true.
- `obj.ShowLinesOff ()` - Tells if some extra lines will be generated. Initial value is true.
- `double = obj.GetLinesLength ()` - Length of the extra lines. This a stricly positive value. Initial value is 1.0.
- `obj.SetLinesLength (double )` - Length of the extra lines. This a stricly positive value. Initial value is 1.0.
- `long = obj.GetMTime ()` - Modified GetMTime because of Planes.

33.95 vtkGeodesicPath

33.95.1 Usage

Serves as a base class for algorithms that trace a geodesic path on a polygonal dataset.

To create an instance of class vtkGeodesicPath, simply invoke its constructor as follows

```
obj = vtkGeodesicPath
```

33.95.2 Methods

The class vtkGeodesicPath has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeodesicPath class.

- `string = obj.GetClassName ()` - Standard methids for printing and determining type information.
- `int = obj.IsA (string name)` - Standard methids for printing and determining type information.
- `vtkGeodesicPath = obj.NewInstance ()` - Standard methids for printing and determining type information.
- `vtkGeodesicPath = obj.SafeDownCast (vtkObject o)` - Standard methids for printing and determining type information.
- `double = obj.GetGeodesicLength ()`

33.96 vtkGeometryFilter

33.96.1 Usage

vtkGeometryFilter is a general-purpose filter to extract geometry (and associated data) from any type of dataset. Geometry is obtained as follows: all 0D, 1D, and 2D cells are extracted. All 2D faces that are used by only one 3D cell (i.e., boundary faces) are extracted. It also is possible to specify conditions on point ids, cell ids, and on bounding box (referred to as "Extent") to control the extraction process.

This filter also may be used to convert any type of data to polygonal type. The conversion process may be less than satisfactory for some 3D datasets. For example, this filter will extract the outer surface of a volume or structured grid dataset. (For structured data you may want to use vtkImageDataGeometryFilter, vtkStructuredGridGeometryFilter, vtkExtractUnstructuredGrid, vtkRectilinearGridGeometryFilter, or vtkExtractVOI.)

To create an instance of class vtkGeometryFilter, simply invoke its constructor as follows

```
obj = vtkGeometryFilter
```

33.96.2 Methods

The class vtkGeometryFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeometryFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeometryFilter = obj.NewInstance ()`
- `vtkGeometryFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetPointClipping (int )` - Turn on/off selection of geometry by point id.
- `int = obj.GetPointClipping ()` - Turn on/off selection of geometry by point id.
- `obj.PointClippingOn ()` - Turn on/off selection of geometry by point id.
- `obj.PointClippingOff ()` - Turn on/off selection of geometry by point id.
- `obj.SetCellClipping (int )` - Turn on/off selection of geometry by cell id.
- `int = obj.GetCellClipping ()` - Turn on/off selection of geometry by cell id.
- `obj.CellClippingOn ()` - Turn on/off selection of geometry by cell id.
- `obj.CellClippingOff ()` - Turn on/off selection of geometry by cell id.
- `obj.SetExtentClipping (int )` - Turn on/off selection of geometry via bounding box.
- `int = obj.GetExtentClipping ()` - Turn on/off selection of geometry via bounding box.
- `obj.ExtentClippingOn ()` - Turn on/off selection of geometry via bounding box.
- `obj.ExtentClippingOff ()` - Turn on/off selection of geometry via bounding box.
- `obj.SetPointMinimum (vtkIdType )` - Specify the minimum point id for point id selection.
- `vtkIdType = obj.GetPointMinimumMinValue ()` - Specify the minimum point id for point id selection.

- `vtkIdType = obj.GetPointMinimumMaxValue ()` - Specify the minimum point id for point id selection.
- `vtkIdType = obj.GetPointMinimum ()` - Specify the minimum point id for point id selection.
- `obj.SetPointMaximum (vtkIdType )` - Specify the maximum point id for point id selection.
- `vtkIdType = obj.GetPointMaximumMinValue ()` - Specify the maximum point id for point id selection.
- `vtkIdType = obj.GetPointMaximumMaxValue ()` - Specify the maximum point id for point id selection.
- `vtkIdType = obj.GetPointMaximum ()` - Specify the maximum point id for point id selection.
- `obj.SetCellMinimum (vtkIdType )` - Specify the minimum cell id for point id selection.
- `vtkIdType = obj.GetCellMinimumMinValue ()` - Specify the minimum cell id for point id selection.
- `vtkIdType = obj.GetCellMinimumMaxValue ()` - Specify the minimum cell id for point id selection.
- `vtkIdType = obj.GetCellMinimum ()` - Specify the minimum cell id for point id selection.
- `obj.SetCellMaximum (vtkIdType )` - Specify the maximum cell id for point id selection.
- `vtkIdType = obj.GetCellMaximumMinValue ()` - Specify the maximum cell id for point id selection.
- `vtkIdType = obj.GetCellMaximumMaxValue ()` - Specify the maximum cell id for point id selection.
- `vtkIdType = obj.GetCellMaximum ()` - Specify the maximum cell id for point id selection.
- `obj.SetExtent (double xMin, double xMax, double yMin, double yMax, double zMin, double zMax)` - Specify a (xmin,xmax, ymin,ymax, zmin,zmax) bounding box to clip data.
- `obj.SetExtent (double extent[6])` - Set / get a (xmin,xmax, ymin,ymax, zmin,zmax) bounding box to clip data.
- `double = obj.GetExtent ()` - Set / get a (xmin,xmax, ymin,ymax, zmin,zmax) bounding box to clip data.
- `obj.SetMerging (int )` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `int = obj.GetMerging ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.MergingOn ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.MergingOff ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified.
- `long = obj.GetMTime ()` - Return the MTime also considering the locator.

33.97 vtkGlyph2D

33.97.1 Usage

This subclass of vtkGlyph3D is a specialization to 2D. Transformations (i.e., translation, scaling, and rotation) are constrained to the plane. For example, rotations due to a vector are computed from the x-y coordinates of the vector only, and are assumed to occur around the z-axis. (See vtkGlyph3D for documentation on the interface to this class.)

To create an instance of class vtkGlyph2D, simply invoke its constructor as follows

```
obj = vtkGlyph2D
```

33.97.2 Methods

The class vtkGlyph2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGlyph2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGlyph2D = obj.NewInstance ()`
- `vtkGlyph2D = obj.SafeDownCast (vtkObject o)`

33.98 vtkGlyph3D

33.98.1 Usage

vtkGlyph3D is a filter that copies a geometric representation (called a glyph) to every point in the input dataset. The glyph is defined with polygonal data from a source filter input. The glyph may be oriented along the input vectors or normals, and it may be scaled according to scalar data or vector magnitude. More than one glyph may be used by creating a table of source objects, each defining a different glyph. If a table of glyphs is defined, then the table can be indexed into by using either scalar value or vector magnitude.

To use this object you'll have to provide an input dataset and a source to define the glyph. Then decide whether you want to scale the glyph and how to scale the glyph (using scalar value or vector magnitude). Next decide whether you want to orient the glyph, and whether to use the vector data or normal data to orient it. Finally, decide whether to use a table of glyphs, or just a single glyph. If you use a table of glyphs, you'll have to decide whether to index into it with scalar value or with vector magnitude.

To create an instance of class vtkGlyph3D, simply invoke its constructor as follows

```
obj = vtkGlyph3D
```

33.98.2 Methods

The class vtkGlyph3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGlyph3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGlyph3D = obj.NewInstance ()`
- `vtkGlyph3D = obj.SafeDownCast (vtkObject o)`

- `obj.SetSource (vtkPolyData pd)` - Set the source to use for he glyph. Old style. See SetSourceConnection.
- `obj.SetSource (int id, vtkPolyData pd)` - Specify a source object at a specified table location. Old style. See SetSourceConnection.
- `obj.SetSourceConnection (int id, vtkAlgorithmOutput algOutput)` - Specify a source object at a specified table location. New style. Source connection is stored in port 1. This method is equivalent to SetInputConnection(1, id, outputPort).
- `obj.SetSourceConnection (vtkAlgorithmOutput algOutput)` - Get a pointer to a source object at a specified table location.
- `vtkPolyData = obj.GetSource (int id)` - Get a pointer to a source object at a specified table location.
- `obj.SetScaling (int )` - Turn on/off scaling of source geometry.
- `obj.ScalingOn ()` - Turn on/off scaling of source geometry.
- `obj.ScalingOff ()` - Turn on/off scaling of source geometry.
- `int = obj.GetScaling ()` - Turn on/off scaling of source geometry.
- `obj.SetScaleMode (int )` - Either scale by scalar or by vector/normal magnitude.
- `int = obj.GetScaleMode ()` - Either scale by scalar or by vector/normal magnitude.
- `obj.SetScaleModeToScaleByScalar ()` - Either scale by scalar or by vector/normal magnitude.
- `obj.SetScaleModeToScaleByVector ()` - Either scale by scalar or by vector/normal magnitude.
- `obj.SetScaleModeToScaleByVectorComponents ()` - Either scale by scalar or by vector/normal magnitude.
- `obj.SetScaleModeToDataScalingOff ()` - Either scale by scalar or by vector/normal magnitude.
- `string = obj.GetScaleModeAsString ()` - Either scale by scalar or by vector/normal magnitude.
- `obj.SetColorMode (int )` - Either color by scale, scalar or by vector/normal magnitude.
- `int = obj.GetColorMode ()` - Either color by scale, scalar or by vector/normal magnitude.
- `obj.SetColorModeToColorByScale ()` - Either color by scale, scalar or by vector/normal magnitude.
- `obj.SetColorModeToColorByScalar ()` - Either color by scale, scalar or by vector/normal magnitude.
- `obj.SetColorModeToColorByVector ()` - Either color by scale, scalar or by vector/normal magnitude.
- `string = obj.GetColorModeAsString ()` - Either color by scale, scalar or by vector/normal magnitude.
- `obj.SetScaleFactor (double )` - Specify scale factor to scale object by.
- `double = obj.GetScaleFactor ()` - Specify scale factor to scale object by.
- `obj.SetRange (double , double )` - Specify range to map scalar values into.
- `obj.SetRange (double  a[2])` - Specify range to map scalar values into.
- `double = obj. GetRange ()` - Specify range to map scalar values into.
- `obj.SetOrient (int )` - Turn on/off orienting of input geometry along vector/normal.

- `obj.OrientOn ()` - Turn on/off orienting of input geometry along vector/normal.
- `obj.OrientOff ()` - Turn on/off orienting of input geometry along vector/normal.
- `int = obj.GetOrient ()` - Turn on/off orienting of input geometry along vector/normal.
- `obj.SetClamping (int )` - Turn on/off clamping of "scalar" values to range. (Scalar value may be vector magnitude if ScaleByVector() is enabled.)
- `obj.ClampingOn ()` - Turn on/off clamping of "scalar" values to range. (Scalar value may be vector magnitude if ScaleByVector() is enabled.)
- `obj.ClampingOff ()` - Turn on/off clamping of "scalar" values to range. (Scalar value may be vector magnitude if ScaleByVector() is enabled.)
- `int = obj.GetClamping ()` - Turn on/off clamping of "scalar" values to range. (Scalar value may be vector magnitude if ScaleByVector() is enabled.)
- `obj.SetVectorMode (int )` - Specify whether to use vector or normal to perform vector operations.
- `int = obj.GetVectorMode ()` - Specify whether to use vector or normal to perform vector operations.
- `obj.SetVectorModeToUseVector ()` - Specify whether to use vector or normal to perform vector operations.
- `obj.SetVectorModeToUseNormal ()` - Specify whether to use vector or normal to perform vector operations.
- `obj.SetVectorModeToVectorRotationOff ()` - Specify whether to use vector or normal to perform vector operations.
- `string = obj.GetVectorModeAsString ()` - Specify whether to use vector or normal to perform vector operations.
- `obj.SetIndexMode (int )` - Index into table of sources by scalar, by vector/normal magnitude, or no indexing. If indexing is turned off, then the first source glyph in the table of glyphs is used. Note that indexing mode will only use the InputScalarsSelection array and not the InputColorScalarsSelection as the scalar source if an array is specified.
- `int = obj.GetIndexMode ()` - Index into table of sources by scalar, by vector/normal magnitude, or no indexing. If indexing is turned off, then the first source glyph in the table of glyphs is used. Note that indexing mode will only use the InputScalarsSelection array and not the InputColorScalarsSelection as the scalar source if an array is specified.
- `obj.SetIndexModeToScalar ()` - Index into table of sources by scalar, by vector/normal magnitude, or no indexing. If indexing is turned off, then the first source glyph in the table of glyphs is used. Note that indexing mode will only use the InputScalarsSelection array and not the InputColorScalarsSelection as the scalar source if an array is specified.
- `obj.SetIndexModeToVector ()` - Index into table of sources by scalar, by vector/normal magnitude, or no indexing. If indexing is turned off, then the first source glyph in the table of glyphs is used. Note that indexing mode will only use the InputScalarsSelection array and not the InputColorScalarsSelection as the scalar source if an array is specified.
- `obj.SetIndexModeToOff ()` - Index into table of sources by scalar, by vector/normal magnitude, or no indexing. If indexing is turned off, then the first source glyph in the table of glyphs is used. Note that indexing mode will only use the InputScalarsSelection array and not the InputColorScalarsSelection as the scalar source if an array is specified.

- `string = obj.GetIndexModeAsString ()` - Index into table of sources by scalar, by vector/normal magnitude, or no indexing. If indexing is turned off, then the first source glyph in the table of glyphs is used. Note that indexing mode will only use the InputScalarsSelection array and not the InputColorScalarsSelection as the scalar source if an array is specified.

- `obj.SetGeneratePointIds (int )` - Enable/disable the generation of point ids as part of the output. The point ids are the id of the input generating point. The point ids are stored in the output point field data and named "InputPointIds". Point generation is useful for debugging and pick operations.

- `int = obj.GetGeneratePointIds ()` - Enable/disable the generation of point ids as part of the output. The point ids are the id of the input generating point. The point ids are stored in the output point field data and named "InputPointIds". Point generation is useful for debugging and pick operations.

- `obj.GeneratePointIdsOn ()` - Enable/disable the generation of point ids as part of the output. The point ids are the id of the input generating point. The point ids are stored in the output point field data and named "InputPointIds". Point generation is useful for debugging and pick operations.

- `obj.GeneratePointIdsOff ()` - Enable/disable the generation of point ids as part of the output. The point ids are the id of the input generating point. The point ids are stored in the output point field data and named "InputPointIds". Point generation is useful for debugging and pick operations.

- `obj.SetPointIdsName (string )` - Set/Get the name of the PointIds array if generated. By default the Ids are named "InputPointIds", but this can be changed with this function.

- `string = obj.GetPointIdsName ()` - Set/Get the name of the PointIds array if generated. By default the Ids are named "InputPointIds", but this can be changed with this function.

- `obj.SetFillCellData (int )` - Enable/disable the generation of cell data as part of the output. The cell data at each cell will match the point data of the input at the glyphed point.

- `int = obj.GetFillCellData ()` - Enable/disable the generation of cell data as part of the output. The cell data at each cell will match the point data of the input at the glyphed point.

- `obj.FillCellDataOn ()` - Enable/disable the generation of cell data as part of the output. The cell data at each cell will match the point data of the input at the glyphed point.

- `obj.FillCellDataOff ()` - Enable/disable the generation of cell data as part of the output. The cell data at each cell will match the point data of the input at the glyphed point.

- `int = obj.IsPointVisible (vtkDataSet , vtkIdType )` - This can be overwritten by subclass to return 0 when a point is blanked. Default implementation is to always return 1;

33.99 vtkGlyphSource2D

33.99.1 Usage

vtkGlyphSource2D can generate a family of 2D glyphs each of which lies in the x-y plane (i.e., the z-coordinate is zero). The class is a helper class to be used with vtkGlyph2D and vtkXYPlotActor.

To use this class, specify the glyph type to use and its attributes. Attributes include its position (i.e., center point), scale, color, and whether the symbol is filled or not (a polygon or closed line sequence). You can also put a short line through the glyph running from -x to +x (the glyph looks like it's on a line), or a cross.

To create an instance of class vtkGlyphSource2D, simply invoke its constructor as follows

```
obj = vtkGlyphSource2D
```

33.99.2 Methods

The class vtkGlyphSource2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGlyphSource2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGlyphSource2D = obj.NewInstance ()`
- `vtkGlyphSource2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetCenter (double , double , double )` - Set the center of the glyph. By default the center is (0,0,0).
- `obj.SetCenter (double  a[3])` - Set the center of the glyph. By default the center is (0,0,0).
- `double = obj. GetCenter ()` - Set the center of the glyph. By default the center is (0,0,0).
- `obj.SetScale (double )` - Set the scale of the glyph. Note that the glyphs are designed to fit in the (1,1) rectangle.
- `double = obj.GetScaleMinValue ()` - Set the scale of the glyph. Note that the glyphs are designed to fit in the (1,1) rectangle.
- `double = obj.GetScaleMaxValue ()` - Set the scale of the glyph. Note that the glyphs are designed to fit in the (1,1) rectangle.
- `double = obj.GetScale ()` - Set the scale of the glyph. Note that the glyphs are designed to fit in the (1,1) rectangle.
- `obj.SetScale2 (double )` - Set the scale of optional portions of the glyph (e.g., the dash and cross is DashOn() and CrossOn()).
- `double = obj.GetScale2MinValue ()` - Set the scale of optional portions of the glyph (e.g., the dash and cross is DashOn() and CrossOn()).
- `double = obj.GetScale2MaxValue ()` - Set the scale of optional portions of the glyph (e.g., the dash and cross is DashOn() and CrossOn()).
- `double = obj.GetScale2 ()` - Set the scale of optional portions of the glyph (e.g., the dash and cross is DashOn() and CrossOn()).
- `obj.SetColor (double , double , double )` - Set the color of the glyph. The default color is white.
- `obj.SetColor (double  a[3])` - Set the color of the glyph. The default color is white.
- `double = obj. GetColor ()` - Set the color of the glyph. The default color is white.
- `obj.SetFilled (int )` - Specify whether the glyph is filled (a polygon) or not (a closed polygon defined by line segments). This only applies to 2D closed glyphs.
- `int = obj.GetFilled ()` - Specify whether the glyph is filled (a polygon) or not (a closed polygon defined by line segments). This only applies to 2D closed glyphs.
- `obj.FilledOn ()` - Specify whether the glyph is filled (a polygon) or not (a closed polygon defined by line segments). This only applies to 2D closed glyphs.
- `obj.FilledOff ()` - Specify whether the glyph is filled (a polygon) or not (a closed polygon defined by line segments). This only applies to 2D closed glyphs.

- `obj.SetDash (int )` - Specify whether a short line segment is drawn through the glyph. (This is in addition to the glyph. If the glyph type is set to "Dash" there is no need to enable this flag.)
- `int = obj.GetDash ()` - Specify whether a short line segment is drawn through the glyph. (This is in addition to the glyph. If the glyph type is set to "Dash" there is no need to enable this flag.)
- `obj.DashOn ()` - Specify whether a short line segment is drawn through the glyph. (This is in addition to the glyph. If the glyph type is set to "Dash" there is no need to enable this flag.)
- `obj.DashOff ()` - Specify whether a short line segment is drawn through the glyph. (This is in addition to the glyph. If the glyph type is set to "Dash" there is no need to enable this flag.)
- `obj.SetCross (int )` - Specify whether a cross is drawn as part of the glyph. (This is in addition to the glyph. If the glyph type is set to "Cross" there is no need to enable this flag.)
- `int = obj.GetCross ()` - Specify whether a cross is drawn as part of the glyph. (This is in addition to the glyph. If the glyph type is set to "Cross" there is no need to enable this flag.)
- `obj.CrossOn ()` - Specify whether a cross is drawn as part of the glyph. (This is in addition to the glyph. If the glyph type is set to "Cross" there is no need to enable this flag.)
- `obj.CrossOff ()` - Specify whether a cross is drawn as part of the glyph. (This is in addition to the glyph. If the glyph type is set to "Cross" there is no need to enable this flag.)
- `obj.SetRotationAngle (double )` - Specify an angle (in degrees) to rotate the glyph around the z-axis. Using this ivar, it is possible to generate rotated glyphs (e.g., crosses, arrows, etc.)
- `double = obj.GetRotationAngle ()` - Specify an angle (in degrees) to rotate the glyph around the z-axis. Using this ivar, it is possible to generate rotated glyphs (e.g., crosses, arrows, etc.)
- `obj.SetGlyphType (int )` - Specify the type of glyph to generate.
- `int = obj.GetGlyphTypeMinValue ()` - Specify the type of glyph to generate.
- `int = obj.GetGlyphTypeMaxValue ()` - Specify the type of glyph to generate.
- `int = obj.GetGlyphType ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToNone ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToVertex ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToDash ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToCross ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToThickCross ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToTriangle ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToSquare ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToCircle ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToDiamond ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToArrow ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToThickArrow ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToHookedArrow ()` - Specify the type of glyph to generate.
- `obj.SetGlyphTypeToEdgeArrow ()`

33.100 vtkGradientFilter

33.100.1 Usage

Estimates the gradient of a field in a data set. The gradient calculation is dependent on the input dataset type. The created gradient array is of the same type as the array it is calculated from (e.g. point data or cell data) as well as data type (e.g. float, double). At the boundary the gradient is not central differencing. The output array has 3*number of components of the input data array. The ordering for the output tuple will be du/dx, du/dy, du/dz, dv/dx, dv/dy, dv/dz, dw/dx, dw/dy, dw/dz for an input array u, v, w.

To create an instance of class vtkGradientFilter, simply invoke its constructor as follows

```
obj = vtkGradientFilter
```

33.100.2 Methods

The class vtkGradientFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGradientFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGradientFilter = obj.NewInstance ()`
- `vtkGradientFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInputScalars (int fieldAssociation, string name)` - These are basically a convenience method that calls SetInputArrayToProcess to set the array used as the input scalars. The fieldAssociation comes from the vtkDataObject::FieldAssocations enum. The fieldAttributeType comes from the vtkDataSetAttributes::AttributeTypes enum.
- `obj.SetInputScalars (int fieldAssociation, int fieldAttributeType)` - These are basically a convenience method that calls SetInputArrayToProcess to set the array used as the input scalars. The fieldAssociation comes from the vtkDataObject::FieldAssocations enum. The fieldAttributeType comes from the vtkDataSetAttributes::AttributeTypes enum.
- `string = obj.GetResultArrayName ()` - Get/Set the name of the resulting array to create. If NULL (the default) then the output array will be named "Gradients".
- `obj.SetResultArrayName (string )` - Get/Set the name of the resulting array to create. If NULL (the default) then the output array will be named "Gradients".
- `int = obj.GetFasterApproximation ()` - When this flag is on (default is off), the gradient filter will provide a less accurate (but close) algorithm that performs fewer derivative calculations (and is therefore faster). The error contains some smoothing of the output data and some possible errors on the boundary. This parameter has no effect when performing the gradient of cell data. This only applies if the input grid is a vtkUnstructuredGrid or a vtkPolyData.
- `obj.SetFasterApproximation (int )` - When this flag is on (default is off), the gradient filter will provide a less accurate (but close) algorithm that performs fewer derivative calculations (and is therefore faster). The error contains some smoothing of the output data and some possible errors on the boundary. This parameter has no effect when performing the gradient of cell data. This only applies if the input grid is a vtkUnstructuredGrid or a vtkPolyData.

- `obj.FasterApproximationOn ()` - When this flag is on (default is off), the gradient filter will provide a less accurate (but close) algorithm that performs fewer derivative calculations (and is therefore faster). The error contains some smoothing of the output data and some possible errors on the boundary. This parameter has no effect when performing the gradient of cell data. This only applies if the input grid is a vtkUnstructuredGrid or a vtkPolyData.
- `obj.FasterApproximationOff ()` - When this flag is on (default is off), the gradient filter will provide a less accurate (but close) algorithm that performs fewer derivative calculations (and is therefore faster). The error contains some smoothing of the output data and some possible errors on the boundary. This parameter has no effect when performing the gradient of cell data. This only applies if the input grid is a vtkUnstructuredGrid or a vtkPolyData.
- `obj.SetComputeVorticity (int )` - Set the resultant array to be vorticity/curl of the input array. The input array must have 3 components.
- `int = obj.GetComputeVorticity ()` - Set the resultant array to be vorticity/curl of the input array. The input array must have 3 components.
- `obj.ComputeVorticityOn ()` - Set the resultant array to be vorticity/curl of the input array. The input array must have 3 components.
- `obj.ComputeVorticityOff ()` - Set the resultant array to be vorticity/curl of the input array. The input array must have 3 components.

33.101 vtkGraphGeodesicPath

33.101.1 Usage

Serves as a base class for algorithms that trace a geodesic on a polygonal dataset treating it as a graph. ie points connecting the vertices of the graph

To create an instance of class vtkGraphGeodesicPath, simply invoke its constructor as follows

```
obj = vtkGraphGeodesicPath
```

33.101.2 Methods

The class vtkGraphGeodesicPath has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphGeodesicPath class.

- `string = obj.GetClassName ()` - Standard methids for printing and determining type information.
- `int = obj.IsA (string name)` - Standard methids for printing and determining type information.
- `vtkGraphGeodesicPath = obj.NewInstance ()` - Standard methids for printing and determining type information.
- `vtkGraphGeodesicPath = obj.SafeDownCast (vtkObject o)` - Standard methids for printing and determining type information.
- `vtkIdType = obj.GetStartVertex ()` - The vertex at the start of the shortest path
- `obj.SetStartVertex (vtkIdType )` - The vertex at the start of the shortest path
- `vtkIdType = obj.GetEndVertex ()` - The vertex at the end of the shortest path
- `obj.SetEndVertex (vtkIdType )` - The vertex at the end of the shortest path

33.102 vtkGraphLayoutFilter

33.102.1 Usage

vtkGraphLayoutFilter will reposition a network of nodes, connected by lines or polylines, into a more pleasing arrangement. The class implements a simple force-directed placement algorithm (Fruchterman & Reingold "Graph Drawing by Force-directed Placement" Software-Practice and Experience 21(11) 1991).

The input to the filter is a vtkPolyData representing the undirected graphs. A graph is represented by a set of polylines and/or lines. The output is also a vtkPolyData, where the point positions have been modified. To use the filter, specify whether you wish the layout to occur in 2D or 3D; the bounds in which the graph should lie (note that you can just use automatic bounds computation); and modify the cool down rate (controls the final process of simulated annealing).

To create an instance of class vtkGraphLayoutFilter, simply invoke its constructor as follows

```
obj = vtkGraphLayoutFilter
```

33.102.2 Methods

The class vtkGraphLayoutFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphLayoutFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphLayoutFilter = obj.NewInstance ()`
- `vtkGraphLayoutFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetGraphBounds (double , double , double , double , double , double )` - Set / get the region in space in which to place the final graph. The GraphBounds only affects the results if AutomaticBoundsComputation is off.
- `obj.SetGraphBounds (double a[6])` - Set / get the region in space in which to place the final graph. The GraphBounds only affects the results if AutomaticBoundsComputation is off.
- `double = obj. GetGraphBounds ()` - Set / get the region in space in which to place the final graph. The GraphBounds only affects the results if AutomaticBoundsComputation is off.
- `obj.SetAutomaticBoundsComputation (int )` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `int = obj.GetAutomaticBoundsComputation ()` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `obj.AutomaticBoundsComputationOn ()` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `obj.AutomaticBoundsComputationOff ()` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `obj.SetMaxNumberOfIterations (int )` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified.

- `int = obj.GetMaxNumberOfIterationsMinValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified.
- `int = obj.GetMaxNumberOfIterationsMaxValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified.
- `int = obj.GetMaxNumberOfIterations ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified.
- `obj.SetCoolDownRate (double )` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified.
- `double = obj.GetCoolDownRateMinValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified.
- `double = obj.GetCoolDownRateMaxValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified.
- `double = obj.GetCoolDownRate ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified.
- `obj.SetThreeDimensionalLayout (int )`
- `int = obj.GetThreeDimensionalLayout ()`
- `obj.ThreeDimensionalLayoutOn ()`
- `obj.ThreeDimensionalLayoutOff ()`

33.103 vtkGraphToPoints

33.103.1 Usage

Converts a vtkGraph to a vtkPolyData containing a set of points. This assumes that the points of the graph have already been filled (perhaps by vtkGraphLayout). The vertex data is passed along to the point data.

To create an instance of class vtkGraphToPoints, simply invoke its constructor as follows

```
obj = vtkGraphToPoints
```

33.103.2 Methods

The class vtkGraphToPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphToPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphToPoints = obj.NewInstance ()`
- `vtkGraphToPoints = obj.SafeDownCast (vtkObject o)`

33.104 vtkGraphToPolyData

33.104.1 Usage

Converts a vtkGraph to a vtkPolyData. This assumes that the points of the graph have already been filled (perhaps by vtkGraphLayout), and coverts all the edge of the graph into lines in the polydata. The vertex data is passed along to the point data, and the edge data is passed along to the cell data.

Only the owned graph edges (i.e. edges with ghost level 0) are copied into the vtkPolyData.

To create an instance of class vtkGraphToPolyData, simply invoke its constructor as follows

```
obj = vtkGraphToPolyData
```

33.104.2 Methods

The class vtkGraphToPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphToPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphToPolyData = obj.NewInstance ()`
- `vtkGraphToPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetEdgeGlyphOutput (bool )` - Create a second output containing points and orientation vectors for drawing arrows or other glyphs on edges. This output should be set as the first input to vtkGlyph3D to place glyphs on the edges. vtkGlyphSource2D's VTK_EDGEARROW_GLYPH provides a good glyph for drawing arrows. Default value is off.
- `bool = obj.GetEdgeGlyphOutput ()` - Create a second output containing points and orientation vectors for drawing arrows or other glyphs on edges. This output should be set as the first input to vtkGlyph3D to place glyphs on the edges. vtkGlyphSource2D's VTK_EDGEARROW_GLYPH provides a good glyph for drawing arrows. Default value is off.
- `obj.EdgeGlyphOutputOn ()` - Create a second output containing points and orientation vectors for drawing arrows or other glyphs on edges. This output should be set as the first input to vtkGlyph3D to place glyphs on the edges. vtkGlyphSource2D's VTK_EDGEARROW_GLYPH provides a good glyph for drawing arrows. Default value is off.
- `obj.EdgeGlyphOutputOff ()` - Create a second output containing points and orientation vectors for drawing arrows or other glyphs on edges. This output should be set as the first input to vtkGlyph3D to place glyphs on the edges. vtkGlyphSource2D's VTK_EDGEARROW_GLYPH provides a good glyph for drawing arrows. Default value is off.
- `obj.SetEdgeGlyphPosition (double )` - The position of the glyph point along the edge. 0 puts a glyph point at the source of each edge. 1 puts a glyph point at the target of each edge. An intermediate value will place the glyph point between the source and target. The default value is 1.
- `double = obj.GetEdgeGlyphPosition ()` - The position of the glyph point along the edge. 0 puts a glyph point at the source of each edge. 1 puts a glyph point at the target of each edge. An intermediate value will place the glyph point between the source and target. The default value is 1.

33.105 vtkGridSynchronizedTemplates3D

33.105.1 Usage

vtkGridSynchronizedTemplates3D is a 3D implementation of the synchronized template algorithm.

To create an instance of class vtkGridSynchronizedTemplates3D, simply invoke its constructor as follows

```
obj = vtkGridSynchronizedTemplates3D
```

33.105.2 Methods

The class vtkGridSynchronizedTemplates3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGridSynchronizedTemplates3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGridSynchronizedTemplates3D = obj.NewInstance ()`
- `vtkGridSynchronizedTemplates3D = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Because we delegate to vtkContourValues
- `obj.SetComputeNormals (int )` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeNormals ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOn ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOff ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeGradients (int )` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeGradients ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOn ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOff ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.

- `obj.SetComputeScalars (int )` - Set/Get the computation of scalars.
- `int = obj.GetComputeScalars ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOn ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOff ()` - Set/Get the computation of scalars.
- `obj.SetValue (int i, double value)` - Get the ith contour value.
- `double = obj.GetValue (int i)` - Get a pointer to an array of contour values. There will be GetNumberOfContours() values in the list.
- `obj.GetValues (double contourValues)` - Set the number of contours to place into the list. You only really need to use this method to reduce list size. The method SetValue() will automatically increase list size as needed.
- `obj.SetNumberOfContours (int number)` - Get the number of contours in the list of contour values.
- `int = obj.GetNumberOfContours ()` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double range[2])` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Needed by templated functions.
- `int = obj.GetExecuteExtent ()` - Needed by templated functions.
- `obj.SetInputMemoryLimit (long limit)` - This filter will initiate streaming so that no piece requested from the input will be larger than this value (KiloBytes).

33.106 vtkHedgeHog

33.106.1 Usage

vtkHedgeHog creates oriented lines from the input data set. Line length is controlled by vector (or normal) magnitude times scale factor. If VectorMode is UseNormal, normals determine the orientation of the lines. Lines are colored by scalar data, if available.

To create an instance of class vtkHedgeHog, simply invoke its constructor as follows

```
obj = vtkHedgeHog
```

33.106.2 Methods

The class vtkHedgeHog has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHedgeHog class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHedgeHog = obj.NewInstance ()`
- `vtkHedgeHog = obj.SafeDownCast (vtkObject o)`
- `obj.SetScaleFactor (double )` - Set scale factor to control size of oriented lines.

- `double = obj.GetScaleFactor ()` - Set scale factor to control size of oriented lines.
- `obj.SetVectorMode (int )` - Specify whether to use vector or normal to perform vector operations.
- `int = obj.GetVectorMode ()` - Specify whether to use vector or normal to perform vector operations.
- `obj.SetVectorModeToUseVector ()` - Specify whether to use vector or normal to perform vector operations.
- `obj.SetVectorModeToUseNormal ()` - Specify whether to use vector or normal to perform vector operations.
- `string = obj.GetVectorModeAsString ()` - Specify whether to use vector or normal to perform vector operations.

33.107 vtkHierarchicalDataExtractDataSets

33.107.1 Usage

Legacy class. Use vtkExtractDataSets instead.

To create an instance of class vtkHierarchicalDataExtractDataSets, simply invoke its constructor as follows

```
obj = vtkHierarchicalDataExtractDataSets
```

33.107.2 Methods

The class vtkHierarchicalDataExtractDataSets has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalDataExtractDataSets class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHierarchicalDataExtractDataSets = obj.NewInstance ()`
- `vtkHierarchicalDataExtractDataSets = obj.SafeDownCast (vtkObject o)`

33.108 vtkHierarchicalDataExtractLevel

33.108.1 Usage

Legacy class. Use vtkExtractLevel instead.

To create an instance of class vtkHierarchicalDataExtractLevel, simply invoke its constructor as follows

```
obj = vtkHierarchicalDataExtractLevel
```

33.108.2 Methods

The class vtkHierarchicalDataExtractLevel has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalDataExtractLevel class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkHierarchicalDataExtractLevel = obj.NewInstance ()`
- `vtkHierarchicalDataExtractLevel = obj.SafeDownCast (vtkObject o)`

33.109 vtkHierarchicalDataLevelFilter

33.109.1 Usage

Legacy class. Use vtkLevelIdScalars instead.

To create an instance of class vtkHierarchicalDataLevelFilter, simply invoke its constructor as follows

```
obj = vtkHierarchicalDataLevelFilter
```

33.109.2 Methods

The class vtkHierarchicalDataLevelFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalDataLevelFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHierarchicalDataLevelFilter = obj.NewInstance ()`
- `vtkHierarchicalDataLevelFilter = obj.SafeDownCast (vtkObject o)`

33.110 vtkHierarchicalDataSetGeometryFilter

33.110.1 Usage

Legacy class. Use vtkCompositeDataGeometryFilter instead.

To create an instance of class vtkHierarchicalDataSetGeometryFilter, simply invoke its constructor as follows

```
obj = vtkHierarchicalDataSetGeometryFilter
```

33.110.2 Methods

The class vtkHierarchicalDataSetGeometryFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalDataSetGeometryFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHierarchicalDataSetGeometryFilter = obj.NewInstance ()`
- `vtkHierarchicalDataSetGeometryFilter = obj.SafeDownCast (vtkObject o)`

33.111 vtkHull

33.111.1 Usage

vtkHull is a filter which will produce an n-sided convex hull given a set of n planes. (The convex hull bounds the input polygonal data.) The hull is generated by squeezing the planes towards the input vtkPolyData, until the planes just touch the vtkPolyData. Then, the resulting planes are used to generate a polyhedron (i.e., hull) that is represented by triangles.

The n planes can be defined in a number of ways including 1) manually specifying each plane; 2) choosing the six face planes of the input's bounding box; 3) choosing the eight vertex planes of the input's bounding box; 4) choosing the twelve edge planes of the input's bounding box; and/or 5) using a recursively subdivided octahedron. Note that when specifying planes, the plane normals should point outside of the convex region.

The output of this filter can be used in combination with vtkLODActor to represent a levels-of-detail in the LOD hierarchy. Another use of this class is to manually specify the planes, and then generate the polyhedron from the planes (without squeezing the planes towards the input). The method GenerateHull() is used to do this.

To create an instance of class vtkHull, simply invoke its constructor as follows

```
obj = vtkHull
```

33.111.2 Methods

The class vtkHull has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHull class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHull = obj.NewInstance ()`
- `vtkHull = obj.SafeDownCast (vtkObject o)`
- `obj.RemoveAllPlanes (void )` - Remove all planes from the current set of planes.
- `int = obj.AddPlane (double A, double B, double C)` - Add a plane to the current set of planes. It will be added at the end of the list, and an index that can later be used to set this plane's normal will be returned. The values A, B, C are from the plane equation Ax + By + Cz + D = 0. This vector does not have to have unit length (but it must have a non-zero length!). If a value 0 ¿ i ¿= -NumberOfPlanes is returned, then the plane is parallel with a previously inserted plane, and —-i-1— is the index of the plane that was previously inserted. If a value i ¡ -NumberOfPlanes is returned, then the plane normal is zero length.
- `int = obj.AddPlane (double plane[3])` - Add a plane to the current set of planes. It will be added at the end of the list, and an index that can later be used to set this plane's normal will be returned. The values A, B, C are from the plane equation Ax + By + Cz + D = 0. This vector does not have to have unit length (but it must have a non-zero length!). If a value 0 ¿ i ¿= -NumberOfPlanes is returned, then the plane is parallel with a previously inserted plane, and —-i-1— is the index of the plane that was previously inserted. If a value i ¡ -NumberOfPlanes is returned, then the plane normal is zero length.
- `obj.SetPlane (int i, double A, double B, double C)` - Set the normal values for plane i. This is a plane that was already added to the current set of planes with AddPlane(), and is now being modified. The values A, B, C are from the plane equation Ax + By + Cz + D = 0. This vector does not have to have unit length. Note that D is set to zero, except in the case of the method taking a vtkPlanes* argument, where it is set to the D value defined there.

- `obj.SetPlane (int i, double plane[3])` - Set the normal values for plane i. This is a plane that was already added to the current set of planes with AddPlane(), and is now being modified. The values A, B, C are from the plane equation Ax + By + Cz + D = 0. This vector does not have to have unit length. Note that D is set to zero, except in the case of the method taking a vtkPlanes* argument, where it is set to the D value defined there.

- `int = obj.AddPlane (double A, double B, double C, double D)` - Variations of AddPlane()/SetPlane() that allow D to be set. These methods are used when GenerateHull() is used.

- `int = obj.AddPlane (double plane[3], double D)` - Variations of AddPlane()/SetPlane() that allow D to be set. These methods are used when GenerateHull() is used.

- `obj.SetPlane (int i, double A, double B, double C, double D)` - Variations of AddPlane()/SetPlane() that allow D to be set. These methods are used when GenerateHull() is used.

- `obj.SetPlane (int i, double plane[3], double D)` - Variations of AddPlane()/SetPlane() that allow D to be set. These methods are used when GenerateHull() is used.

- `obj.SetPlanes (vtkPlanes planes)` - Set all the planes at once using a vtkPlanes implicit function. This also sets the D value, so it can be used with GenerateHull().

- `int = obj.GetNumberOfPlanes ()` - Get the number of planes in the current set of planes.

- `obj.AddCubeVertexPlanes ()` - Add the 8 planes that represent the vertices of a cube - the combination of the three face planes connecting to a vertex - (1,1,1), (1,1,-1), (1,-1,1), (1,-1,1), (-1,1,1), (-1,1,-1), (-1,-1,1), (-1,-1-1).

- `obj.AddCubeEdgePlanes ()` - Add the 12 planes that represent the edges of a cube - halfway between the two connecting face planes - (1,1,0), (-1,-1,0), (-1,1,0), (1,-1,0), (0,1,1), (0,-1,-1), (0,1,-1), (0,-1,1), (1,0,1), (-1,0,-1), (1,0,-1), (-1,0,1)

- `obj.AddCubeFacePlanes ()` - Add the six planes that make up the faces of a cube - (1,0,0), (-1, 0, 0), (0,1,0), (0,-1,0), (0,0,1), (0,0,-1)

- `obj.AddRecursiveSpherePlanes (int level)` - Add the planes that represent the normals of the vertices of a polygonal sphere formed by recursively subdividing the triangles in an octahedron. Each triangle is subdivided by connecting the midpoints of the edges thus forming 4 smaller triangles. The level indicates how many subdivisions to do with a level of 0 used to add the 6 planes from the original octahedron, level 1 will add 18 planes, and so on.

- `obj.GenerateHull (vtkPolyData pd, double bounds)` - A special method that is used to generate a polyhedron directly from a set of n planes. The planes that are supplied by the user are not squeezed towards the input data (in fact the user need not specify an input). To use this method, you must provide an instance of vtkPolyData into which the points and cells defining the polyhedron are placed. You must also provide a bounding box where you expect the resulting polyhedron to lie. This can be a very generous fit, it's only used to create the initial polygons that are eventually clipped.

- `obj.GenerateHull (vtkPolyData pd, double xmin, double xmax, double ymin, double ymax, double zmin,` - A special method that is used to generate a polyhedron directly from a set of n planes. The planes that are supplied by the user are not squeezed towards the input data (in fact the user need not specify an input). To use this method, you must provide an instance of vtkPolyData into which the points and cells defining the polyhedron are placed. You must also provide a bounding box where you expect the resulting polyhedron to lie. This can be a very generous fit, it's only used to create the initial polygons that are eventually clipped.

33.112 vtkHyperOctreeContourFilter

33.112.1 Usage

vtkContourFilter is a filter that takes as input any dataset and generates on output isosurfaces and/or isolines. The exact form of the output depends upon the dimensionality of the input data. Data consisting of 3D cells will generate isosurfaces, data consisting of 2D cells will generate isolines, and data with 1D or 0D cells will generate isopoints. Combinations of output type are possible if the input dimension is mixed.

To use this filter you must specify one or more contour values. You can either use the method SetValue() to specify each contour value, or use GenerateValues() to generate a series of evenly spaced contours. It is also possible to accelerate the operation of this filter (at the cost of extra memory) by using a vtkScalarTree. A scalar tree is used to quickly locate cells that contain a contour surface. This is especially effective if multiple contours are being extracted. If you want to use a scalar tree, invoke the method UseScalarTreeOn().

To create an instance of class vtkHyperOctreeContourFilter, simply invoke its constructor as follows

```
obj = vtkHyperOctreeContourFilter
```

33.112.2 Methods

The class vtkHyperOctreeContourFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeContourFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeContourFilter = obj.NewInstance ()`
- `vtkHyperOctreeContourFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)` - Get the ith contour value.
- `double = obj.GetValue (int i)` - Get a pointer to an array of contour values. There will be GetNumberOfContours() values in the list.
- `obj.GetValues (double contourValues)` - Set the number of contours to place into the list. You only really need to use this method to reduce list size. The method SetValue() will automatically increase list size as needed.
- `obj.SetNumberOfContours (int number)` - Get the number of contours in the list of contour values.
- `int = obj.GetNumberOfContours ()` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double range[2])` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Modified GetMTime Because we delegate to vtkContourValues
- `long = obj.GetMTime ()` - Modified GetMTime Because we delegate to vtkContourValues
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.

33.113 vtkHyperOctreeCutter

33.113.1 Usage

vtkHyperOctreeCutter is a filter to cut through data using any subclass of vtkImplicitFunction. That is, a polygonal surface is created corresponding to the implicit function F(x,y,z) = value(s), where you can specify one or more values used to cut with.

In VTK, cutting means reducing a cell of dimension N to a cut surface of dimension N-1. For example, a tetrahedron when cut by a plane (i.e., vtkPlane implicit function) will generate triangles. (In comparison, clipping takes a N dimensional cell and creates N dimension primitives.)

vtkHyperOctreeCutter is generally used to "slice-through" a dataset, generating a surface that can be visualized. It is also possible to use vtkHyperOctreeCutter to do a form of volume rendering. vtkHyperOctreeCutter does this by generating multiple cut surfaces (usually planes) which are ordered (and rendered) from back-to-front. The surfaces are set translucent to give a volumetric rendering effect.

Note that data can be cut using either 1) the scalar values associated with the dataset or 2) an implicit function associated with this class. By default, if an implicit function is set it is used to cut the data set, otherwise the dataset scalars are used to perform the cut.

To create an instance of class vtkHyperOctreeCutter, simply invoke its constructor as follows

```
obj = vtkHyperOctreeCutter
```

33.113.2 Methods

The class vtkHyperOctreeCutter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeCutter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeCutter = obj.NewInstance ()`
- `vtkHyperOctreeCutter = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)` - Get the ith contour value.
- `double = obj.GetValue (int i)` - Get a pointer to an array of contour values. There will be GetNumberOfContours() values in the list.
- `obj.GetValues (double contourValues)` - Set the number of contours to place into the list. You only really need to use this method to reduce list size. The method SetValue() will automatically increase list size as needed.
- `obj.SetNumberOfContours (int number)` - Get the number of contours in the list of contour values.
- `int = obj.GetNumberOfContours ()` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double range[2])` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Override GetMTime because we delegate to vtkContourValues and refer to vtkImplicitFunction.
- `long = obj.GetMTime ()` - Override GetMTime because we delegate to vtkContourValues and refer to vtkImplicitFunction.
- `obj.SetCutFunction (vtkImplicitFunction )`

- `vtkImplicitFunction = obj.GetCutFunction ()`
- `obj.SetGenerateCutScalars (int )` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data.
- `int = obj.GetGenerateCutScalars ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data.
- `obj.GenerateCutScalarsOn ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data.
- `obj.GenerateCutScalarsOff ()` - If this flag is enabled, then the output scalar values will be interpolated from the implicit function values, and not the input scalar data.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Specify a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.SetSortBy (int )` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.
- `int = obj.GetSortByMinValue ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.
- `int = obj.GetSortByMaxValue ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.
- `int = obj.GetSortBy ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

  Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.
- `obj.SetSortByToSortByValue ()` - Set the sorting order for the generated polydata. There are two possibilities: Sort by value = 0 - This is the most efficient sort. For each cell, all contour values are processed. This is the default. Sort by cell = 1 - For each contour value, all cells are processed. This order should be used if the extracted polygons must be rendered in a back-to-front or front-to-back order. This is very problem dependent. For most applications, the default order is fine (and faster).

Sort by cell is going to have a problem if the input has 2D and 3D cells. Cell data will be scrambled becauses with vtkPolyData output, verts and lines have lower cell ids than triangles.

- `obj.SetSortByToSortByCell ()` - Return the sorting procedure as a descriptive character string.
- `string = obj.GetSortByAsString ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.

33.114 vtkHyperOctreeDepth

33.114.1 Usage

This filter returns a shallow copy of its input HyperOctree with a new data attribute field containing the depth of each cell.

To create an instance of class vtkHyperOctreeDepth, simply invoke its constructor as follows

```
obj = vtkHyperOctreeDepth
```

33.114.2 Methods

The class vtkHyperOctreeDepth has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeDepth class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeDepth = obj.NewInstance ()`
- `vtkHyperOctreeDepth = obj.SafeDownCast (vtkObject o)`

33.115 vtkHyperOctreeDualGridContourFilter

33.115.1 Usage

use of unsigned short to hold level index limits tree depth to 16.

To use this filter you must specify one or more contour values. You can either use the method SetValue() to specify each contour value, or use GenerateValues() to generate a series of evenly spaced contours. It is also possible to accelerate the operation of this filter (at the cost of extra memory) by using a vtkScalarTree. A scalar tree is used to quickly locate cells that contain a contour surface. This is especially effective if multiple contours are being extracted. If you want to use a scalar tree, invoke the method UseScalarTreeOn().

To create an instance of class vtkHyperOctreeDualGridContourFilter, simply invoke its constructor as follows

```
obj = vtkHyperOctreeDualGridContourFilter
```

33.115.2 Methods

The class vtkHyperOctreeDualGridContourFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeDualGridContourFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeDualGridContourFilter = obj.NewInstance ()`
- `vtkHyperOctreeDualGridContourFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)` - Get the ith contour value.
- `double = obj.GetValue (int i)` - Get a pointer to an array of contour values. There will be GetNumberOfContours() values in the list.
- `obj.GetValues (double contourValues)` - Set the number of contours to place into the list. You only really need to use this method to reduce list size. The method SetValue() will automatically increase list size as needed.
- `obj.SetNumberOfContours (int number)` - Get the number of contours in the list of contour values.
- `int = obj.GetNumberOfContours ()` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double range[2])` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Modified GetMTime Because we delegate to vtkContourValues
- `long = obj.GetMTime ()` - Modified GetMTime Because we delegate to vtkContourValues
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.

33.116 vtkHyperOctreeFractalSource

33.116.1 Usage

To create an instance of class vtkHyperOctreeFractalSource, simply invoke its constructor as follows

```
obj = vtkHyperOctreeFractalSource
```

33.116.2 Methods

The class vtkHyperOctreeFractalSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeFractalSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeFractalSource = obj.NewInstance ()`
- `vtkHyperOctreeFractalSource = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetMaximumLevel ()` - Return the maximum number of levels of the hyperoctree.
- `obj.SetMaximumLevel (int levels)` - Set the maximum number of levels of the hyperoctree. If GetMinLevels()¿=levels, GetMinLevels() is changed to levels-1.
- `obj.SetMinimumLevel (int level)` - Return the minimal number of levels of systematic subdivision.
- `int = obj.GetMinimumLevel ()` - Return the minimal number of levels of systematic subdivision.
- `obj.SetProjectionAxes (int x, int y, int z)` - Set the projection from the 4D space (4 parameters / 2 imaginary numbers) to the axes of the 3D Volume. 0=C_Real, 1=C_Imaginary, 2=X_Real, 4=X_Imaginary
- `obj.SetProjectionAxes (int a[3])` - Set the projection from the 4D space (4 parameters / 2 imaginary numbers) to the axes of the 3D Volume. 0=C_Real, 1=C_Imaginary, 2=X_Real, 4=X_Imaginary
- `int = obj. GetProjectionAxes ()` - Set the projection from the 4D space (4 parameters / 2 imaginary numbers) to the axes of the 3D Volume. 0=C_Real, 1=C_Imaginary, 2=X_Real, 4=X_Imaginary
- `obj.SetOriginCX (double , double , double , double )` - Imaginary and real value for C (constant in equation) and X (initial value).
- `obj.SetOriginCX (double a[4])` - Imaginary and real value for C (constant in equation) and X (initial value).
- `double = obj. GetOriginCX ()` - Imaginary and real value for C (constant in equation) and X (initial value).
- `obj.SetSizeCX (double , double , double , double )` - Just a different way of setting the sample. This sets the size of the 4D volume. SampleCX is computed from size and extent. Size is ignored when a dimension i 0 (collapsed).
- `obj.SetSizeCX (double a[4])` - Just a different way of setting the sample. This sets the size of the 4D volume. SampleCX is computed from size and extent. Size is ignored when a dimension i 0 (collapsed).
- `double = obj. GetSizeCX ()` - Just a different way of setting the sample. This sets the size of the 4D volume. SampleCX is computed from size and extent. Size is ignored when a dimension i 0 (collapsed).
- `obj.SetMaximumNumberOfIterations (short )` - The maximum number of cycles run to see if the value goes over 2
- `GetMaximumNumberOfIterationsMinValue = obj.()` - The maximum number of cycles run to see if the value goes over 2
- `GetMaximumNumberOfIterationsMaxValue = obj.()` - The maximum number of cycles run to see if the value goes over 2

- `char = obj.GetMaximumNumberOfIterations ()` - The maximum number of cycles run to see if the value goes over 2
- `obj.SetDimension (int )` - Create a 2D or 3D fractal.
- `int = obj.GetDimensionMinValue ()` - Create a 2D or 3D fractal.
- `int = obj.GetDimensionMaxValue ()` - Create a 2D or 3D fractal.
- `int = obj.GetDimension ()` - Create a 2D or 3D fractal.
- `obj.SetSpanThreshold (double )` - Controls when a leaf gets subdivided. If the corner values span a larger range than this value, the leaf is subdivided. This defaults to 2.
- `double = obj.GetSpanThreshold ()` - Controls when a leaf gets subdivided. If the corner values span a larger range than this value, the leaf is subdivided. This defaults to 2.

33.117 vtkHyperOctreeLimiter

33.117.1 Usage

This filter returns a lower resolution copy of its input vtkHyperOctree. It does a length/area/volume weighted averaging to obtain data at each cut point. Above the cut level, leaf attribute data is simply copied.

To create an instance of class vtkHyperOctreeLimiter, simply invoke its constructor as follows

```
obj = vtkHyperOctreeLimiter
```

33.117.2 Methods

The class vtkHyperOctreeLimiter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeLimiter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeLimiter = obj.NewInstance ()`
- `vtkHyperOctreeLimiter = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetMaximumLevel ()` - Return the maximum number of levels of the hyperoctree.
- `obj.SetMaximumLevel (int levels)` - Set the maximum number of levels of the hyperoctree.

33.118 vtkHyperOctreeSampleFunction

33.118.1 Usage

vtkHyperOctreeSampleFunction is a source object that evaluates an implicit function to drive the subdivision process. The user can specify the threshold over which a subdivision occurs, the maximum and minimum level of subdivisions and the dimension of the hyperoctree.

To create an instance of class vtkHyperOctreeSampleFunction, simply invoke its constructor as follows

```
obj = vtkHyperOctreeSampleFunction
```

33.118.2 Methods

The class vtkHyperOctreeSampleFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeSampleFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeSampleFunction = obj.NewInstance ()`
- `vtkHyperOctreeSampleFunction = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetLevels ()` - Return the maximum number of levels of the hyperoctree.
- `obj.SetLevels (int levels)` - Set the maximum number of levels of the hyperoctree. If GetMinLevels()¿=levels, GetMinLevels() is changed to levels-1.
- `int = obj.GetMinLevels ()` - Return the minimal number of levels of systematic subdivision.
- `obj.SetMinLevels (int minLevels)` - Set the minimal number of levels of systematic subdivision.
- `double = obj.GetThreshold ()` - Return the threshold over which a subdivision is required.
- `obj.SetThreshold (double threshold)` - Set the threshold over which a subdivision is required.
- `int = obj.GetDimension ()` - Return the dimension of the tree (1D:binary tree(2 children), 2D:quadtree(4 children), 3D:octree (8 children))
- `obj.SetDimension (int dim)`
- `obj.SetSize (double , double , double )` - Set the size on each axis.
- `obj.SetSize (double  a[3])` - Set the size on each axis.
- `double = obj. GetSize ()` - Return the size on each axis.
- `obj.SetOrigin (double , double , double )` - Set the origin (position of corner (0,0,0) of the root.
- `obj.SetOrigin (double  a[3])` - Set the origin (position of corner (0,0,0) of the root.
- `double = obj. GetOrigin ()` - Set the origin (position of corner (0,0,0) of the root. Return the origin (position of corner (0,0,0)) of the root.
- `double = obj.GetWidth ()` - Return the length along the x-axis.
- `obj.SetWidth (double width)` - Set the length along the x-axis.
- `double = obj.GetHeight ()` - Return the length along the y-axis. Relevant only if GetDimension()¿=2
- `obj.SetHeight (double height)` - Set the length along the y-axis. Relevant only if GetDimension()¿=2
- `double = obj.GetDepth ()` - Return the length along the z-axis. Relevant only if GetDimension()¿=3
- `obj.SetDepth (double depth)` - Return the length along the z-axis. Relevant only if GetDimension()¿=3
- `obj.SetImplicitFunction (vtkImplicitFunction )` - Specify the implicit function to use to generate data.

- `vtkImplicitFunction = obj.GetImplicitFunction ()` - Specify the implicit function to use to generate data.
- `obj.SetOutputScalarType (int )` - Set what type of scalar data this source should generate.
- `int = obj.GetOutputScalarType ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToDouble ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToFloat ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToLong ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedLong ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToInt ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedInt ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToShort ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedShort ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToChar ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedChar ()` - Return the MTime also considering the implicit function.
- `long = obj.GetMTime ()` - Return the MTime also considering the implicit function.

33.119 vtkHyperOctreeSurfaceFilter

33.119.1 Usage

vtkHyperOctreeSurfaceFilter extracts the surface of an hyperoctree.

To create an instance of class vtkHyperOctreeSurfaceFilter, simply invoke its constructor as follows

```
obj = vtkHyperOctreeSurfaceFilter
```

33.119.2 Methods

The class vtkHyperOctreeSurfaceFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeSurfaceFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeSurfaceFilter = obj.NewInstance ()`
- `vtkHyperOctreeSurfaceFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetMerging (int )` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.

- `int = obj.GetMerging ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.MergingOn ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.MergingOff ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `long = obj.GetMTime ()` - Return the MTime also considering the locator.
- `obj.SetPassThroughCellIds (int )` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory.
- `int = obj.GetPassThroughCellIds ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory.
- `obj.PassThroughCellIdsOn ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory.
- `obj.PassThroughCellIdsOff ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for cell picking. The default is off to conserve memory.

33.120 vtkHyperOctreeToUniformGridFilter

33.120.1 Usage

vtkHyperOctreeToUniformGridFilter creates a uniform grid with a resolution based on the number of levels of the hyperoctree. Then, it copies celldata in each cell of the uniform grid that belongs to an actual leaf of the hyperoctree.

To create an instance of class vtkHyperOctreeToUniformGridFilter, simply invoke its constructor as follows

```
obj = vtkHyperOctreeToUniformGridFilter
```

33.120.2 Methods

The class vtkHyperOctreeToUniformGridFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperOctreeToUniformGridFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperOctreeToUniformGridFilter = obj.NewInstance ()`
- `vtkHyperOctreeToUniformGridFilter = obj.SafeDownCast (vtkObject o)`

33.121 vtkHyperStreamline

33.121.1 Usage

vtkHyperStreamline is a filter that integrates through a tensor field to generate a hyperstreamline. The integration is along the maximum eigenvector and the cross section of the hyperstreamline is defined by the two other eigenvectors. Thus the shape of the hyperstreamline is "tube-like", with the cross section being elliptical. Hyperstreamlines are used to visualize tensor fields.

The starting point of a hyperstreamline can be defined in one of two ways. First, you may specify an initial position. This is a x-y-z global coordinate. The second option is to specify a starting location. This is cellId, subId, and cell parametric coordinates.

The integration of the hyperstreamline occurs through the major eigenvector field. IntegrationStepLength controls the step length within each cell (i.e., this is the fraction of the cell length). The length of the hyperstreamline is controlled by MaximumPropagationDistance. This parameter is the length of the hyperstreamline in units of distance. The tube itself is composed of many small sub-tubes - NumberOfSides controls the number of sides in the tube, and StepLength controls the length of the sub-tubes.

Because hyperstreamlines are often created near regions of singularities, it is possible to control the scaling of the tube cross section by using a logarithmic scale. Use LogScalingOn to turn this capability on. The Radius value controls the initial radius of the tube.

To create an instance of class vtkHyperStreamline, simply invoke its constructor as follows

```
obj = vtkHyperStreamline
```

33.121.2 Methods

The class vtkHyperStreamline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHyperStreamline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHyperStreamline = obj.NewInstance ()`
- `vtkHyperStreamline = obj.SafeDownCast (vtkObject o)`
- `obj.SetStartLocation (vtkIdType cellId, int subId, double pcoords[3])` - Specify the start of the hyperstreamline in the cell coordinate system. That is, cellId and subId (if composite cell), and parametric coordinates.
- `obj.SetStartLocation (vtkIdType cellId, int subId, double r, double s, double t)` - Specify the start of the hyperstreamline in the cell coordinate system. That is, cellId and subId (if composite cell), and parametric coordinates.
- `obj.SetStartPosition (double x[3])` - Specify the start of the hyperstreamline in the global coordinate system. Starting from position implies that a search must be performed to find initial cell to start integration from.
- `obj.SetStartPosition (double x, double y, double z)` - Specify the start of the hyperstreamline in the global coordinate system. Starting from position implies that a search must be performed to find initial cell to start integration from.
- `double = obj.GetStartPosition ()` - Get the start position of the hyperstreamline in global x-y-z coordinates.
- `obj.SetMaximumPropagationDistance (double )` - Set / get the maximum length of the hyperstreamline expressed as absolute distance (i.e., arc length) value.

- `double = obj.GetMaximumPropagationDistanceMinValue ()` - Set / get the maximum length of the hyperstreamline expressed as absolute distance (i.e., arc length) value.
- `double = obj.GetMaximumPropagationDistanceMaxValue ()` - Set / get the maximum length of the hyperstreamline expressed as absolute distance (i.e., arc length) value.
- `double = obj.GetMaximumPropagationDistance ()` - Set / get the maximum length of the hyperstreamline expressed as absolute distance (i.e., arc length) value.
- `obj.SetIntegrationEigenvector (int )` - Set / get the eigenvector field through which to ingrate. It is possible to integrate using the major, medium or minor eigenvector field. The major eigenvector is the eigenvector whose corresponding eigenvalue is closest to positive infinity. The minor eigenvector is the eigenvector whose corresponding eigenvalue is closest to negative infinity. The medium eigenvector is the eigenvector whose corresponding eigenvalue is between the major and minor eigenvalues.
- `int = obj.GetIntegrationEigenvectorMinValue ()` - Set / get the eigenvector field through which to ingrate. It is possible to integrate using the major, medium or minor eigenvector field. The major eigenvector is the eigenvector whose corresponding eigenvalue is closest to positive infinity. The minor eigenvector is the eigenvector whose corresponding eigenvalue is closest to negative infinity. The medium eigenvector is the eigenvector whose corresponding eigenvalue is between the major and minor eigenvalues.
- `int = obj.GetIntegrationEigenvectorMaxValue ()` - Set / get the eigenvector field through which to ingrate. It is possible to integrate using the major, medium or minor eigenvector field. The major eigenvector is the eigenvector whose corresponding eigenvalue is closest to positive infinity. The minor eigenvector is the eigenvector whose corresponding eigenvalue is closest to negative infinity. The medium eigenvector is the eigenvector whose corresponding eigenvalue is between the major and minor eigenvalues.
- `int = obj.GetIntegrationEigenvector ()` - Set / get the eigenvector field through which to ingrate. It is possible to integrate using the major, medium or minor eigenvector field. The major eigenvector is the eigenvector whose corresponding eigenvalue is closest to positive infinity. The minor eigenvector is the eigenvector whose corresponding eigenvalue is closest to negative infinity. The medium eigenvector is the eigenvector whose corresponding eigenvalue is between the major and minor eigenvalues.
- `obj.SetIntegrationEigenvectorToMajor ()` - Set / get the eigenvector field through which to ingrate. It is possible to integrate using the major, medium or minor eigenvector field. The major eigenvector is the eigenvector whose corresponding eigenvalue is closest to positive infinity. The minor eigenvector is the eigenvector whose corresponding eigenvalue is closest to negative infinity. The medium eigenvector is the eigenvector whose corresponding eigenvalue is between the major and minor eigenvalues.
- `obj.SetIntegrationEigenvectorToMedium ()` - Set / get the eigenvector field through which to ingrate. It is possible to integrate using the major, medium or minor eigenvector field. The major eigenvector is the eigenvector whose corresponding eigenvalue is closest to positive infinity. The minor eigenvector is the eigenvector whose corresponding eigenvalue is closest to negative infinity. The medium eigenvector is the eigenvector whose corresponding eigenvalue is between the major and minor eigenvalues.
- `obj.SetIntegrationEigenvectorToMinor ()` - Set / get the eigenvector field through which to ingrate. It is possible to integrate using the major, medium or minor eigenvector field. The major eigenvector is the eigenvector whose corresponding eigenvalue is closest to positive infinity. The minor eigenvector is the eigenvector whose corresponding eigenvalue is closest to negative infinity. The medium eigenvector is the eigenvector whose corresponding eigenvalue is between the major and minor eigenvalues.
- `obj.IntegrateMajorEigenvector ()` - Use the major eigenvector field as the vector field through which to integrate. The major eigenvector is the eigenvector whose corresponding eigenvalue is closest to positive infinity.

- `obj.IntegrateMediumEigenvector ()` - Use the medium eigenvector field as the vector field through which to integrate. The medium eigenvector is the eigenvector whose corresponding eigenvalue is between the major and minor eigenvalues.
- `obj.IntegrateMinorEigenvector ()` - Use the minor eigenvector field as the vector field through which to integrate. The minor eigenvector is the eigenvector whose corresponding eigenvalue is closest to negative infinity.
- `obj.SetIntegrationStepLength (double )` - Set / get a nominal integration step size (expressed as a fraction of the size of each cell).
- `double = obj.GetIntegrationStepLengthMinValue ()` - Set / get a nominal integration step size (expressed as a fraction of the size of each cell).
- `double = obj.GetIntegrationStepLengthMaxValue ()` - Set / get a nominal integration step size (expressed as a fraction of the size of each cell).
- `double = obj.GetIntegrationStepLength ()` - Set / get a nominal integration step size (expressed as a fraction of the size of each cell).
- `obj.SetStepLength (double )` - Set / get the length of a tube segment composing the hyperstreamline. The length is specified as a fraction of the diagonal length of the input bounding box.
- `double = obj.GetStepLengthMinValue ()` - Set / get the length of a tube segment composing the hyperstreamline. The length is specified as a fraction of the diagonal length of the input bounding box.
- `double = obj.GetStepLengthMaxValue ()` - Set / get the length of a tube segment composing the hyperstreamline. The length is specified as a fraction of the diagonal length of the input bounding box.
- `double = obj.GetStepLength ()` - Set / get the length of a tube segment composing the hyperstreamline. The length is specified as a fraction of the diagonal length of the input bounding box.
- `obj.SetIntegrationDirection (int )` - Specify the direction in which to integrate the hyperstreamline.
- `int = obj.GetIntegrationDirectionMinValue ()` - Specify the direction in which to integrate the hyperstreamline.
- `int = obj.GetIntegrationDirectionMaxValue ()` - Specify the direction in which to integrate the hyperstreamline.
- `int = obj.GetIntegrationDirection ()` - Specify the direction in which to integrate the hyperstreamline.
- `obj.SetIntegrationDirectionToForward ()` - Specify the direction in which to integrate the hyperstreamline.
- `obj.SetIntegrationDirectionToBackward ()` - Specify the direction in which to integrate the hyperstreamline.
- `obj.SetIntegrationDirectionToIntegrateBothDirections ()` - Specify the direction in which to integrate the hyperstreamline.
- `obj.SetTerminalEigenvalue (double )` - Set/get terminal eigenvalue. If major eigenvalue falls below this value, hyperstreamline terminates propagation.
- `double = obj.GetTerminalEigenvalueMinValue ()` - Set/get terminal eigenvalue. If major eigenvalue falls below this value, hyperstreamline terminates propagation.

- `double = obj.GetTerminalEigenvalueMaxValue ()` - Set/get terminal eigenvalue. If major eigenvalue falls below this value, hyperstreamline terminates propagation.
- `double = obj.GetTerminalEigenvalue ()` - Set/get terminal eigenvalue. If major eigenvalue falls below this value, hyperstreamline terminates propagation.
- `obj.SetNumberOfSides (int )` - Set / get the number of sides for the hyperstreamlines. At a minimum, number of sides is 3.
- `int = obj.GetNumberOfSidesMinValue ()` - Set / get the number of sides for the hyperstreamlines. At a minimum, number of sides is 3.
- `int = obj.GetNumberOfSidesMaxValue ()` - Set / get the number of sides for the hyperstreamlines. At a minimum, number of sides is 3.
- `int = obj.GetNumberOfSides ()` - Set / get the number of sides for the hyperstreamlines. At a minimum, number of sides is 3.
- `obj.SetRadius (double )` - Set / get the initial tube radius. This is the maximum "elliptical" radius at the beginning of the tube. Radius varies based on ratio of eigenvalues. Note that tube section is actually elliptical and may become a point or line in cross section in some cases.
- `double = obj.GetRadiusMinValue ()` - Set / get the initial tube radius. This is the maximum "elliptical" radius at the beginning of the tube. Radius varies based on ratio of eigenvalues. Note that tube section is actually elliptical and may become a point or line in cross section in some cases.
- `double = obj.GetRadiusMaxValue ()` - Set / get the initial tube radius. This is the maximum "elliptical" radius at the beginning of the tube. Radius varies based on ratio of eigenvalues. Note that tube section is actually elliptical and may become a point or line in cross section in some cases.
- `double = obj.GetRadius ()` - Set / get the initial tube radius. This is the maximum "elliptical" radius at the beginning of the tube. Radius varies based on ratio of eigenvalues. Note that tube section is actually elliptical and may become a point or line in cross section in some cases.
- `obj.SetLogScaling (int )` - Turn on/off logarithmic scaling. If scaling is on, the log base 10 of the computed eigenvalues are used to scale the cross section radii.
- `int = obj.GetLogScaling ()` - Turn on/off logarithmic scaling. If scaling is on, the log base 10 of the computed eigenvalues are used to scale the cross section radii.
- `obj.LogScalingOn ()` - Turn on/off logarithmic scaling. If scaling is on, the log base 10 of the computed eigenvalues are used to scale the cross section radii.
- `obj.LogScalingOff ()` - Turn on/off logarithmic scaling. If scaling is on, the log base 10 of the computed eigenvalues are used to scale the cross section radii.

33.122 vtkIconGlyphFilter

33.122.1 Usage

vtkIconGlyphFilter takes in a vtkPointSet where each point corresponds to the center of an icon. Scalar integer data must also be set to give each point an icon index. This index is a zero based row major index into an image that contains a grid of icons. You must also set pixel Size of the icon image and the size of a particular icon.

To create an instance of class vtkIconGlyphFilter, simply invoke its constructor as follows

```
obj = vtkIconGlyphFilter
```

33.122.2 Methods

The class vtkIconGlyphFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIconGlyphFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIconGlyphFilter = obj.NewInstance ()`
- `vtkIconGlyphFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetIconSize (int , int )` - Specify the Width and Height, in pixels, of an icon in the icon sheet
- `obj.SetIconSize (int  a[2])` - Specify the Width and Height, in pixels, of an icon in the icon sheet
- `int = obj. GetIconSize ()` - Specify the Width and Height, in pixels, of an icon in the icon sheet
- `obj.SetIconSheetSize (int , int )` - Specify the Width and Height, in pixels, of an icon in the icon sheet
- `obj.SetIconSheetSize (int  a[2])` - Specify the Width and Height, in pixels, of an icon in the icon sheet
- `int = obj. GetIconSheetSize ()` - Specify the Width and Height, in pixels, of an icon in the icon sheet
- `obj.SetUseIconSize (bool b)` - Specify whether the Quad generated to place the icon on will be either 1 x 1 or the dimensions specified by IconSize.
- `bool = obj.GetUseIconSize ()` - Specify whether the Quad generated to place the icon on will be either 1 x 1 or the dimensions specified by IconSize.
- `obj.UseIconSizeOn ()` - Specify whether the Quad generated to place the icon on will be either 1 x 1 or the dimensions specified by IconSize.
- `obj.UseIconSizeOff ()` - Specify whether the Quad generated to place the icon on will be either 1 x 1 or the dimensions specified by IconSize.
- `obj.SetGravity (int )` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.
- `int = obj.GetGravity ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.
- `obj.SetGravityToTopRight ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.
- `obj.SetGravityToTopCenter ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.
- `obj.SetGravityToTopLeft ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.

- `obj.SetGravityToCenterRight ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.
- `obj.SetGravityToCenterCenter ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.
- `obj.SetGravityToCenterLeft ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.
- `obj.SetGravityToBottomRight ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.
- `obj.SetGravityToBottomCenter ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.
- `obj.SetGravityToBottomLeft ()` - Specify if the input points define the center of the icon quad or one of top right corner, top center, top left corner, center right, center, center center left, bottom right corner, bottom center or bottom left corner.

33.123 vtkIdFilter

33.123.1 Usage

vtkIdFilter is a filter to that generates scalars or field data using cell and point ids. That is, the point attribute data scalars or field data are generated from the point ids, and the cell attribute data scalars or field data are generated from the the cell ids.

Typically this filter is used with vtkLabeledDataMapper (and possibly vtkSelectVisiblePoints) to create labels for points and cells, or labels for the point or cell data scalar values.

To create an instance of class vtkIdFilter, simply invoke its constructor as follows

```
obj = vtkIdFilter
```

33.123.2 Methods

The class vtkIdFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIdFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIdFilter = obj.NewInstance ()`
- `vtkIdFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetPointIds (int )` - Enable/disable the generation of point ids. Default is on.
- `int = obj.GetPointIds ()` - Enable/disable the generation of point ids. Default is on.
- `obj.PointIdsOn ()` - Enable/disable the generation of point ids. Default is on.
- `obj.PointIdsOff ()` - Enable/disable the generation of point ids. Default is on.

- `obj.SetCellIds (int )` - Enable/disable the generation of point ids. Default is on.
- `int = obj.GetCellIds ()` - Enable/disable the generation of point ids. Default is on.
- `obj.CellIdsOn ()` - Enable/disable the generation of point ids. Default is on.
- `obj.CellIdsOff ()` - Enable/disable the generation of point ids. Default is on.
- `obj.SetFieldData (int )` - Set/Get the flag which controls whether to generate scalar data or field data. If this flag is off, scalar data is generated. Otherwise, field data is generated. Default is off.
- `int = obj.GetFieldData ()` - Set/Get the flag which controls whether to generate scalar data or field data. If this flag is off, scalar data is generated. Otherwise, field data is generated. Default is off.
- `obj.FieldDataOn ()` - Set/Get the flag which controls whether to generate scalar data or field data. If this flag is off, scalar data is generated. Otherwise, field data is generated. Default is off.
- `obj.FieldDataOff ()` - Set/Get the flag which controls whether to generate scalar data or field data. If this flag is off, scalar data is generated. Otherwise, field data is generated. Default is off.
- `obj.SetIdsArrayName (string )` - Set/Get the name of the Ids array if generated. By default the Ids are named "vtkIdFilter_Ids", but this can be changed with this function.
- `string = obj.GetIdsArrayName ()` - Set/Get the name of the Ids array if generated. By default the Ids are named "vtkIdFilter_Ids", but this can be changed with this function.

33.124 vtkImageDataGeometryFilter

33.124.1 Usage

vtkImageDataGeometryFilter is a filter that extracts geometry from a structured points dataset. By specifying appropriate i-j-k indices (via the "Extent" instance variable), it is possible to extract a point, a line, a plane (i.e., image), or a "volume" from dataset. (Since the output is of type polydata, the volume is actually a (n x m x o) region of points.)

The extent specification is zero-offset. That is, the first k-plane in a 50x50x50 volume is given by (0,49, 0,49, 0,0).

To create an instance of class vtkImageDataGeometryFilter, simply invoke its constructor as follows

```
obj = vtkImageDataGeometryFilter
```

33.124.2 Methods

The class vtkImageDataGeometryFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageDataGeometryFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageDataGeometryFilter = obj.NewInstance ()`
- `vtkImageDataGeometryFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetExtent (int extent[6])` - Set / get the extent (imin,imax, jmin,jmax, kmin,kmax) indices.
- `obj.SetExtent (int iMin, int iMax, int jMin, int jMax, int kMin, int kMax)` - Set / get the extent (imin,imax, jmin,jmax, kmin,kmax) indices.

- `obj.SetThresholdCells (int )` - Set ThresholdCells to true if you wish to skip any voxel/pixels which have scalar values less than the specified threshold. Currently this functionality is only implemented for 2D imagedata
- `int = obj.GetThresholdCells ()` - Set ThresholdCells to true if you wish to skip any voxel/pixels which have scalar values less than the specified threshold. Currently this functionality is only implemented for 2D imagedata
- `obj.ThresholdCellsOn ()` - Set ThresholdCells to true if you wish to skip any voxel/pixels which have scalar values less than the specified threshold. Currently this functionality is only implemented for 2D imagedata
- `obj.ThresholdCellsOff ()` - Set ThresholdCells to true if you wish to skip any voxel/pixels which have scalar values less than the specified threshold. Currently this functionality is only implemented for 2D imagedata
- `obj.SetThresholdValue (double )` - Set ThresholdValue to the scalar value by which to threshhold cells when extracting geometry when ThresholdCells is true. Cells with scalar values greater than the threshold will be output.
- `double = obj.GetThresholdValue ()` - Set ThresholdValue to the scalar value by which to threshhold cells when extracting geometry when ThresholdCells is true. Cells with scalar values greater than the threshold will be output.
- `obj.ThresholdValueOn ()` - Set ThresholdValue to the scalar value by which to threshhold cells when extracting geometry when ThresholdCells is true. Cells with scalar values greater than the threshold will be output.
- `obj.ThresholdValueOff ()` - Set ThresholdValue to the scalar value by which to threshhold cells when extracting geometry when ThresholdCells is true. Cells with scalar values greater than the threshold will be output.
- `obj.SetOutputTriangles (int )` - Set OutputTriangles to true if you wish to generate triangles instead of quads when extracting cells from 2D imagedata Currently this functionality is only implemented for 2D imagedata
- `int = obj.GetOutputTriangles ()` - Set OutputTriangles to true if you wish to generate triangles instead of quads when extracting cells from 2D imagedata Currently this functionality is only implemented for 2D imagedata
- `obj.OutputTrianglesOn ()` - Set OutputTriangles to true if you wish to generate triangles instead of quads when extracting cells from 2D imagedata Currently this functionality is only implemented for 2D imagedata
- `obj.OutputTrianglesOff ()` - Set OutputTriangles to true if you wish to generate triangles instead of quads when extracting cells from 2D imagedata Currently this functionality is only implemented for 2D imagedata

33.125 vtkImageMarchingCubes

33.125.1 Usage

vtkImageMarchingCubes is a filter that takes as input images (e.g., 3D image region) and generates on output one or more isosurfaces. One or more contour values must be specified to generate the isosurfaces. Alternatively, you can specify a min/max scalar range and the number of contours to generate a series of evenly spaced contour values. This filter can stream, so that the entire volume need not be loaded at once. Streaming is controlled using the instance variable InputMemoryLimit, which has units KBytes.

To create an instance of class vtkImageMarchingCubes, simply invoke its constructor as follows

```
obj = vtkImageMarchingCubes
```

33.125.2 Methods

The class vtkImageMarchingCubes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMarchingCubes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMarchingCubes = obj.NewInstance ()`
- `vtkImageMarchingCubes = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)` - Methods to set contour values
- `double = obj.GetValue (int i)` - Methods to set contour values
- `obj.GetValues (double contourValues)` - Methods to set contour values
- `obj.SetNumberOfContours (int number)` - Methods to set contour values
- `int = obj.GetNumberOfContours ()` - Methods to set contour values
- `obj.GenerateValues (int numContours, double range[2])` - Methods to set contour values
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Methods to set contour values
- `long = obj.GetMTime ()` - Because we delegate to vtkContourValues & refer to vtkImplicitFunction
- `obj.SetComputeScalars (int )` - Set/Get the computation of scalars.
- `int = obj.GetComputeScalars ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOn ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOff ()` - Set/Get the computation of scalars.
- `obj.SetComputeNormals (int )` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeNormals ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOn ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOff ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeGradients (int )` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.

- `int = obj.GetComputeGradients ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOn ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOff ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetLocatorPoint (int cellX, int cellY, int edge)`
- `obj.AddLocatorPoint (int cellX, int cellY, int edge, int ptId)`
- `obj.IncrementLocatorZ ()`
- `obj.SetInputMemoryLimit (int )` - The InputMemoryLimit determines the chunk size (the number of slices requested at each iteration). The units of this limit is KiloBytes. For now, only the Z axis is split.
- `int = obj.GetInputMemoryLimit ()` - The InputMemoryLimit determines the chunk size (the number of slices requested at each iteration). The units of this limit is KiloBytes. For now, only the Z axis is split.

33.126 vtkImplicitTextureCoords

33.126.1 Usage

vtkImplicitTextureCoords is a filter to generate 1D, 2D, or 3D texture coordinates from one, two, or three implicit functions, respectively. In combinations with a vtkBooleanTexture map (or another texture map of your own creation), the texture coordinates can be used to highlight (via color or intensity) or cut (via transparency) dataset geometry without any complex geometric processing. (Note: the texture coordinates are referred to as r-s-t coordinates.)

The texture coordinates are automatically normalized to lie between (0,1). Thus, no matter what the implicit functions evaluate to, the resulting texture coordinates lie between (0,1), with the zero implicit function value mapped to the 0.5 texture coordinates value. Depending upon the maximum negative/positive implicit function values, the full (0,1) range may not be occupied (i.e., the positive/negative ranges are mapped using the same scale factor).

A boolean variable InvertTexture is available to flip the texture coordinates around 0.5 (value 1.0 becomes 0.0, 0.25-¿0.75). This is equivalent to flipping the texture map (but a whole lot easier).

To create an instance of class vtkImplicitTextureCoords, simply invoke its constructor as follows

```
obj = vtkImplicitTextureCoords
```

33.126.2 Methods

The class vtkImplicitTextureCoords has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitTextureCoords class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkImplicitTextureCoords = obj.NewInstance ()`
- `vtkImplicitTextureCoords = obj.SafeDownCast (vtkObject o)`
- `obj.SetRFunction (vtkImplicitFunction )` - Specify an implicit function to compute the r texture coordinate.
- `vtkImplicitFunction = obj.GetRFunction ()` - Specify an implicit function to compute the r texture coordinate.
- `obj.SetSFunction (vtkImplicitFunction )` - Specify an implicit function to compute the s texture coordinate.
- `vtkImplicitFunction = obj.GetSFunction ()` - Specify an implicit function to compute the s texture coordinate.
- `obj.SetTFunction (vtkImplicitFunction )` - Specify an implicit function to compute the t texture coordinate.
- `vtkImplicitFunction = obj.GetTFunction ()` - Specify an implicit function to compute the t texture coordinate.
- `obj.SetFlipTexture (int )` - If enabled, this will flip the sense of inside and outside the implicit function (i.e., a rotation around the r-s-t=0.5 axis).
- `int = obj.GetFlipTexture ()` - If enabled, this will flip the sense of inside and outside the implicit function (i.e., a rotation around the r-s-t=0.5 axis).
- `obj.FlipTextureOn ()` - If enabled, this will flip the sense of inside and outside the implicit function (i.e., a rotation around the r-s-t=0.5 axis).
- `obj.FlipTextureOff ()` - If enabled, this will flip the sense of inside and outside the implicit function (i.e., a rotation around the r-s-t=0.5 axis).

33.127 vtkInterpolateDataSetAttributes

33.127.1 Usage

vtkInterpolateDataSetAttributes is a filter that interpolates data set attribute values between input data sets. The input to the filter must be datasets of the same type, same number of cells, and same number of points. The output of the filter is a data set of the same type as the input dataset and whose attribute values have been interpolated at the parametric value specified.

The filter is used by specifying two or more input data sets (total of N), and a parametric value t (0 ¡= t ¡= N-1). The output will contain interpolated data set attributes common to all input data sets. (For example, if one input has scalars and vectors, and another has just scalars, then only scalars will be interpolated and output.)

To create an instance of class vtkInterpolateDataSetAttributes, simply invoke its constructor as follows

```
obj = vtkInterpolateDataSetAttributes
```

33.127.2 Methods

The class vtkInterpolateDataSetAttributes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInterpolateDataSetAttributes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInterpolateDataSetAttributes = obj.NewInstance ()`
- `vtkInterpolateDataSetAttributes = obj.SafeDownCast (vtkObject o)`
- `vtkDataSetCollection = obj.GetInputList ()` - Return the list of inputs to this filter.
- `obj.SetT (double )` - Specify interpolation parameter t.
- `double = obj.GetTMinValue ()` - Specify interpolation parameter t.
- `double = obj.GetTMaxValue ()` - Specify interpolation parameter t.
- `double = obj.GetT ()` - Specify interpolation parameter t.

33.128 vtkInterpolatingSubdivisionFilter

33.128.1 Usage

vtkInterpolatingSubdivisionFilter is an abstract class that defines the protocol for interpolating subdivision surface filters.

To create an instance of class vtkInterpolatingSubdivisionFilter, simply invoke its constructor as follows

```
obj = vtkInterpolatingSubdivisionFilter
```

33.128.2 Methods

The class vtkInterpolatingSubdivisionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInterpolatingSubdivisionFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInterpolatingSubdivisionFilter = obj.NewInstance ()`
- `vtkInterpolatingSubdivisionFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfSubdivisions (int )` - Set/get the number of subdivisions.
- `int = obj.GetNumberOfSubdivisions ()` - Set/get the number of subdivisions.

33.129 vtkKdTreeSelector

33.129.1 Usage

If SetKdTree is used, the filter ignores the input and selects based on that kd-tree. If SetKdTree is not used, the filter builds a kd-tree using the input point set and uses that tree for selection. The output is a vtkSelection containing the ids found in the kd-tree using the specified bounds.

To create an instance of class vtkKdTreeSelector, simply invoke its constructor as follows

```
obj = vtkKdTreeSelector
```

33.129.2 Methods

The class vtkKdTreeSelector has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkKdTreeSelector class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkKdTreeSelector = obj.NewInstance ()`
- `vtkKdTreeSelector = obj.SafeDownCast (vtkObject o)`
- `obj.SetKdTree (vtkKdTree tree)` - The kd-tree to use to find selected ids. The kd-tree must be initialized with the desired set of points. When this is set, the optional input is ignored.
- `vtkKdTree = obj.GetKdTree ()` - The kd-tree to use to find selected ids. The kd-tree must be initialized with the desired set of points. When this is set, the optional input is ignored.
- `obj.SetSelectionBounds (double , double , double , double , double , double )` - The bounds of the form (xmin,xmax,ymin,ymax,zmin,zmax). To perform a search in 2D, use the bounds (xmin,xmax,ymin,ymax,VTI
- `obj.SetSelectionBounds (double a[6])` - The bounds of the form (xmin,xmax,ymin,ymax,zmin,zmax). To perform a search in 2D, use the bounds (xmin,xmax,ymin,ymax,VTK_DOUBLE_MIN,VTK_DOUBLE_MAX).
- `double = obj. GetSelectionBounds ()` - The bounds of the form (xmin,xmax,ymin,ymax,zmin,zmax). To perform a search in 2D, use the bounds (xmin,xmax,ymin,ymax,VTK_DOUBLE_MIN,VTK_DOUBLE_MAX).
- `obj.SetSelectionFieldName (string )` - The field name to use when generating the selection. If set, creates a VALUES selection. If not set (or is set to NULL), creates a INDICES selection. By default this is not set.
- `string = obj.GetSelectionFieldName ()` - The field name to use when generating the selection. If set, creates a VALUES selection. If not set (or is set to NULL), creates a INDICES selection. By default this is not set.
- `obj.SetSelectionAttribute (int )` - The field attribute to use when generating the selection. If set, creates a PEDIGREEIDS or GLOBALIDS selection. If not set (or is set to -1), creates a INDICES selection. By default this is not set. NOTE: This should be set a constant in vtkDataSetAttributes, not vtkSelection.
- `int = obj.GetSelectionAttribute ()` - The field attribute to use when generating the selection. If set, creates a PEDIGREEIDS or GLOBALIDS selection. If not set (or is set to -1), creates a INDICES selection. By default this is not set. NOTE: This should be set a constant in vtkDataSetAttributes, not vtkSelection.
- `obj.SetSingleSelection (bool )` - Whether to only allow up to one value in the result. The item selected is closest to the center of the bounds, if there are any points within the selection threshold. Default is off.
- `bool = obj.GetSingleSelection ()` - Whether to only allow up to one value in the result. The item selected is closest to the center of the bounds, if there are any points within the selection threshold. Default is off.
- `obj.SingleSelectionOn ()` - Whether to only allow up to one value in the result. The item selected is closest to the center of the bounds, if there are any points within the selection threshold. Default is off.

- `obj.SingleSelectionOff ()` - Whether to only allow up to one value in the result. The item selected is closest to the center of the bounds, if there are any points within the selection threshold. Default is off.
- `obj.SetSingleSelectionThreshold (double )` - The threshold for the single selection. A single point is added to the selection if it is within this threshold from the bounds center. Default is 1.
- `double = obj.GetSingleSelectionThreshold ()` - The threshold for the single selection. A single point is added to the selection if it is within this threshold from the bounds center. Default is 1.
- `long = obj.GetMTime ()`

33.130 vtkLevelIdScalars

33.130.1 Usage

vtkLevelIdScalars is a filter that generates scalars using the level number for each level. Note that all datasets within a level get the same scalar. The new scalars array is named ĻevelIdScalars.

To create an instance of class vtkLevelIdScalars, simply invoke its constructor as follows

```
obj = vtkLevelIdScalars
```

33.130.2 Methods

The class vtkLevelIdScalars has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLevelIdScalars class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLevelIdScalars = obj.NewInstance ()`
- `vtkLevelIdScalars = obj.SafeDownCast (vtkObject o)`

33.131 vtkLinearExtrusionFilter

33.131.1 Usage

vtkLinearExtrusionFilter is a modeling filter. It takes polygonal data as input and generates polygonal data on output. The input dataset is swept according to some extrusion function and creates new polygonal primitives. These primitives form a "skirt" or swept surface. For example, sweeping a line results in a quadrilateral, and sweeping a triangle creates a "wedge".

There are a number of control parameters for this filter. You can control whether the sweep of a 2D object (i.e., polygon or triangle strip) is capped with the generating geometry via the "Capping" ivar. Also, you can extrude in the direction of a user specified vector, towards a point, or in the direction of vertex normals (normals must be provided - use vtkPolyDataNormals if necessary). The amount of extrusion is controlled by the "ScaleFactor" instance variable.

The skirt is generated by locating certain topological features. Free edges (edges of polygons or triangle strips only used by one polygon or triangle strips) generate surfaces. This is true also of lines or polylines. Vertices generate lines.

This filter can be used to create 3D fonts, 3D irregular bar charts, or to model 2 1/2D objects like punched plates. It also can be used to create solid objects from 2D polygonal meshes.

To create an instance of class vtkLinearExtrusionFilter, simply invoke its constructor as follows

```
obj = vtkLinearExtrusionFilter
```

33.131.2 Methods

The class vtkLinearExtrusionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLinearExtrusionFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLinearExtrusionFilter = obj.NewInstance ()`
- `vtkLinearExtrusionFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetExtrusionType (int )` - Set/Get the type of extrusion.
- `int = obj.GetExtrusionTypeMinValue ()` - Set/Get the type of extrusion.
- `int = obj.GetExtrusionTypeMaxValue ()` - Set/Get the type of extrusion.
- `int = obj.GetExtrusionType ()` - Set/Get the type of extrusion.
- `obj.SetExtrusionTypeToVectorExtrusion ()` - Set/Get the type of extrusion.
- `obj.SetExtrusionTypeToNormalExtrusion ()` - Set/Get the type of extrusion.
- `obj.SetExtrusionTypeToPointExtrusion ()` - Set/Get the type of extrusion.
- `obj.SetCapping (int )` - Turn on/off the capping of the skirt.
- `int = obj.GetCapping ()` - Turn on/off the capping of the skirt.
- `obj.CappingOn ()` - Turn on/off the capping of the skirt.
- `obj.CappingOff ()` - Turn on/off the capping of the skirt.
- `obj.SetScaleFactor (double )` - Set/Get extrusion scale factor,
- `double = obj.GetScaleFactor ()` - Set/Get extrusion scale factor,
- `obj.SetVector (double , double , double )` - Set/Get extrusion vector. Only needs to be set if VectorExtrusion is turned on.
- `obj.SetVector (double a[3])` - Set/Get extrusion vector. Only needs to be set if VectorExtrusion is turned on.
- `double = obj. GetVector ()` - Set/Get extrusion vector. Only needs to be set if VectorExtrusion is turned on.
- `obj.SetExtrusionPoint (double , double , double )` - Set/Get extrusion point. Only needs to be set if PointExtrusion is turned on. This is the point towards which extrusion occurs.
- `obj.SetExtrusionPoint (double a[3])` - Set/Get extrusion point. Only needs to be set if PointExtrusion is turned on. This is the point towards which extrusion occurs.
- `double = obj. GetExtrusionPoint ()` - Set/Get extrusion point. Only needs to be set if PointExtrusion is turned on. This is the point towards which extrusion occurs.

33.132 vtkLinearSubdivisionFilter

33.132.1 Usage

vtkLinearSubdivisionFilter is a filter that generates output by subdividing its input polydata. Each subdivision iteration create 4 new triangles for each triangle in the polydata.

To create an instance of class vtkLinearSubdivisionFilter, simply invoke its constructor as follows

```
obj = vtkLinearSubdivisionFilter
```

33.132.2 Methods

The class vtkLinearSubdivisionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLinearSubdivisionFilter class.

- `string = obj.GetClassName ()` - Construct object with NumberOfSubdivisions set to 1.
- `int = obj.IsA (string name)` - Construct object with NumberOfSubdivisions set to 1.
- `vtkLinearSubdivisionFilter = obj.NewInstance ()` - Construct object with NumberOfSubdivisions set to 1.
- `vtkLinearSubdivisionFilter = obj.SafeDownCast (vtkObject o)` - Construct object with NumberOfSubdivisions set to 1.

33.133 vtkLineSource

33.133.1 Usage

vtkLineSource is a source object that creates a polyline defined by two endpoints. The number of segments composing the polyline is controlled by setting the object resolution.

To create an instance of class vtkLineSource, simply invoke its constructor as follows

```
obj = vtkLineSource
```

33.133.2 Methods

The class vtkLineSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLineSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLineSource = obj.NewInstance ()`
- `vtkLineSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetPoint1 (double , double , double )` - Set position of first end point.
- `obj.SetPoint1 (double  a[3])` - Set position of first end point.
- `double = obj. GetPoint1 ()` - Set position of first end point.
- `obj.SetPoint2 (double , double , double )` - Set position of other end point.

- `obj.SetPoint2 (double  a[3])` - Set position of other end point.
- `double = obj. GetPoint2 ()` - Set position of other end point.
- `obj.SetResolution (int )` - Divide line into resolution number of pieces.
- `int = obj.GetResolutionMinValue ()` - Divide line into resolution number of pieces.
- `int = obj.GetResolutionMaxValue ()` - Divide line into resolution number of pieces.
- `int = obj.GetResolution ()` - Divide line into resolution number of pieces.

33.134 vtkLinkEdgels

33.134.1 Usage

vtkLinkEdgels links edgels into digital curves which are then stored as polylines. The algorithm works one pixel at a time only looking at its immediate neighbors. There is a GradientThreshold that can be set that eliminates any pixels with a smaller gradient value. This can be used as the lower threshold of a two value edgel thresholding.

For the remaining edgels, links are first tried for the four connected neighbors. A successful neighbor will satisfy three tests. First both edgels must be above the gradient threshold. Second, the difference between the orientation between the two edgels (Alpha) and each edgels orientation (Phi) must be less than LinkThreshold. Third, the difference between the two edgels Phi values must be less than PhiThreshold. The most successful link is selected. The measure is simply the sum of the three angle differences (actually stored as the sum of the cosines). If none of the four connect neighbors succeeds, then the eight connect neighbors are examined using the same method.

This filter requires gradient information so you will need to use a vtkImageGradient at some point prior to this filter. Typically a vtkNonMaximumSuppression filter is also used. vtkThresholdEdgels can be used to complete the two value edgel thresholding as used in a Canny edge detector. The vtkSubpixelPositionEdgels filter can also be used after this filter to adjust the edgel locations.

To create an instance of class vtkLinkEdgels, simply invoke its constructor as follows

```
obj = vtkLinkEdgels
```

33.134.2 Methods

The class vtkLinkEdgels has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLinkEdgels class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLinkEdgels = obj.NewInstance ()`
- `vtkLinkEdgels = obj.SafeDownCast (vtkObject o)`
- `obj.SetLinkThreshold (double )` - Set/Get the threshold for Phi vs. Alpha link thresholding.
- `double = obj.GetLinkThreshold ()` - Set/Get the threshold for Phi vs. Alpha link thresholding.
- `obj.SetPhiThreshold (double )` - Set/get the threshold for Phi vs. Phi link thresholding.
- `double = obj.GetPhiThreshold ()` - Set/get the threshold for Phi vs. Phi link thresholding.
- `obj.SetGradientThreshold (double )` - Set/Get the threshold for image gradient thresholding.
- `double = obj.GetGradientThreshold ()` - Set/Get the threshold for image gradient thresholding.

33.135 vtkLoopSubdivisionFilter

33.135.1 Usage

vtkLoopSubdivisionFilter is an approximating subdivision scheme that creates four new triangles for each triangle in the mesh. The user can specify the NumberOfSubdivisions. Loop's subdivision scheme is described in: Loop, C., "Smooth Subdivision surfaces based on triangles,", Masters Thesis, University of Utah, August 1987. For a nice summary of the technique see, Hoppe, H., et. al, "Piecewise Smooth Surface Reconstruction,:, Proceedings of Siggraph 94 (Orlando, Florida, July 24-29, 1994). In COmputer Graphics Proceedings, Annual COnference Series, 1994, ACM SIGGRAPH, pp. 295-302. ¡P¿ The filter only operates on triangles. Users should use the vtkTriangleFilter to triangulate meshes that contain polygons or triangle strips. ¡P¿ The filter approximates point data using the same scheme. New triangles create at a subdivision step will have the cell data of their parent cell.

To create an instance of class vtkLoopSubdivisionFilter, simply invoke its constructor as follows

```
obj = vtkLoopSubdivisionFilter
```

33.135.2 Methods

The class vtkLoopSubdivisionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLoopSubdivisionFilter class.

- `string = obj.GetClassName ()` - Construct object with NumberOfSubdivisions set to 1.
- `int = obj.IsA (string name)` - Construct object with NumberOfSubdivisions set to 1.
- `vtkLoopSubdivisionFilter = obj.NewInstance ()` - Construct object with NumberOfSubdivisions set to 1.
- `vtkLoopSubdivisionFilter = obj.SafeDownCast (vtkObject o)` - Construct object with NumberOfSubdivisions set to 1.

33.136 vtkMarchingContourFilter

33.136.1 Usage

vtkMarchingContourFilter is a filter that takes as input any dataset and generates on output isosurfaces and/or isolines. The exact form of the output depends upon the dimensionality of the input data. Data consisting of 3D cells will generate isosurfaces, data consisting of 2D cells will generate isolines, and data with 1D or 0D cells will generate isopoints. Combinations of output type are possible if the input dimension is mixed.

This filter will identify special dataset types (e.g., structured points) and use the appropriate specialized filter to process the data. For examples, if the input dataset type is a volume, this filter will create an internal vtkMarchingCubes instance and use it. This gives much better performance.

To use this filter you must specify one or more contour values. You can either use the method SetValue() to specify each contour value, or use GenerateValues() to generate a series of evenly spaced contours. It is also possible to accelerate the operation of this filter (at the cost of extra memory) by using a vtkScalarTree. A scalar tree is used to quickly locate cells that contain a contour surface. This is especially effective if multiple contours are being extracted. If you want to use a scalar tree, invoke the method UseScalarTreeOn().

To create an instance of class vtkMarchingContourFilter, simply invoke its constructor as follows

```
obj = vtkMarchingContourFilter
```

33.136.2 Methods

The class vtkMarchingContourFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMarchingContourFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMarchingContourFilter = obj.NewInstance ()`
- `vtkMarchingContourFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)` - Methods to set / get contour values.
- `double = obj.GetValue (int i)` - Methods to set / get contour values.
- `obj.GetValues (double contourValues)` - Methods to set / get contour values.
- `obj.SetNumberOfContours (int number)` - Methods to set / get contour values.
- `int = obj.GetNumberOfContours ()` - Methods to set / get contour values.
- `obj.GenerateValues (int numContours, double range[2])` - Methods to set / get contour values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Methods to set / get contour values.
- `long = obj.GetMTime ()` - Modified GetMTime Because we delegate to vtkContourValues
- `obj.SetComputeNormals (int )` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeNormals ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOn ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOff ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeGradients (int )` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeGradients ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOn ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.

- `obj.ComputeGradientsOff ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeScalars (int )` - Set/Get the computation of scalars.
- `int = obj.GetComputeScalars ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOn ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOff ()` - Set/Get the computation of scalars.
- `obj.SetUseScalarTree (int )` - Enable the use of a scalar tree to accelerate contour extraction.
- `int = obj.GetUseScalarTree ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.UseScalarTreeOn ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.UseScalarTreeOff ()` - Enable the use of a scalar tree to accelerate contour extraction.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.

33.137 vtkMarchingCubes

33.137.1 Usage

vtkMarchingCubes is a filter that takes as input a volume (e.g., 3D structured point set) and generates on output one or more isosurfaces. One or more contour values must be specified to generate the isosurfaces. Alternatively, you can specify a min/max scalar range and the number of contours to generate a series of evenly spaced contour values.

To create an instance of class vtkMarchingCubes, simply invoke its constructor as follows

```
obj = vtkMarchingCubes
```

33.137.2 Methods

The class vtkMarchingCubes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMarchingCubes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMarchingCubes = obj.NewInstance ()`
- `vtkMarchingCubes = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (int i, double value)`
- `double = obj.GetValue (int i)`

- `obj.GetValues (double contourValues)`
- `obj.SetNumberOfContours (int number)`
- `int = obj.GetNumberOfContours ()`
- `obj.GenerateValues (int numContours, double range[2])`
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)`
- `long = obj.GetMTime ()`
- `obj.SetComputeNormals (int )` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeNormals ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOn ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOff ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeGradients (int )` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeGradients ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOn ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOff ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeScalars (int )` - Set/Get the computation of scalars.
- `int = obj.GetComputeScalars ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOn ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOff ()` - Set/Get the computation of scalars.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Overide the default locator. Useful for changing the number of bins for performance or specifying a more aggressive locator.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Overide the default locator. Useful for changing the number of bins for performance or specifying a more aggressive locator.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.

33.138 vtkMarchingSquares

33.138.1 Usage

vtkMarchingSquares is a filter that takes as input a structured points set and generates on output one or more isolines. One or more contour values must be specified to generate the isolines. Alternatively, you can specify a min/max scalar range and the number of contours to generate a series of evenly spaced contour values.

To generate contour lines the input data must be of topological dimension 2 (i.e., an image). If not, you can use the ImageRange ivar to select an image plane from an input volume. This avoids having to extract a plane first (using vtkExtractSubVolume). The filter deals with this by first trying to use the input data directly, and if not a 2D image, then uses the ImageRange ivar to reduce it to an image.

To create an instance of class vtkMarchingSquares, simply invoke its constructor as follows

```
obj = vtkMarchingSquares
```

33.138.2 Methods

The class vtkMarchingSquares has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMarchingSquares class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMarchingSquares = obj.NewInstance ()`
- `vtkMarchingSquares = obj.SafeDownCast (vtkObject o)`
- `obj.SetImageRange (int [6])` - Set/Get the i-j-k index range which define a plane on which to generate contour lines. Using this ivar it is possible to input a 3D volume directly and then generate contour lines on one of the i-j-k planes, or a portion of a plane.
- `int = obj. GetImageRange ()` - Set/Get the i-j-k index range which define a plane on which to generate contour lines. Using this ivar it is possible to input a 3D volume directly and then generate contour lines on one of the i-j-k planes, or a portion of a plane.
- `obj.SetImageRange (int imin, int imax, int jmin, int jmax, int kmin, int kmax)` - Set/Get the i-j-k index range which define a plane on which to generate contour lines. Using this ivar it is possible to input a 3D volume directly and then generate contour lines on one of the i-j-k planes, or a portion of a plane.
- `obj.SetValue (int i, double value)` - Methods to set contour values
- `double = obj.GetValue (int i)` - Methods to set contour values
- `obj.GetValues (double contourValues)` - Methods to set contour values
- `obj.SetNumberOfContours (int number)` - Methods to set contour values
- `int = obj.GetNumberOfContours ()` - Methods to set contour values
- `obj.GenerateValues (int numContours, double range[2])` - Methods to set contour values
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Methods to set contour values
- `long = obj.GetMTime ()` - Because we delegate to vtkContourValues

- `obj.SetLocator (vtkIncrementalPointLocator locator)`
- `vtkIncrementalPointLocator = obj.GetLocator ()`
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified. The locator is used to merge coincident points.

33.139 vtkMaskFields

33.139.1 Usage

vtkMaskFields is used to mark which fields in the input dataset get copied to the output. The output will contain only those fields marked as on by the filter.

To create an instance of class vtkMaskFields, simply invoke its constructor as follows

```
obj = vtkMaskFields
```

33.139.2 Methods

The class vtkMaskFields has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMaskFields class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMaskFields = obj.NewInstance ()`
- `vtkMaskFields = obj.SafeDownCast (vtkObject o)`
- `obj.CopyFieldOn (int fieldLocation, string name)` - Turn on/off the copying of the field or specified by name. During the copying/passing, the following rules are followed for each array: 1. If the copy flag for an array is set (on or off), it is applied This overrides rule 2. 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array A field name and a location must be specified. For example: @verbatim maskFields-¿CopyFieldOff(vtkMaskFields::CELL_DATA, "foo"); @endverbatim causes the field "foo" on the input cell data to not get copied to the output.
- `obj.CopyFieldOff (int fieldLocation, string name)` - Turn on/off the copying of the attribute or specified by vtkDataSetAttributes:AttributeTypes. During the copying/passing, the following rules are followed for each array: 1. If the copy flag for an array is set (on or off), it is applied This overrides rule 2. 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array An attribute type and a location must be specified. For example: @verbatim maskFields-¿CopyAttributeOff(vtkMaskFields::POINT_DATA, vtkDataSetAttributes::SCALARS); @endverbatim causes the scalars on the input point data to not get copied to the output.
- `obj.CopyAttributeOn (int attributeLocation, int attributeType)` - Turn on/off the copying of the attribute or specified by vtkDataSetAttributes:AttributeTypes. During the copying/passing, the following rules are followed for each array: 1. If the copy flag for an array is set (on or off), it is applied This overrides rule 2. 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array An attribute type and a location must be specified. For example: @verbatim maskFields-¿CopyAttributeOff(vtkMaskFields::POINT_DATA, vtkDataSetAttributes::SCALARS); @endverbatim causes the scalars on the input point data to not get copied to the output.
- `obj.CopyAttributeOff (int attributeLocation, int attributeType)` - Convenience methods which operate on all field data or attribute data. More specific than CopyAllOn or CopyAllOff

- `obj.CopyFieldsOff ()` - Convenience methods which operate on all field data or attribute data. More specific than CopyAllOn or CopyAllOff
- `obj.CopyAttributesOff ()`
- `obj.CopyFieldsOn ()`
- `obj.CopyAttributesOn ()` - Helper methods used by other language bindings. Allows the caller to specify arguments as strings instead of enums.
- `obj.CopyAttributeOn (string attributeLoc, string attributeType)` - Helper methods used by other language bindings. Allows the caller to specify arguments as strings instead of enums.
- `obj.CopyAttributeOff (string attributeLoc, string attributeType)` - Helper methods used by other language bindings. Allows the caller to specify arguments as strings instead of enums.
- `obj.CopyFieldOn (string fieldLoc, string name)` - Helper methods used by other language bindings. Allows the caller to specify arguments as strings instead of enums.
- `obj.CopyFieldOff (string fieldLoc, string name)` - Helper methods used by other language bindings. Allows the caller to specify arguments as strings instead of enums.
- `obj.CopyAllOn ()` - Turn on copying of all data. During the copying/passing, the following rules are followed for each array: 1. If the copy flag for an array is set (on or off), it is applied This overrides rule 2. 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array
- `obj.CopyAllOff ()` - Turn off copying of all data. During the copying/passing, the following rules are followed for each array: 1. If the copy flag for an array is set (on or off), it is applied This overrides rule 2. 2. If CopyAllOn is set, copy the array. If CopyAllOff is set, do not copy the array

33.140 vtkMaskPoints

33.140.1 Usage

vtkMaskPoints is a filter that passes through points and point attributes from input dataset. (Other geometry is not passed through.) It is possible to mask every nth point, and to specify an initial offset to begin masking from. A special random mode feature enables random selection of points. The filter can also generate vertices (topological primitives) as well as points. This is useful because vertices are rendered while points are not.

To create an instance of class vtkMaskPoints, simply invoke its constructor as follows

```
obj = vtkMaskPoints
```

33.140.2 Methods

The class vtkMaskPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMaskPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMaskPoints = obj.NewInstance ()`
- `vtkMaskPoints = obj.SafeDownCast (vtkObject o)`
- `obj.SetOnRatio (int )` - Turn on every nth point.
- `int = obj.GetOnRatioMinValue ()` - Turn on every nth point.

- `int = obj.GetOnRatioMaxValue ()` - Turn on every nth point.
- `int = obj.GetOnRatio ()` - Turn on every nth point.
- `obj.SetMaximumNumberOfPoints (vtkIdType )` - Limit the number of points that can be passed through.
- `vtkIdType = obj.GetMaximumNumberOfPointsMinValue ()` - Limit the number of points that can be passed through.
- `vtkIdType = obj.GetMaximumNumberOfPointsMaxValue ()` - Limit the number of points that can be passed through.
- `vtkIdType = obj.GetMaximumNumberOfPoints ()` - Limit the number of points that can be passed through.
- `obj.SetOffset (vtkIdType )` - Start with this point.
- `vtkIdType = obj.GetOffsetMinValue ()` - Start with this point.
- `vtkIdType = obj.GetOffsetMaxValue ()` - Start with this point.
- `vtkIdType = obj.GetOffset ()` - Start with this point.
- `obj.SetRandomMode (int )` - Special flag causes randomization of point selection. If this mode is on, statistically every nth point (i.e., OnRatio) will be displayed.
- `int = obj.GetRandomMode ()` - Special flag causes randomization of point selection. If this mode is on, statistically every nth point (i.e., OnRatio) will be displayed.
- `obj.RandomModeOn ()` - Special flag causes randomization of point selection. If this mode is on, statistically every nth point (i.e., OnRatio) will be displayed.
- `obj.RandomModeOff ()` - Special flag causes randomization of point selection. If this mode is on, statistically every nth point (i.e., OnRatio) will be displayed.
- `obj.SetGenerateVertices (int )` - Generate output polydata vertices as well as points. A useful convenience method because vertices are drawn (they are topology) while points are not (they are geometry). By default this method is off.
- `int = obj.GetGenerateVertices ()` - Generate output polydata vertices as well as points. A useful convenience method because vertices are drawn (they are topology) while points are not (they are geometry). By default this method is off.
- `obj.GenerateVerticesOn ()` - Generate output polydata vertices as well as points. A useful convenience method because vertices are drawn (they are topology) while points are not (they are geometry). By default this method is off.
- `obj.GenerateVerticesOff ()` - Generate output polydata vertices as well as points. A useful convenience method because vertices are drawn (they are topology) while points are not (they are geometry). By default this method is off.
- `obj.SetSingleVertexPerCell (int )` - When vertex generation is enabled, by default vertices are produced as multi-vertex cells (more than one per cell), if you wish to have a single vertex per cell, enable this flag.
- `int = obj.GetSingleVertexPerCell ()` - When vertex generation is enabled, by default vertices are produced as multi-vertex cells (more than one per cell), if you wish to have a single vertex per cell, enable this flag.

- `obj.SingleVertexPerCellOn ()` - When vertex generation is enabled, by default vertices are produced as multi-vertex cells (more than one per cell), if you wish to have a single vertex per cell, enable this flag.
- `obj.SingleVertexPerCellOff ()` - When vertex generation is enabled, by default vertices are produced as multi-vertex cells (more than one per cell), if you wish to have a single vertex per cell, enable this flag.

33.141 vtkMaskPolyData

33.141.1 Usage

vtkMaskPolyData is a filter that sub-samples the cells of input polygonal data. The user specifies every nth item, with an initial offset to begin sampling.

To create an instance of class vtkMaskPolyData, simply invoke its constructor as follows

```
obj = vtkMaskPolyData
```

33.141.2 Methods

The class vtkMaskPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMaskPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMaskPolyData = obj.NewInstance ()`
- `vtkMaskPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetOnRatio (int )` - Turn on every nth entity (cell).
- `int = obj.GetOnRatioMinValue ()` - Turn on every nth entity (cell).
- `int = obj.GetOnRatioMaxValue ()` - Turn on every nth entity (cell).
- `int = obj.GetOnRatio ()` - Turn on every nth entity (cell).
- `obj.SetOffset (vtkIdType )` - Start with this entity (cell).
- `vtkIdType = obj.GetOffsetMinValue ()` - Start with this entity (cell).
- `vtkIdType = obj.GetOffsetMaxValue ()` - Start with this entity (cell).
- `vtkIdType = obj.GetOffset ()` - Start with this entity (cell).

33.142 vtkMassProperties

33.142.1 Usage

vtkMassProperties estimates the volume, the surface area, and the normalized shape index of a triangle mesh. The algorithm implemented here is based on the discrete form of the divergence theorem. The general assumption here is that the model is of closed surface. For more details see the following reference (Alyassin A.M. et al, "Evaluation of new algorithms for the interactive measurement of surface area and volume", Med Phys 21(6) 1994.).

To create an instance of class vtkMassProperties, simply invoke its constructor as follows

```
obj = vtkMassProperties
```

33.142.2 Methods

The class vtkMassProperties has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMassProperties class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMassProperties = obj.NewInstance ()`
- `vtkMassProperties = obj.SafeDownCast (vtkObject o)`
- `double = obj.GetVolume ()` - Compute and return the projected volume. Typically you should compare this volume to the value returned by GetVolume if you get an error (GetVolume()-GetVolumeProjected())*10000 that is greater than GetVolume() this should identify a problem: * Either the polydata is not closed * Or the polydata contains triangle that are flipped
- `double = obj.GetVolumeProjected ()` - Compute and return the volume projected on to each axis aligned plane.
- `double = obj.GetVolumeX ()` - Compute and return the volume projected on to each axis aligned plane.
- `double = obj.GetVolumeY ()` - Compute and return the volume projected on to each axis aligned plane.
- `double = obj.GetVolumeZ ()` - Compute and return the weighting factors for the maximum unit normal component (MUNC).
- `double = obj.GetKx ()` - Compute and return the weighting factors for the maximum unit normal component (MUNC).
- `double = obj.GetKy ()` - Compute and return the weighting factors for the maximum unit normal component (MUNC).
- `double = obj.GetKz ()` - Compute and return the area.
- `double = obj.GetSurfaceArea ()` - Compute and return the min cell area.
- `double = obj.GetMinCellArea ()` - Compute and return the max cell area.
- `double = obj.GetMaxCellArea ()` - Compute and return the normalized shape index. This characterizes the deviation of the shape of an object from a sphere. A sphere's NSI is one. This number is always ¿= 1.0.
- `double = obj.GetNormalizedShapeIndex ()`

33.143 vtkMergeCells

33.143.1 Usage

Designed to work with distributed vtkDataSets, this class will take vtkDataSets and merge them back into a single vtkUnstructuredGrid.

The vtkPoints object of the unstructured grid will have data type VTK_FLOAT, regardless of the data type of the points of the input vtkDataSets. If this is a problem, someone must let me know.

It is assumed the different DataSets have the same field arrays. If the name of a global point ID array is provided, this class will refrain from including duplicate points in the merged Ugrid. This class differs

from vtkAppendFilter in these ways: (1) it uses less memory than that class (which uses memory equal to twice the size of the final Ugrid) but requires that you know the size of the final Ugrid in advance (2) this class assumes the individual DataSets have the same field arrays, while vtkAppendFilter intersects the field arrays (3) this class knows duplicate points may be appearing in the DataSets and can filter those out, (4) this class is not a filter.

To create an instance of class vtkMergeCells, simply invoke its constructor as follows

```
obj = vtkMergeCells
```

33.143.2 Methods

The class vtkMergeCells has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMergeCells class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMergeCells = obj.NewInstance ()`
- `vtkMergeCells = obj.SafeDownCast (vtkObject o)`
- `obj.SetUnstructuredGrid (vtkUnstructuredGrid )`
- `vtkUnstructuredGrid = obj.GetUnstructuredGrid ()`
- `obj.SetTotalNumberOfCells (vtkIdType )`
- `vtkIdType = obj.GetTotalNumberOfCells ()`
- `obj.SetTotalNumberOfPoints (vtkIdType )`
- `vtkIdType = obj.GetTotalNumberOfPoints ()`
- `obj.SetUseGlobalIds (int )`
- `int = obj.GetUseGlobalIds ()`
- `obj.SetPointMergeTolerance (float )`
- `float = obj.GetPointMergeToleranceMinValue ()`
- `float = obj.GetPointMergeToleranceMaxValue ()`
- `float = obj.GetPointMergeTolerance ()`
- `obj.SetUseGlobalCellIds (int )`
- `int = obj.GetUseGlobalCellIds ()`
- `obj.SetMergeDuplicatePoints (int )`
- `int = obj.GetMergeDuplicatePoints ()`
- `obj.MergeDuplicatePointsOn ()`
- `obj.MergeDuplicatePointsOff ()`
- `obj.SetTotalNumberOfDataSets (int )`
- `int = obj.GetTotalNumberOfDataSets ()`
- `int = obj.MergeDataSet (vtkDataSet set)`
- `obj.Finish ()`

33.144 vtkMergeDataObjectFilter

33.144.1 Usage

vtkMergeDataObjectFilter is a filter that merges the field from a vtkDataObject with a vtkDataSet. The resulting combined dataset can then be processed by other filters (e.g., vtkFieldDataToAttributeDataFilter) to create attribute data like scalars, vectors, etc.

The filter operates as follows. The field data from the vtkDataObject is merged with the input's vtkDataSet and then placed in the output. You can choose to place the field data into the cell data field, the point data field, or the datasets field (i.e., the one inherited from vtkDataSet's superclass vtkDataObject). All this data shuffling occurs via reference counting, therefore memory is not copied.

One of the uses of this filter is to allow you to read/generate the structure of a dataset independent of the attributes. So, for example, you could store the dataset geometry/topology in one file, and field data in another. Then use this filter in combination with vtkFieldDataToAttributeData to create a dataset ready for processing in the visualization pipeline.

To create an instance of class vtkMergeDataObjectFilter, simply invoke its constructor as follows

```
obj = vtkMergeDataObjectFilter
```

33.144.2 Methods

The class vtkMergeDataObjectFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMergeDataObjectFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMergeDataObjectFilter = obj.NewInstance ()`
- `vtkMergeDataObjectFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetDataObject (vtkDataObject object)` - Specify the data object to merge with the input dataset.
- `vtkDataObject = obj.GetDataObject ()` - Specify the data object to merge with the input dataset.
- `obj.SetOutputField (int )` - Specify where to place the field data during the merge process. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.
- `int = obj.GetOutputField ()` - Specify where to place the field data during the merge process. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.
- `obj.SetOutputFieldToDataObjectField ()` - Specify where to place the field data during the merge process. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.
- `obj.SetOutputFieldToPointDataField ()` - Specify where to place the field data during the merge process. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.
- `obj.SetOutputFieldToCellDataField ()` - Specify where to place the field data during the merge process. There are three choices: the field data associated with the vtkDataObject superclass; the point field attribute data; and the cell field attribute data.

33.145 vtkMergeFields

33.145.1 Usage

vtkMergeFields is used to merge mutliple field into one. The new field is put in the same field data as the original field. For example @verbatim mf-¿SetOutputField("foo", vtkMergeFields::POINT_DATA); mf-¿SetNumberOfComponents(2); mf-¿Merge(0, "array1", 1); mf-¿Merge(1, "array2", 0); @endverbatim will tell vtkMergeFields to use the 2nd component of array1 and the 1st component of array2 to create a 2 component field called foo. The same can be done using Tcl: @verbatim mf SetOutputField foo POINT_DATA mf Merge 0 array1 1 mf Merge 1 array2 0

Field locations: DATA_OBJECT, POINT_DATA, CELL_DATA @endverbatim

To create an instance of class vtkMergeFields, simply invoke its constructor as follows

```
obj = vtkMergeFields
```

33.145.2 Methods

The class vtkMergeFields has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMergeFields class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMergeFields = obj.NewInstance ()`
- `vtkMergeFields = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutputField (string name, int fieldLoc)` - The output field will have the given name and it will be in fieldLoc (the input fields also have to be in fieldLoc).
- `obj.SetOutputField (string name, string fieldLoc)` - Helper method used by the other language bindings. Allows the caller to specify arguments as strings instead of enums.Returns an operation id which can later be used to remove the operation.
- `obj.Merge (int component, string arrayName, int sourceComp)` - Add a component (arrayName,sourceComp) to the output field.
- `obj.SetNumberOfComponents (int )` - Set the number of the components in the output field. This has to be set before execution. Default value is 0.
- `int = obj.GetNumberOfComponents ()` - Set the number of the components in the output field. This has to be set before execution. Default value is 0.

33.146 vtkMergeFilter

33.146.1 Usage

vtkMergeFilter is a filter that extracts separate components of data from different datasets and merges them into a single dataset. The output from this filter is of the same type as the input (i.e., vtkDataSet.) It treats both cell and point data set attributes.

To create an instance of class vtkMergeFilter, simply invoke its constructor as follows

```
obj = vtkMergeFilter
```

33.146.2 Methods

The class vtkMergeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMergeFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMergeFilter = obj.NewInstance ()`
- `vtkMergeFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetGeometry (vtkDataSet input)` - Specify object from which to extract geometry information. Old style. Use SetGeometryConnection() instead.
- `vtkDataSet = obj.GetGeometry ()` - Specify object from which to extract geometry information. Old style. Use SetGeometryConnection() instead.
- `obj.SetGeometryConnection (vtkAlgorithmOutput algOutput)` - Specify object from which to extract scalar information. Old style. Use SetScalarsConnection() instead.
- `obj.SetScalars (vtkDataSet )` - Specify object from which to extract scalar information. Old style. Use SetScalarsConnection() instead.
- `vtkDataSet = obj.GetScalars ()` - Specify object from which to extract scalar information. Old style. Use SetScalarsConnection() instead.
- `obj.SetScalarsConnection (vtkAlgorithmOutput algOutput)` - Set / get the object from which to extract vector information. Old style. Use SetVectorsConnection() instead.
- `obj.SetVectors (vtkDataSet )` - Set / get the object from which to extract vector information. Old style. Use SetVectorsConnection() instead.
- `vtkDataSet = obj.GetVectors ()` - Set / get the object from which to extract vector information. Old style. Use SetVectorsConnection() instead.
- `obj.SetVectorsConnection (vtkAlgorithmOutput algOutput)` - Set / get the object from which to extract normal information. Old style. Use SetNormalsConnection() instead.
- `obj.SetNormals (vtkDataSet )` - Set / get the object from which to extract normal information. Old style. Use SetNormalsConnection() instead.
- `vtkDataSet = obj.GetNormals ()` - Set / get the object from which to extract normal information. Old style. Use SetNormalsConnection() instead.
- `obj.SetNormalsConnection (vtkAlgorithmOutput algOutput)` - Set / get the object from which to extract texture coordinates information. Old style. Use SetTCoordsConnection() instead.
- `obj.SetTCoords (vtkDataSet )` - Set / get the object from which to extract texture coordinates information. Old style. Use SetTCoordsConnection() instead.
- `vtkDataSet = obj.GetTCoords ()` - Set / get the object from which to extract texture coordinates information. Old style. Use SetTCoordsConnection() instead.
- `obj.SetTCoordsConnection (vtkAlgorithmOutput algOutput)` - Set / get the object from which to extract tensor data. Old style. Use SetTensorsConnection() instead.
- `obj.SetTensors (vtkDataSet )` - Set / get the object from which to extract tensor data. Old style. Use SetTensorsConnection() instead.

- `vtkDataSet = obj.GetTensors ()` - Set / get the object from which to extract tensor data. Old style. Use SetTensorsConnection() instead.
- `obj.SetTensorsConnection (vtkAlgorithmOutput algOutput)` - Set the object from which to extract a field and the name of the field. Note that this does not create pipeline connectivity.
- `obj.AddField (string name, vtkDataSet input)` - Set the object from which to extract a field and the name of the field. Note that this does not create pipeline connectivity.

33.147 vtkMeshQuality

33.147.1 Usage

vtkMeshQuality computes one or more functions of (geometric) quality for each 2-D and 3-D cell (triangle, quadrilateral, tetrahedron, or hexahedron) of a mesh. These functions of quality are then averaged over the entire mesh. The minimum, average, maximum, and unbiased variance of quality for each type of cell is stored in the output mesh's FieldData. The FieldData arrays are named "Mesh Triangle Quality," "Mesh Quadrilateral Quality," "Mesh Tetrahedron Quality," and "Mesh Hexahedron Quality." Each array has a single tuple with 5 components. The first 4 components are the quality statistics mentioned above; the final value is the number of cells of the given type. This final component makes aggregation of statistics for distributed mesh data possible.

By default, the per-cell quality is added to the mesh's cell data, in an array named "Quality." Cell types not supported by this filter will have an entry of 0. Use SaveCellQualityOff() to store only the final statistics.

This version of the filter written by Philippe Pebay and David Thompson overtakes an older version written by Leila Baghdadi, Hanif Ladak, and David Steinman at the Imaging Research Labs, Robarts Research Institute. That version only supported tetrahedral radius ratio. See the CompatibilityModeOn() member for information on how to make this filter behave like the previous implementation. For more information on the triangle quality functions of this class, cf. Pebay & Baker 2003, Analysis of triangle quality measures, Math Comp 72:244. For more information on the quadrangle quality functions of this class, cf. Pebay 2004, Planar Quadrangle Quality Measures, Eng Comp 20:2.

To create an instance of class vtkMeshQuality, simply invoke its constructor as follows

```
obj = vtkMeshQuality
```

33.147.2 Methods

The class vtkMeshQuality has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMeshQuality class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMeshQuality = obj.NewInstance ()`
- `vtkMeshQuality = obj.SafeDownCast (vtkObject o)`
- `obj.SetSaveCellQuality (int )` - This variable controls whether or not cell quality is stored as cell data in the resulting mesh or discarded (leaving only the aggregate quality average of the entire mesh, recorded in the FieldData).
- `int = obj.GetSaveCellQuality ()` - This variable controls whether or not cell quality is stored as cell data in the resulting mesh or discarded (leaving only the aggregate quality average of the entire mesh, recorded in the FieldData).

- `obj.SaveCellQualityOn ()` - This variable controls whether or not cell quality is stored as cell data in the resulting mesh or discarded (leaving only the aggregate quality average of the entire mesh, recorded in the FieldData).

- `obj.SaveCellQualityOff ()` - This variable controls whether or not cell quality is stored as cell data in the resulting mesh or discarded (leaving only the aggregate quality average of the entire mesh, recorded in the FieldData).

- `obj.SetTriangleQualityMeasure (int )` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `int = obj.GetTriangleQualityMeasure ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTriangleQualityMeasureToArea ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTriangleQualityMeasureToEdgeRatio ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTriangleQualityMeasureToAspectRatio ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RAT VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTriangleQualityMeasureToRadiusRatio ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RAT VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTriangleQualityMeasureToAspectFrobenius ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_C VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTriangleQualityMeasureToMinAngle ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.
- `obj.SetTriangleQualityMeasureToMaxAngle ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.
- `obj.SetTriangleQualityMeasureToCondition ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.
- `obj.SetTriangleQualityMeasureToScaledJacobian ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_C VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.
- `obj.SetTriangleQualityMeasureToRelativeSizeSquared ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_C VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.
- `obj.SetTriangleQualityMeasureToShape ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.
- `obj.SetTriangleQualityMeasureToShapeAndSize ()` - Set/Get the particular estimator used to function the quality of triangles. The default is VTK_QUALITY_RADIUS_RATIO and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, and VTK_QUALITY_EDGE_RAT VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_CONDITION, VTK_QUALITY_SC VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.
- `obj.SetTriangleQualityMeasureToDistortion ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasure (int )` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `int = obj.GetQuadQualityMeasure ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToEdgeRatio ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToAspectRatio ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToRadiusRatio ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO

VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToMedAspectFrobenius ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RA VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToMaxAspectFrobenius ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RA VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToMaxEdgeRatios ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToSkew ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToTaper ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToWarpage ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToArea ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToStretch ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToMinAngle ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO

VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToMaxAngle ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToOddy ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToCondition ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToJacobian ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToScaledJacobian ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToShear ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToShape ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToRelativeSizeSquared ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RA VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToShapeAndSize ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO

VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToShearAndSize ()` - Set/Get the particular estimator used to measure the quality of quadrilaterals. The default is VTK_QUALITY_EDGE_RATIO and valid values also include VTK_QUALITY_RADIUS_RATIO, VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_MAX_EDGE_RATIO VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_WARPAGE, VTK_QUALITY_AREA, VTK_QUALITY_STRETCH, VTK_QUALITY_MIN_ANGLE, VTK_QUALITY_MAX_ANGLE, VTK_QUALITY_ODD VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

  Scope: Except for VTK_QUALITY_EDGE_RATIO, these estimators are intended for planar quadrilaterals only; use at your own risk if you really want to assess non-planar quadrilateral quality with those.

- `obj.SetQuadQualityMeasureToDistortion ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasure (int )` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `int = obj.GetTetQualityMeasure ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToEdgeRatio ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToAspectRatio ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS,

VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToRadiusRatio ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToAspectFrobenius ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBE VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToMinAngle ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToCollapseRatio ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENI VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToAspectBeta ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToAspectGamma ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToVolume ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA,

VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J
VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED,
VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToCondition ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToJacobian ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToScaledJacobian ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENI VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToShape ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENIUS, VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToRelativeSizeSquared ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FR VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToShapeAndSize ()` - Set/Get the particular estimator used to measure the quality of tetrahedra. The default is VTK_QUALITY_RADIUS_RATIO (identical to Verdict's aspect ratio beta) and valid values also include VTK_QUALITY_ASPECT_RATIO, VTK_QUALITY_ASPECT_FROBENI VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_COLLAPSE_RATIO, VTK_QUALITY_ASPECT_BETA, VTK_QUALITY_ASPECT_GAMMA, VTK_QUALITY_VOLUME, VTK_QUALITY_CONDITION, VTK_QUALITY_J VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetTetQualityMeasureToDistortion ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE_RATIO, VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUM VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY

VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasure (int )` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE_RAT VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.
- `int = obj.GetHexQualityMeasure ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToEdgeRatio ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToMedAspectFrobenius ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE_RATIO, VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUM VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToMaxAspectFrobenius ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE_RATIO, VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUM VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToMaxEdgeRatios ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE_RATIO, VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUM VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToSkew ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToTaper ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDG VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToVolume ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDG VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToStretch ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToDiagonal ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToDimension ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.
- `obj.SetHexQualityMeasureToOddy ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE

VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToCondition ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToJacobian ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToScaledJacobian ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE_RATIO, VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUM VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToShear ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDG VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToShape ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDG VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUME, VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY, VTK_QUALITY_CONDITIO VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SH VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToRelativeSizeSquared ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE_RATIO, VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUM VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY

VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToShapeAndSize ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE_RATIO, VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUM VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToShearAndSize ()` - Set/Get the particular estimator used to measure the quality of hexahedra. The default is VTK_QUALITY_MAX_ASPECT_FROBENIUS and valid values also include VTK_QUALITY_EDGE_RATIO, VTK_QUALITY_MAX_ASPECT_FROBENIUS, VTK_QUALITY_MAX_EDGE_RATIO, VTK_QUALITY_SKEW, VTK_QUALITY_TAPER, VTK_QUALITY_VOLUM VTK_QUALITY_STRETCH, VTK_QUALITY_DIAGONAL, VTK_QUALITY_DIMENSION, VTK_QUALITY_ODDY VTK_QUALITY_CONDITION, VTK_QUALITY_JACOBIAN, VTK_QUALITY_SCALED_JACOBIAN, VTK_QUALITY_SHEAR, VTK_QUALITY_SHAPE, VTK_QUALITY_RELATIVE_SIZE_SQUARED, VTK_QUALITY_SHAPE_AND_SIZE, VTK_QUALITY_SHEAR_AND_SIZE, and VTK_QUALITY_DISTORTION.

- `obj.SetHexQualityMeasureToDistortion ()` - This is a static function used to calculate the area of a triangle. It assumes that you pass the correct type of cell – no type checking is performed because this method is called from the inner loop of the Execute() member function.

- `obj.SetRatio (int r)` - These methods are deprecated. Use Get/SetSaveCellQuality() instead.

  Formerly, SetRatio could be used to disable computation of the tetrahedral radius ratio so that volume alone could be computed. Now, cell quality is always computed, but you may decide not to store the result for each cell. This allows average cell quality of a mesh to be calculated without requiring per-cell storage.

- `int = obj.GetRatio ()` - These methods are deprecated. Use Get/SetSaveCellQuality() instead.

  Formerly, SetRatio could be used to disable computation of the tetrahedral radius ratio so that volume alone could be computed. Now, cell quality is always computed, but you may decide not to store the result for each cell. This allows average cell quality of a mesh to be calculated without requiring per-cell storage.

- `obj.RatioOn ()` - These methods are deprecated. Use Get/SetSaveCellQuality() instead.

  Formerly, SetRatio could be used to disable computation of the tetrahedral radius ratio so that volume alone could be computed. Now, cell quality is always computed, but you may decide not to store the result for each cell. This allows average cell quality of a mesh to be calculated without requiring per-cell storage.

- `obj.RatioOff ()` - These methods are deprecated. Use Get/SetSaveCellQuality() instead.

  Formerly, SetRatio could be used to disable computation of the tetrahedral radius ratio so that volume alone could be computed. Now, cell quality is always computed, but you may decide not to store the result for each cell. This allows average cell quality of a mesh to be calculated without requiring per-cell storage.

- `obj.SetVolume (int cv)` - These methods are deprecated. The functionality of computing cell volume is being removed until it can be computed for any 3D cell. (The previous implementation only worked for tetrahedra.)

  For now, turning on the volume computation will put this filter into "compatibility mode," where tetrahedral cell volume is stored in first component of each output tuple and the radius ratio is stored

in the second component. You may also use CompatibilityModeOn()/Off() to enter this mode. In this mode, cells other than tetrahedra will have report a volume of 0.0 (if volume computation is enabled).

By default, volume computation is disabled and compatibility mode is off, since it does not make a lot of sense for meshes with non-tetrahedral cells.

- `int = obj.GetVolume ()` - These methods are deprecated. The functionality of computing cell volume is being removed until it can be computed for any 3D cell. (The previous implementation only worked for tetrahedra.)

  For now, turning on the volume computation will put this filter into "compatibility mode," where tetrahedral cell volume is stored in first component of each output tuple and the radius ratio is stored in the second component. You may also use CompatibilityModeOn()/Off() to enter this mode. In this mode, cells other than tetrahedra will have report a volume of 0.0 (if volume computation is enabled).

  By default, volume computation is disabled and compatibility mode is off, since it does not make a lot of sense for meshes with non-tetrahedral cells.

- `obj.VolumeOn ()` - These methods are deprecated. The functionality of computing cell volume is being removed until it can be computed for any 3D cell. (The previous implementation only worked for tetrahedra.)

  For now, turning on the volume computation will put this filter into "compatibility mode," where tetrahedral cell volume is stored in first component of each output tuple and the radius ratio is stored in the second component. You may also use CompatibilityModeOn()/Off() to enter this mode. In this mode, cells other than tetrahedra will have report a volume of 0.0 (if volume computation is enabled).

  By default, volume computation is disabled and compatibility mode is off, since it does not make a lot of sense for meshes with non-tetrahedral cells.

- `obj.VolumeOff ()` - These methods are deprecated. The functionality of computing cell volume is being removed until it can be computed for any 3D cell. (The previous implementation only worked for tetrahedra.)

  For now, turning on the volume computation will put this filter into "compatibility mode," where tetrahedral cell volume is stored in first component of each output tuple and the radius ratio is stored in the second component. You may also use CompatibilityModeOn()/Off() to enter this mode. In this mode, cells other than tetrahedra will have report a volume of 0.0 (if volume computation is enabled).

  By default, volume computation is disabled and compatibility mode is off, since it does not make a lot of sense for meshes with non-tetrahedral cells.

- `obj.SetCompatibilityMode (int cm)` - CompatibilityMode governs whether, when both a quality function and cell volume are to be stored as cell data, the two values are stored in a single array. When compatibility mode is off (the default), two separate arrays are used – one labeled "Quality" and the other labeled "Volume". When compatibility mode is on, both values are stored in a single array, with volume as the first component and quality as the second component.

  Enabling CompatibilityMode changes the default tetrahedral quality function to VTK_QUALITY_RADIUS_RATIO and turns volume computation on. (This matches the default behavior of the initial implementation of vtkMeshQuality.) You may change quality function and volume computation without leaving compatibility mode.

  Disabling compatibility mode does not affect the current volume computation or tetrahedral quality function settings.

  The final caveat to CompatibilityMode is that regardless of its setting, the resulting array will be of type vtkDoubleArray rather than the original vtkFloatArray. This is a safety function to keep the authors from diving off of the Combinatorial Coding Cliff into Certain Insanity.

- `int = obj.GetCompatibilityMode ()` - CompatibilityMode governs whether, when both a quality function and cell volume are to be stored as cell data, the two values are stored in a single array. When compatibility mode is off (the default), two separate arrays are used – one labeled "Quality" and the

other labeled "Volume". When compatibility mode is on, both values are stored in a single array, with volume as the first component and quality as the second component.

Enabling CompatibilityMode changes the default tetrahedral quality function to VTK_QUALITY_RADIUS_RATIO and turns volume computation on. (This matches the default behavior of the initial implementation of vtkMeshQuality.) You may change quality function and volume computation without leaving compatibility mode.

Disabling compatibility mode does not affect the current volume computation or tetrahedral quality function settings.

The final caveat to CompatibilityMode is that regardless of its setting, the resulting array will be of type vtkDoubleArray rather than the original vtkFloatArray. This is a safety function to keep the authors from diving off of the Combinatorial Coding Cliff into Certain Insanity.

- `obj.CompatibilityModeOn ()` - CompatibilityMode governs whether, when both a quality function and cell volume are to be stored as cell data, the two values are stored in a single array. When compatibility mode is off (the default), two separate arrays are used – one labeled "Quality" and the other labeled "Volume". When compatibility mode is on, both values are stored in a single array, with volume as the first component and quality as the second component.

  Enabling CompatibilityMode changes the default tetrahedral quality function to VTK_QUALITY_RADIUS_RATIO and turns volume computation on. (This matches the default behavior of the initial implementation of vtkMeshQuality.) You may change quality function and volume computation without leaving compatibility mode.

  Disabling compatibility mode does not affect the current volume computation or tetrahedral quality function settings.

  The final caveat to CompatibilityMode is that regardless of its setting, the resulting array will be of type vtkDoubleArray rather than the original vtkFloatArray. This is a safety function to keep the authors from diving off of the Combinatorial Coding Cliff into Certain Insanity.

- `obj.CompatibilityModeOff ()` - CompatibilityMode governs whether, when both a quality function and cell volume are to be stored as cell data, the two values are stored in a single array. When compatibility mode is off (the default), two separate arrays are used – one labeled "Quality" and the other labeled "Volume". When compatibility mode is on, both values are stored in a single array, with volume as the first component and quality as the second component.

  Enabling CompatibilityMode changes the default tetrahedral quality function to VTK_QUALITY_RADIUS_RATIO and turns volume computation on. (This matches the default behavior of the initial implementation of vtkMeshQuality.) You may change quality function and volume computation without leaving compatibility mode.

  Disabling compatibility mode does not affect the current volume computation or tetrahedral quality function settings.

  The final caveat to CompatibilityMode is that regardless of its setting, the resulting array will be of type vtkDoubleArray rather than the original vtkFloatArray. This is a safety function to keep the authors from diving off of the Combinatorial Coding Cliff into Certain Insanity.

33.148 vtkModelMetadata

33.148.1 Usage

This class is inspired by the Exodus II file format, but because this class does not depend on the Exodus library, it should be possible to use it to represent metadata for other dataset file formats. Sandia Labs uses it in their Exodus II reader, their Exodus II writer and their EnSight writer. vtkDistributedDataFilter looks for metadata attached to it's input and redistributes the metadata with the grid.

The fields in this class are those described in the document "EXODUS II: A Finite Element Data Model", SAND92-2137, November 1995.

Element and node IDs stored in this object must be global IDs, in the event that the original dataset was partitioned across many files.

One way to initialize this object is by using vtkExodusModel (a Sandia class used by the Sandia Exodus reader). That class will take an open Exodus II file and a vtkUnstructuredGrid drawn from it and will set the required fields.

Alternatively, you can use all the Set* methods to set the individual fields. This class does not copy the data, it simply uses your pointer. This class will free the storage associated with your pointer when the class is deleted. Most fields have sensible defaults. The only requirement is that if you are using this ModelMetadata to write out an Exodus or EnSight file in parallel, you must SetBlockIds and SetBlockIdArrayName. Your vtkUnstructuredGrid must have a cell array giving the block ID for each cell.

To create an instance of class vtkModelMetadata, simply invoke its constructor as follows

```
obj = vtkModelMetadata
```

33.148.2 Methods

The class vtkModelMetadata has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkModelMetadata class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkModelMetadata = obj.NewInstance ()`
- `vtkModelMetadata = obj.SafeDownCast (vtkObject o)`
- `obj.PrintGlobalInformation ()`
- `obj.PrintLocalInformation ()`
- `obj.SetTitle (string )` - The title of the dataset.
- `obj.AddInformationLine (string info)` - Add an information line.
- `obj.AddQARecord (string name, string version, string date, string time)` - Add a QA record. They fields are: The code name The code version number The date (MM/DD/YY or NULL for today) The time (HH:MM:SS or NULL for right now)
- `obj.SetTimeStepIndex (int )` - Set the index of the time step represented by the results data in the file attached to this ModelMetadata object. Time step indices start at 0 in this file, they start at 1 in an Exodus file.
- `obj.SetTimeSteps (int numberOfTimeSteps, float timeStepValues)` - Set the total number of time steps in the file, and the value at each time step. We use your time step value array and delete it when we're done.
- `obj.SetNumberOfBlocks (int )` - The number of blocks in the file. Set this before setting any of the block arrays.
- `obj.SetBlockIds (int )` - An arbitrary integer ID for each block. We use your pointer, and free the memory when the object is freed.
- `int = obj.SetBlockNumberOfElements (int nelts)` - Set or get a pointer to a list of the number of elements in each block. We use your pointers, and free the memory when the object is freed.
- `obj.SetBlockNodesPerElement (int )` - Set or get a pointer to a list of the number of nodes in the elements of each block. We use your pointers, and free the memory when the object is freed.

- `obj.SetBlockElementIdList (int )` - Set or get a pointer to a list global element IDs for the elements in each block. We use your pointers, and free the memory when the object is freed.
- `int = obj.SetBlockNumberOfAttributesPerElement (int natts)` - Set or get a pointer to a list of the number of attributes stored for the elements in each block. We use your pointers, and free the memory when the object is freed.
- `obj.SetBlockAttributes (float )` - Set or get a pointer to a list of the attributes for all blocks. The order of the list should be by block, by element within the block, by attribute. Omit blocks that don't have element attributes.
- `obj.SetNumberOfNodeSets (int )` - The number of node sets in the file. Set this value before setting the various node set arrays.
- `obj.SetNodeSetIds (int )` - Set or get the list the IDs for each node set. Length of list is the number of node sets. We use your pointer, and free the memory when the object is freed.
- `int = obj.SetNodeSetSize (int )` - Set or get a pointer to a list of the number of nodes in each node set. We use your pointer, and free the memory when the object is freed.
- `obj.SetNodeSetNodeIdList (int )` - Set or get a pointer to a concatenated list of the IDs of all nodes in each node set. First list all IDs in node set 0, then all IDs in node set 1, and so on. We use your pointer, and free the memory when the object is freed.
- `int = obj.SetNodeSetNumberOfDistributionFactors (int )` - Set or get a list of the number of distribution factors stored by each node set. This is either 0 or equal to the number of nodes in the node set. Length of list is number of node sets. We use your pointer, and free the memory when the object is freed.
- `obj.SetNodeSetDistributionFactors (float )` - Set or get a list of the distribution factors for the node sets. The list is organized by node set, and within node set by node. We use your pointer, and free the memory when the object is freed.
- `obj.SetNumberOfSideSets (int )` - Set or get the number of side sets. Set this value before setting any of the other side set arrays.
- `obj.SetSideSetIds (int )` - Set or get a pointer to a list giving the ID of each side set. We use your pointer, and free the memory when the object is freed.
- `int = obj.SetSideSetSize (int sizes)` - Set or get a pointer to a list of the number of sides in each side set. We use your pointer, and free the memory when the object is freed.
- `int = obj.SetSideSetNumberOfDistributionFactors (int df)` - Set or get a pointer to a list of the number of distribution factors stored by each side set. Each side set has either no distribution factors, or 1 per node in the side set. We use your pointer, and free the memory when the object is freed.
- `obj.SetSideSetElementList (int )` - Set or get a pointer to a list of the elements containing each side in each side set. The list is organized by side set, and within side set by element. We use your pointer, and free the memory when the object is freed.
- `obj.SetSideSetSideList (int )` - Set or get a pointer to the element side for each side in the side set. (See the manual for the convention for numbering sides in different types of cells.) Side Ids are arranged by side set and within side set by side, and correspond to the SideSetElementList. We use your pointer, and free the memory when the object is freed.
- `obj.SetSideSetNumDFPerSide (int numNodes)` - Set or get a pointer to a list of the number of nodes in each side of each side set. This list is organized by side set, and within side set by side. We use your pointer, and free the memory when the object is freed.

- `obj.SetSideSetDistributionFactors (float )` - Set or get a pointer to a list of all the distribution factors. For every side set that has distribution factors, the number of factors per node was given in the SideSetNumberOfDistributionFactors array. If this number for a given side set is N, then for that side set we have N floating point values for each node for each side in the side set. If nodes are repeated in more than one side, we repeat the distribution factors. So this list is in order by side set, by node. We use your pointer, and free the memory when the object is freed.
- `obj.SetBlockPropertyValue (int )` - Set or get value for each variable for each block. List the integer values in order by variable and within variable by block.
- `obj.SetNodeSetPropertyValue (int )` - Set or get value for each variable for each node set. List the integer values in order by variable and within variable by node set.
- `obj.SetSideSetPropertyValue (int )` - Set or get value for each variable for each side set. List the integer values in order by variable and within variable by side set.
- `obj.SetGlobalVariableValue (float f)` - Set or get the values of the global variables at the current time step.
- `obj.SetElementVariableTruthTable (int )` - A truth table indicating which element variables are defined for which blocks. The variables are all the original element variables that were in the file. The table is by block ID and within block ID by variable.
- `obj.SetAllVariablesDefinedInAllBlocks (int )` - Instead of a truth table of all "1"s, you can set this instance variable to indicate that all variables are defined in all blocks.
- `obj.AllVariablesDefinedInAllBlocksOn ()` - Instead of a truth table of all "1"s, you can set this instance variable to indicate that all variables are defined in all blocks.
- `obj.AllVariablesDefinedInAllBlocksOff ()` - Instead of a truth table of all "1"s, you can set this instance variable to indicate that all variables are defined in all blocks.
- `int = obj.ElementVariableIsDefinedInBlock (string varname, int blockId)` - If the element variable named is defined for the block Id provided (in the element variable truth table) return a 1, otherwise return a 0. If the variable name or block Id are unrecognized, the default value of 1 is returned. (This is an "original" variable name, from the file, not a name created for the vtkUnstructuredGrid. Use FindOriginal*VariableName to map between the two.)
- `string = obj.FindOriginalElementVariableName (string name, int component)` - Given the name of an element variable the vtkUnstructuredGrid described by this ModelMetadata, and a component number, give the name of the scalar array in the original file that turned into that component when the file was read into VTK.
- `string = obj.FindOriginalNodeVariableName (string name, int component)` - Given the name of an node variable the vtkUnstructuredGrid described by this ModelMetadata, and a component number, give the name of the scalar array in the original file that turned into that component when the file was read into VTK.
- `obj.Pack (vtkDataSet ugrid)` - Pack this object's metadata into a field array of a dataset.
- `int = obj.Unpack (vtkDataSet ugrid, int deleteIt)` - Unpack the metadata stored in a dataset, and initialize this object with it. Return 1 if there's no metadata packed into the grid, 0 if OK. If deleteIt is ON, then delete the grid's packed data after unpacking it into the object.
- `int = obj.AddUGridElementVariable (string ugridVarName, string origName, int numComponents)` - In order to write Exodus files from vtkUnstructuredGrid objects that were read from Exodus files, we need to know the mapping from variable names in the UGrid to variable names in the Exodus file. (The Exodus reader combines scalar variables with similar names into vectors in the UGrid.) When building the UGrid to which this ModelMetadata refers, add each element and node variable name

with this call, including the name of original variable that yielded it's first component, and the number of components. If a variable is removed from the UGrid, remove it from the ModelMetadata. (If this information is missing or incomplete, the ExodusIIWriter can still do something sensible in creating names for variables.)

- `int = obj.RemoveUGridElementVariable (string ugridVarName)` - In order to write Exodus files from vtkUnstructuredGrid objects that were read from Exodus files, we need to know the mapping from variable names in the UGrid to variable names in the Exodus file. (The Exodus reader combines scalar variables with similar names into vectors in the UGrid.) When building the UGrid to which this ModelMetadata refers, add each element and node variable name with this call, including the name of original variable that yielded it's first component, and the number of components. If a variable is removed from the UGrid, remove it from the ModelMetadata. (If this information is missing or incomplete, the ExodusIIWriter can still do something sensible in creating names for variables.)
- `int = obj.AddUGridNodeVariable (string ugridVarName, string origName, int numComponents)`
- `int = obj.RemoveUGridNodeVariable (string ugridVarName)`
- `int = obj.MergeModelMetadata (vtkModelMetadata em)` - In VTK we take vtkUnstructuredGrids and perform operations on them, including subsetting and merging grids. We need to modify the metadata object when this happens. MergeModelMetadata merges the supplied model (both global and local metadata) into this model. The models must be from the same file set.

  MergeModelMetadata assumes that no element in one metadata object appears in the other. (It doesn't test for duplicate elements when merging the two metadata objects.)
- `int = obj.MergeGlobalInformation (vtkModelMetadata em)` - The metadata is divided into global metadata and local metadata. MergeGlobalInformation merges just the global metadata of the supplied object into the global metadata of this object.
- `vtkModelMetadata = obj.ExtractModelMetadata (vtkIdTypeArray globalCellIdList, vtkDataSet grid)` - Create and return a new metadata object which contains the information for the subset of global cell IDs provided. We need the grid containing the cells so we can find point Ids as well, and also the name of the global cell ID array and the name of the global point ID array.
- `vtkModelMetadata = obj.ExtractGlobalMetadata ()` - Create and return a new metadata object containing only the global metadata of this metadata object.
- `obj.FreeAllGlobalData ()` - Free selected portions of the metadata when updating values in the vtkModelMetadata object. Resetting a particular field, (i.e. SetNodeSetIds) frees the previous setting, but if you are not setting every field, you may want to do a wholesale "Free" first.

  FreeAllGlobalData frees all the fields which don't depend on which time step, which blocks, or which variables are in the input. FreeAllLocalData frees all the fields which do depend on which time step, blocks or variables are in the input. FreeBlockDependentData frees all metadata fields which depend on which blocks were read in.
- `obj.FreeAllLocalData ()` - Free selected portions of the metadata when updating values in the vtkModelMetadata object. Resetting a particular field, (i.e. SetNodeSetIds) frees the previous setting, but if you are not setting every field, you may want to do a wholesale "Free" first.

  FreeAllGlobalData frees all the fields which don't depend on which time step, which blocks, or which variables are in the input. FreeAllLocalData frees all the fields which do depend on which time step, blocks or variables are in the input. FreeBlockDependentData frees all metadata fields which depend on which blocks were read in.
- `obj.FreeBlockDependentData ()` - Free selected portions of the metadata when updating values in the vtkModelMetadata object. Resetting a particular field, (i.e. SetNodeSetIds) frees the previous setting, but if you are not setting every field, you may want to do a wholesale "Free" first.

FreeAllGlobalData frees all the fields which don't depend on which time step, which blocks, or which variables are in the input. FreeAllLocalData frees all the fields which do depend on which time step, blocks or variables are in the input. FreeBlockDependentData frees all metadata fields which depend on which blocks were read in.

- `obj.FreeOriginalElementVariableNames ()` - Free selected portions of the metadata when updating values in the vtkModelMetadata object. Resetting a particular field, (i.e. SetNodeSetIds) frees the previous setting, but if you are not setting every field, you may want to do a wholesale "Free" first.

  FreeAllGlobalData frees all the fields which don't depend on which time step, which blocks, or which variables are in the input. FreeAllLocalData frees all the fields which do depend on which time step, blocks or variables are in the input. FreeBlockDependentData frees all metadata fields which depend on which blocks were read in.

- `obj.FreeOriginalNodeVariableNames ()` - Free selected portions of the metadata when updating values in the vtkModelMetadata object. Resetting a particular field, (i.e. SetNodeSetIds) frees the previous setting, but if you are not setting every field, you may want to do a wholesale "Free" first.

  FreeAllGlobalData frees all the fields which don't depend on which time step, which blocks, or which variables are in the input. FreeAllLocalData frees all the fields which do depend on which time step, blocks or variables are in the input. FreeBlockDependentData frees all metadata fields which depend on which blocks were read in.

- `obj.FreeUsedElementVariableNames ()` - Free selected portions of the metadata when updating values in the vtkModelMetadata object. Resetting a particular field, (i.e. SetNodeSetIds) frees the previous setting, but if you are not setting every field, you may want to do a wholesale "Free" first.

  FreeAllGlobalData frees all the fields which don't depend on which time step, which blocks, or which variables are in the input. FreeAllLocalData frees all the fields which do depend on which time step, blocks or variables are in the input. FreeBlockDependentData frees all metadata fields which depend on which blocks were read in.

- `obj.FreeUsedNodeVariableNames ()` - Free selected portions of the metadata when updating values in the vtkModelMetadata object. Resetting a particular field, (i.e. SetNodeSetIds) frees the previous setting, but if you are not setting every field, you may want to do a wholesale "Free" first.

  FreeAllGlobalData frees all the fields which don't depend on which time step, which blocks, or which variables are in the input. FreeAllLocalData frees all the fields which do depend on which time step, blocks or variables are in the input. FreeBlockDependentData frees all metadata fields which depend on which blocks were read in.

- `obj.FreeUsedElementVariables ()` - Free selected portions of the metadata when updating values in the vtkModelMetadata object. Resetting a particular field, (i.e. SetNodeSetIds) frees the previous setting, but if you are not setting every field, you may want to do a wholesale "Free" first.

  FreeAllGlobalData frees all the fields which don't depend on which time step, which blocks, or which variables are in the input. FreeAllLocalData frees all the fields which do depend on which time step, blocks or variables are in the input. FreeBlockDependentData frees all metadata fields which depend on which blocks were read in.

- `obj.FreeUsedNodeVariables ()` - Free selected portions of the metadata when updating values in the vtkModelMetadata object. Resetting a particular field, (i.e. SetNodeSetIds) frees the previous setting, but if you are not setting every field, you may want to do a wholesale "Free" first.

  FreeAllGlobalData frees all the fields which don't depend on which time step, which blocks, or which variables are in the input. FreeAllLocalData frees all the fields which do depend on which time step, blocks or variables are in the input. FreeBlockDependentData frees all metadata fields which depend on which blocks were read in.

- `obj.Reset ()` - Set the object back to it's initial state

- `int = obj.GetBlockLocalIndex (int id)` - Block information is stored in arrays. This method returns the array index for a given block ID.

33.149 vtkMultiBlockDataGroupFilter

33.149.1 Usage

vtkMultiBlockDataGroupFilter is an M to 1 filter that merges multiple input into one multi-group dataset. It will assign each input to one group of the multi-group dataset and will assign each update piece as a sub-block. For example, if there are two inputs and four update pieces, the output contains two groups with four datasets each.

To create an instance of class vtkMultiBlockDataGroupFilter, simply invoke its constructor as follows

```
obj = vtkMultiBlockDataGroupFilter
```

33.149.2 Methods

The class vtkMultiBlockDataGroupFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiBlockDataGroupFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiBlockDataGroupFilter = obj.NewInstance ()`
- `vtkMultiBlockDataGroupFilter = obj.SafeDownCast (vtkObject o)`
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

33.150 vtkMultiBlockMergeFilter

33.150.1 Usage

vtkMultiBlockMergeFilter is an M to 1 filter similar to vtkMultiBlockDataGroupFilter. However where as that class creates N groups in the output for N inputs, this creates 1 group in the output with N datasets inside it. In actuality if the inputs have M blocks, this will produce M blocks, each of which has N datasets. Inside the merged group, the i'th data set comes from the i'th data set in the i'th input.

To create an instance of class vtkMultiBlockMergeFilter, simply invoke its constructor as follows

```
obj = vtkMultiBlockMergeFilter
```

33.150.2 Methods

The class vtkMultiBlockMergeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiBlockMergeFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiBlockMergeFilter = obj.NewInstance ()`
- `vtkMultiBlockMergeFilter = obj.SafeDownCast (vtkObject o)`
- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

33.151 vtkMultiThreshold

33.151.1 Usage

This filter can be substituted for a chain of several vtkThreshold filters and can also perform more sophisticated subsetting operations. It generates a vtkMultiBlockDataSet as its output. This multiblock dataset contains a vtkUnstructuredGrid for each thresholded subset you request. A thresholded subset can be a set defined by an interval over a point or cell attribute of the mesh; these subsets are called IntervalSets. A thresholded subset can also be a boolean combination of one or more IntervalSets; these subsets are called BooleanSets. BooleanSets allow complex logic since their output can depend on multiple intervals over multiple variables defined on the input mesh. This is useful because it eliminates the need for thresholding several times and then appending the results, as can be required with vtkThreshold when one wants to remove some range of values (e.g., a notch filter). Cells are not repeated when they belong to more than one interval unless those intervals have different output grids.

Another advantage this filter provides over vtkThreshold is the ability to threshold on non-scalar (i.e., vector, tensor, etc.) attributes without first computing an array containing some norm of the desired attribute. vtkMultiThreshold provides L_1, L_2, and L_∞ norms.

This filter makes a distinction between intermediate subsets and subsets that will be output to a grid. Each intermediate subset you create with AddIntervalSet or AddBooleanSet is given a unique integer identifier (via the return values of these member functions). If you wish for a given set to be output, you must call OutputSet and pass it one of these identifiers. The return of OutputSet is the integer index of the output set in the multiblock dataset created by this filter.

For example, if an input mesh defined three attributes T, P, and s, one might wish to find cells that satisfy "T ¡ 320 [K] && (P ¿ 101 [kPa] —— s ¡ 0.1 [kJ/kg/K])". To accomplish this with a vtkMultiThreshold filter,

```
vtkMultiThreshold* thr;
int intervalSets[3];

intervalSets[0] = thr->AddIntervalSet( vtkMath::NegInf(), 320., vtkMultiThreshold::CLOSED, vtkMultiThr
    vtkDataObject::FIELD\_ASSOCIATION\_POINTS, ''T'', 0, 1 );
intervalSets[1] = thr->AddIntervalSet( 101., vtkMath::Inf(), vtkMultiThreshold::OPEN, vtkMultiThreshol
    vtkDataObject::FIELD\_ASSOCIATION\_CELLS, ''P'', 0, 1 );
intervalSets[2] = thr->AddIntervalSet( vtkMath::NegInf(), 0.1, vtkMultiThreshold::CLOSED, vtkMultiThres
    vtkDataObject::FIELD\_ASSOCIATION\_POINTS, ''s'', 0, 1 );

int intermediate = thr->AddBooleanSet( vtkMultiThreshold::OR, 2, \&intervalSets[1] );

int intersection[2];
intersection[0] = intervalSets[0];
```

```
intersection[1] = intermediate;
int outputSet = thr->AddBooleanSet( vtkMultiThreshold::AND, 2, intersection );

int outputGridIndex = thr->OutputSet( outputSet );
thr->Update();
```

The result of this filter will be a multiblock dataset that contains a single child with the desired cells. If we had also called ¡code¿thr-¿OutputSet(intervalSets[0]);¡/code¿, there would be two child meshes and one would contain all cells with T ¡ 320 [K]. In that case, the output can be represented by this graph

```
digraph MultiThreshold {
  set0 [shape=rect,style=filled,label=''point T(0) in [-Inf,320['']
  set1 [shape=rect,label=''cell P(0) in ]101,Inf]'']
  set2 [shape=rect,label=''point s(0) in [-Inf,0.1['']
  set3 [shape=rect,label=''OR'']
  set4 [shape=rect,style=filled,label=''AND'']
  set0 -> set4
  set1 -> set3
  set2 -> set3
  set3 -> set4
}
```

The filled rectangles represent sets that are output.

To create an instance of class vtkMultiThreshold, simply invoke its constructor as follows

```
obj = vtkMultiThreshold
```

33.151.2 Methods

The class vtkMultiThreshold has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiThreshold class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiThreshold = obj.NewInstance ()`
- `vtkMultiThreshold = obj.SafeDownCast (vtkObject o)`
- `int = obj.AddIntervalSet (double xmin, double xmax, int omin, int omax, int assoc, string arrayName`
  - Add a mesh subset to be computed by thresholding an attribute of the input mesh. The subset can then be added to an output mesh with OuputSet() or combined with other sets using AddBooleanSet. If you wish to include all cells with values below some number a, call with xmin set to vtkMath::NegInf() and xmax set to a. Similarly, if you wish to include all cells with values above some number a, call with xmin set to a and xmax set to vtkMath::Inf(). When specifying Inf() or NegInf() for an endpoint, it does not matter whether you specify and open or closed endpoint.

  When creating intervals, any integers can be used for the IDs of output meshes. All that matters is that the same ID be used if intervals should output to the same mesh. The outputs are ordered with ascending IDs in output block 0.

  It is possible to specify an invalid interval, in which case these routines will return -1. Invalid intervals occur when - an array does not exist, - center is invalid, - xmin == xmax and omin and/or omax are vtkMultiThreshold::OPEN, or - xmin ¿ xmax. - xmin or xmax is not a number (i.e., IEEE NaN).

Having both xmin and xmax equal NaN is allowed. vtkMath provides a portable way to specify IEEE infinities and Nan. Note that specifying an interval completely out of the bounds of an attribute is considered valid. In fact, it is occasionally useful to create a closed interval with both endpoints set to ∞ or both endpoints set to $-\infty$ in order to locate cells with problematic values.

@param xmin The minimum attribute value @param xmax The maximum attribute value @param omin Whether the interval should be open or closed at xmin. Use vtkMultiThreshold::OPEN or vtkMultiThreshold::CLOSED. @param omax Whether the interval should be open or closed at xmax. Use vtkMultiThreshold::OPEN or vtkMultiThreshold::CLOSED. @param assoc One of vtkDataObject::FIELD_ASSOCIATION_CELLS or vtkDataObject::FIELD_ASSOCIATION_POINTS indicating whether a point or cell array should be used. @param arrayName The name of the array to use for thresholding @param attribType The attribute to use for thresholding. One of vtkDataSetAttributes::SCALARS, VECTORS, TENSORS, NORMALS, TCOORDS, or GLOBALIDS. @param component The number of the component to threshold on or one of the following enumerants for norms: LINFINITY_NORM, L2_NORM, L1_NORM. @param allScalars When center is vtkDataObject::FIELD_ASSOCIATION must all scalars be in the interval for the cell to be passed to the output, or just a single point's scalar? @return An index used to identify the cells selected by the interval or -1 if the interval specification was invalid. If a valid value is returned, you may pass it to OutputSet().

- `int = obj.AddIntervalSet (double xmin, double xmax, int omin, int omax, int assoc, int attribType,` - Add a mesh subset to be computed by thresholding an attribute of the input mesh. The subset can then be added to an output mesh with OuputSet() or combined with other sets using AddBooleanSet. If you wish to include all cells with values below some number a, call with xmin set to vtkMath::NegInf() and xmax set to a. Similarly, if you wish to include all cells with values above some number a, call with xmin set to a and xmax set to vtkMath::Inf(). When specifying Inf() or NegInf() for an endpoint, it does not matter whether you specify and open or closed endpoint.

  When creating intervals, any integers can be used for the IDs of output meshes. All that matters is that the same ID be used if intervals should output to the same mesh. The outputs are ordered with ascending IDs in output block 0.

  It is possible to specify an invalid interval, in which case these routines will return -1. Invalid intervals occur when - an array does not exist, - center is invalid, - xmin == xmax and omin and/or omax are vtkMultiThreshold::OPEN, or - xmin ¿ xmax. - xmin or xmax is not a number (i.e., IEEE NaN). Having both xmin and xmax equal NaN is allowed. vtkMath provides a portable way to specify IEEE infinities and Nan. Note that specifying an interval completely out of the bounds of an attribute is considered valid. In fact, it is occasionally useful to create a closed interval with both endpoints set to ∞ or both endpoints set to $-\infty$ in order to locate cells with problematic values.

  @param xmin The minimum attribute value @param xmax The maximum attribute value @param omin Whether the interval should be open or closed at xmin. Use vtkMultiThreshold::OPEN or vtkMultiThreshold::CLOSED. @param omax Whether the interval should be open or closed at xmax. Use vtkMultiThreshold::OPEN or vtkMultiThreshold::CLOSED. @param assoc One of vtkDataObject::FIELD_ASSOCIATION_CELLS or vtkDataObject::FIELD_ASSOCIATION_POINTS indicating whether a point or cell array should be used. @param arrayName The name of the array to use for thresholding @param attribType The attribute to use for thresholding. One of vtkDataSetAttributes::SCALARS, VECTORS, TENSORS, NORMALS, TCOORDS, or GLOBALIDS. @param component The number of the component to threshold on or one of the following enumerants for norms: LINFINITY_NORM, L2_NORM, L1_NORM. @param allScalars When center is vtkDataObject::FIELD_ASSOCIATION must all scalars be in the interval for the cell to be passed to the output, or just a single point's scalar? @return An index used to identify the cells selected by the interval or -1 if the interval specification was invalid. If a valid value is returned, you may pass it to OutputSet().

- `int = obj.AddLowpassIntervalSet (double xmax, int assoc, string arrayName, int component, int allSc` - These convenience members make it easy to insert closed intervals. The "notch" interval is accomplished by creating a bandpass interval and applying a NAND operation. In this case, the set ID returned in the NAND operation set ID. Note that you can pass xmin == xmax when creating a

bandpass threshold to retrieve elements matching exactly one value (since the intervals created by these routines are closed).

- `int = obj.AddHighpassIntervalSet (double xmin, int assoc, string arrayName, int component, int allS` - These convenience members make it easy to insert closed intervals. The "notch" interval is accomplished by creating a bandpass interval and applying a NAND operation. In this case, the set ID returned in the NAND operation set ID. Note that you can pass xmin == xmax when creating a bandpass threshold to retrieve elements matching exactly one value (since the intervals created by these routines are closed).
- `int = obj.AddBandpassIntervalSet (double xmin, double xmax, int assoc, string arrayName, int compor` - These convenience members make it easy to insert closed intervals. The "notch" interval is accomplished by creating a bandpass interval and applying a NAND operation. In this case, the set ID returned in the NAND operation set ID. Note that you can pass xmin == xmax when creating a bandpass threshold to retrieve elements matching exactly one value (since the intervals created by these routines are closed).
- `int = obj.AddNotchIntervalSet (double xlo, double xhi, int assoc, string arrayName, int component,` - These convenience members make it easy to insert closed intervals. The "notch" interval is accomplished by creating a bandpass interval and applying a NAND operation. In this case, the set ID returned in the NAND operation set ID. Note that you can pass xmin == xmax when creating a bandpass threshold to retrieve elements matching exactly one value (since the intervals created by these routines are closed).
- `int = obj.AddBooleanSet (int operation, int numInputs, int inputs)` - Create a new mesh subset using boolean operations on pre-existing sets.
- `int = obj.OutputSet (int setId)` - Create an output mesh containing a boolean or interval subset of the input mesh.
- `obj.Reset ()` - Remove all the intervals currently defined.

33.152 vtkOBBDicer

33.152.1 Usage

vtkOBBDicer separates the cells of a dataset into spatially aggregated pieces using a Oriented Bounding Box (OBB). These pieces can then be operated on by other filters (e.g., vtkThreshold). One application is to break very large polygonal models into pieces and performing viewing and occlusion culling on the pieces.

Refer to the superclass documentation (vtkDicer) for more information.

To create an instance of class vtkOBBDicer, simply invoke its constructor as follows

```
obj = vtkOBBDicer
```

33.152.2 Methods

The class vtkOBBDicer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOBBDicer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOBBDicer = obj.NewInstance ()`
- `vtkOBBDicer = obj.SafeDownCast (vtkObject o)`

33.153 vtkOBBTree

33.153.1 Usage

vtkOBBTree is an object to generate oriented bounding box (OBB) trees. An oriented bounding box is a bounding box that does not necessarily line up along coordinate axes. The OBB tree is a hierarchical tree structure of such boxes, where deeper levels of OBB confine smaller regions of space.

To build the OBB, a recursive, top-down process is used. First, the root OBB is constructed by finding the mean and covariance matrix of the cells (and their points) that define the dataset. The eigenvectors of the covariance matrix are extracted, giving a set of three orthogonal vectors that define the tightest-fitting OBB. To create the two children OBB's, a split plane is found that (approximately) divides the number cells in half. These are then assigned to the children OBB's. This process then continues until the MaxLevel ivar limits the recursion, or no split plane can be found.

A good reference for OBB-trees is Gottschalk & Manocha in Proceedings of Siggraph '96.

To create an instance of class vtkOBBTree, simply invoke its constructor as follows

```
obj = vtkOBBTree
```

33.153.2 Methods

The class vtkOBBTree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOBBTree class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOBBTree = obj.NewInstance ()`
- `vtkOBBTree = obj.SafeDownCast (vtkObject o)`
- `int = obj.IntersectWithLine (double a0[3], double a1[3], vtkPoints points, vtkIdList cellIds)` - Take the passed line segment and intersect it with the data set. This method assumes that the data set is a vtkPolyData that describes a closed surface, and the intersection points that are returned in 'points' alternate between entrance points and exit points. The return value of the function is 0 if no intersections were found, -1 if point 'a0' lies inside the closed surface, or +1 if point 'a0' lies outside the closed surface. Either 'points' or 'cellIds' can be set to NULL if you don't want to receive that information.
- `obj.ComputeOBB (vtkDataSet input, double corner[3], double max[3], double mid[3], double min[3], do` - Compute an OBB for the input dataset using the cells in the data. Return the corner point and the three axes defining the orientation of the OBB. Also return a sorted list of relative "sizes" of axes for comparison purposes.
- `int = obj.InsideOrOutside (double point[3])` - Determine whether a point is inside or outside the data used to build this OBB tree. The data must be a closed surface vtkPolyData data set. The return value is +1 if outside, -1 if inside, and 0 if undecided.
- `obj.FreeSearchStructure ()` - Satisfy locator's abstract interface, see vtkLocator.
- `obj.BuildLocator ()` - Satisfy locator's abstract interface, see vtkLocator.
- `obj.GenerateRepresentation (int level, vtkPolyData pd)` - Create polygonal representation for OBB tree at specified level. If level ¡ 0, then the leaf OBB nodes will be gathered. The aspect ratio (ar) and line diameter (d) are used to control the building of the representation. If a OBB node edge ratio's are greater than ar, then the dimension of the OBB is collapsed (OBB-¿plane-¿line). A "line" OBB will be represented either as two crossed polygons, or as a line, depending on the relative diameter of the OBB compared to the diameter (d).

33.154 vtkOutlineCornerFilter

33.154.1 Usage

vtkOutlineCornerFilter is a filter that generates wireframe outline corners of any data set. The outline consists of the eight corners of the dataset bounding box.

To create an instance of class vtkOutlineCornerFilter, simply invoke its constructor as follows

```
obj = vtkOutlineCornerFilter
```

33.154.2 Methods

The class vtkOutlineCornerFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOutlineCornerFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOutlineCornerFilter = obj.NewInstance ()`
- `vtkOutlineCornerFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetCornerFactor (double )` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `double = obj.GetCornerFactorMinValue ()` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `double = obj.GetCornerFactorMaxValue ()` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `double = obj.GetCornerFactor ()` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds

33.155 vtkOutlineCornerSource

33.155.1 Usage

vtkOutlineCornerSource creates wireframe outline corners around a user-specified bounding box.

To create an instance of class vtkOutlineCornerSource, simply invoke its constructor as follows

```
obj = vtkOutlineCornerSource
```

33.155.2 Methods

The class vtkOutlineCornerSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOutlineCornerSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOutlineCornerSource = obj.NewInstance ()`
- `vtkOutlineCornerSource = obj.SafeDownCast (vtkObject o)`

- `obj.SetCornerFactor (double )` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `double = obj.GetCornerFactorMinValue ()` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `double = obj.GetCornerFactorMaxValue ()` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `double = obj.GetCornerFactor ()` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds

33.156 vtkOutlineFilter

33.156.1 Usage

vtkOutlineFilter is a filter that generates a wireframe outline of any data set. The outline consists of the twelve edges of the dataset bounding box.

To create an instance of class vtkOutlineFilter, simply invoke its constructor as follows

```
obj = vtkOutlineFilter
```

33.156.2 Methods

The class vtkOutlineFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOutlineFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOutlineFilter = obj.NewInstance ()`
- `vtkOutlineFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetGenerateFaces (int )` - Generate solid faces for the box. This is off by default.
- `obj.GenerateFacesOn ()` - Generate solid faces for the box. This is off by default.
- `obj.GenerateFacesOff ()` - Generate solid faces for the box. This is off by default.
- `int = obj.GetGenerateFaces ()` - Generate solid faces for the box. This is off by default.

33.157 vtkOutlineSource

33.157.1 Usage

vtkOutlineSource creates a wireframe outline around a user-specified bounding box. The outline may be created aligned with the x,y,z axis - in which case it is defined by the 6 bounds xmin,xmax,ymin,ymax,zmin,zmax via SetBounds(). Alternatively, the box may be arbitrarily aligned, in which case it should be set via the SetCorners() member.

To create an instance of class vtkOutlineSource, simply invoke its constructor as follows

```
obj = vtkOutlineSource
```

33.157.2 Methods

The class vtkOutlineSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOutlineSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOutlineSource = obj.NewInstance ()`
- `vtkOutlineSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetBoxType (int )` - Set box type to AxisAligned (default) or Oriented. Use the method SetBounds() with AxisAligned mode, and SetCorners() with Oriented mode.
- `int = obj.GetBoxType ()` - Set box type to AxisAligned (default) or Oriented. Use the method SetBounds() with AxisAligned mode, and SetCorners() with Oriented mode.
- `obj.SetBoxTypeToAxisAligned ()` - Set box type to AxisAligned (default) or Oriented. Use the method SetBounds() with AxisAligned mode, and SetCorners() with Oriented mode.
- `obj.SetBoxTypeToOriented ()` - Specify the bounds of the box to be used in Axis Aligned mode.
- `obj.SetBounds (double , double , double , double , double , double )` - Specify the bounds of the box to be used in Axis Aligned mode.
- `obj.SetBounds (double a[6])` - Specify the bounds of the box to be used in Axis Aligned mode.
- `double = obj. GetBounds ()` - Specify the bounds of the box to be used in Axis Aligned mode.
- `obj.SetCorners (double [24])` - Specify the corners of the outline when in Oriented mode, the values are supplied as 8*3 double values The correct corner ordering is using x,y,z convention for the unit cube as follows: 0,0,0,1,0,0,0,1,0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,1.
- `double = obj. GetCorners ()` - Specify the corners of the outline when in Oriented mode, the values are supplied as 8*3 double values The correct corner ordering is using x,y,z convention for the unit cube as follows: 0,0,0,1,0,0,0,1,0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,1.
- `obj.SetGenerateFaces (int )` - Generate solid faces for the box. This is off by default.
- `obj.GenerateFacesOn ()` - Generate solid faces for the box. This is off by default.
- `obj.GenerateFacesOff ()` - Generate solid faces for the box. This is off by default.
- `int = obj.GetGenerateFaces ()` - Generate solid faces for the box. This is off by default.

33.158 vtkParametricFunctionSource

33.158.1 Usage

This class tessellates parametric functions. The user must specify how many points in the parametric coordinate directions are required (i.e., the resolution), and the mode to use to generate scalars.

.SECTION Thanks Andrew Maclean a.maclean@cas.edu.au for creating and contributing the class.

To create an instance of class vtkParametricFunctionSource, simply invoke its constructor as follows

```
obj = vtkParametricFunctionSource
```

33.158.2 Methods

The class vtkParametricFunctionSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParametricFunctionSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParametricFunctionSource = obj.NewInstance ()`
- `vtkParametricFunctionSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetParametricFunction (vtkParametricFunction )` - Specify the parametric function to use to generate the tessellation.
- `vtkParametricFunction = obj.GetParametricFunction ()` - Specify the parametric function to use to generate the tessellation.
- `obj.SetUResolution (int )` - Set/Get the number of subdivisions / tessellations in the u parametric direction. Note that the number of tessellant points in the u direction is the UResolution + 1.
- `int = obj.GetUResolution ()` - Set/Get the number of subdivisions / tessellations in the u parametric direction. Note that the number of tessellant points in the u direction is the UResolution + 1.
- `obj.SetVResolution (int )` - Set/Get the number of subdivisions / tessellations in the v parametric direction. Note that the number of tessellant points in the v direction is the VResolution + 1.
- `int = obj.GetVResolution ()` - Set/Get the number of subdivisions / tessellations in the v parametric direction. Note that the number of tessellant points in the v direction is the VResolution + 1.
- `obj.SetWResolution (int )` - Set/Get the number of subdivisions / tessellations in the w parametric direction. Note that the number of tessellant points in the w direction is the WResolution + 1.
- `int = obj.GetWResolution ()` - Set/Get the number of subdivisions / tessellations in the w parametric direction. Note that the number of tessellant points in the w direction is the WResolution + 1.
- `obj.GenerateTextureCoordinatesOn ()` - Set/Get the generation of texture coordinates. This is off by default. Note that this is only applicable to parametric surfaces whose parametric dimension is 2. Note that texturing may fail in some cases.
- `obj.GenerateTextureCoordinatesOff ()` - Set/Get the generation of texture coordinates. This is off by default. Note that this is only applicable to parametric surfaces whose parametric dimension is 2. Note that texturing may fail in some cases.
- `obj.SetGenerateTextureCoordinates (int )` - Set/Get the generation of texture coordinates. This is off by default. Note that this is only applicable to parametric surfaces whose parametric dimension is 2. Note that texturing may fail in some cases.
- `int = obj.GetGenerateTextureCoordinates ()` - Set/Get the generation of texture coordinates. This is off by default. Note that this is only applicable to parametric surfaces whose parametric dimension is 2. Note that texturing may fail in some cases.

- `obj.SetScalarMode (int )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().
- `int = obj.GetScalarModeMinValue ()` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().
- `int = obj.GetScalarModeMaxValue ()` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().
- `int = obj.GetScalarMode ()` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().
- `obj.SetScalarModeToNone (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg,

0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToU (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToV (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToU0 (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToV0 (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending

upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToU0V0 (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToModulus (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToPhase (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToQuadrant (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToX (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToY (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToZ (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToDistance (void )` - Get/Set the mode used for the scalar data. The options are: SCALAR_NONE, (default) scalars are not generated. SCALAR_U, the scalar is set to the u-value. SCALAR_V, the scalar is set to the v-value. SCALAR_U0, the scalar is set to 1 if u = (u_max - u_min)/2 = u_avg, 0 otherwise. SCALAR_V0, the scalar is set to 1 if v = (v_max - v_min)/2 = v_avg, 0 otherwise. SCALAR_U0V0, the scalar is set to 1 if u == u_avg, 2 if v == v_avg, 3 if u = u_avg && v = v_avg, 0 otherwise. SCALAR_MODULUS, the scalar is set to (sqrt(u*u+v*v)), this is measured relative to (u_avg,v_avg). SCALAR_PHASE, the scalar is set to (atan2(v,u)) (in degrees, 0 to 360), this is measured relative to (u_avg,v_avg). SCALAR_QUADRANT, the scalar is set to 1, 2, 3 or 4 depending upon the quadrant of the point (u,v). SCALAR_X, the scalar is set to the x-value. SCALAR_Y, the scalar is set to the y-value. SCALAR_Z, the scalar is set to the z-value. SCALAR_DISTANCE, the scalar is set to (sqrt(x*x+y*y+z*z)). I.e. distance from the origin. SCALAR_FUNCTION_DEFINED, the scalar is set to the value returned from EvaluateScalar().

- `obj.SetScalarModeToFunctionDefined (void )` - Return the MTime also considering the parametric function.

- `long = obj.GetMTime ()` - Return the MTime also considering the parametric function.

33.159 vtkPlaneSource

33.159.1 Usage

vtkPlaneSource creates an m x n array of quadrilaterals arranged as a regular tiling in a plane. The plane is defined by specifying an origin point, and then two other points that, together with the origin, define two axes for the plane. These axes do not have to be orthogonal - so you can create a parallelogram. (The axes must not be parallel.) The resolution of the plane (i.e., number of subdivisions) is controlled by the ivars XResolution and YResolution.

By default, the plane is centered at the origin and perpendicular to the z-axis, with width and height of length 1 and resolutions set to 1.

There are three convenience methods that allow you to easily move the plane. The first, SetNormal(), allows you to specify the plane normal. The effect of this method is to rotate the plane around the center of the plane, aligning the plane normal with the specified normal. The rotation is about the axis defined by the cross product of the current normal with the new normal. The second, SetCenter(), translates the center of the plane to the specified center point. The third method, Push(), allows you to translate the plane along the plane normal by the distance specified. (Negative Push values translate the plane in the negative normal direction.) Note that the SetNormal(), SetCenter() and Push() methods modify the Origin, Point1, and/or Point2 instance variables.

To create an instance of class vtkPlaneSource, simply invoke its constructor as follows

```
obj = vtkPlaneSource
```

33.159.2 Methods

The class vtkPlaneSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPlaneSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPlaneSource = obj.NewInstance ()`
- `vtkPlaneSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetXResolution (int )` - Specify the resolution of the plane along the first axes.
- `int = obj.GetXResolution ()` - Specify the resolution of the plane along the first axes.
- `obj.SetYResolution (int )` - Specify the resolution of the plane along the second axes.
- `int = obj.GetYResolution ()` - Specify the resolution of the plane along the second axes.
- `obj.SetResolution (int xR, int yR)` - Set the number of x-y subdivisions in the plane.
- `obj.SetOrigin (double , double , double )` - Specify a point defining the origin of the plane.
- `obj.SetOrigin (double  a[3])` - Specify a point defining the origin of the plane.
- `double = obj. GetOrigin ()` - Specify a point defining the origin of the plane.
- `obj.SetPoint1 (double x, double y, double z)` - Specify a point defining the first axis of the plane.
- `obj.SetPoint1 (double pnt[3])` - Specify a point defining the first axis of the plane.
- `double = obj. GetPoint1 ()` - Specify a point defining the first axis of the plane.

- `obj.SetPoint2 (double x, double y, double z)` - Specify a point defining the second axis of the plane.
- `obj.SetPoint2 (double pnt[3])` - Specify a point defining the second axis of the plane.
- `double = obj. GetPoint2 ()` - Specify a point defining the second axis of the plane.
- `obj.SetCenter (double x, double y, double z)` - Set/Get the center of the plane. Works in conjunction with the plane normal to position the plane. Don't use this method to define the plane. Instead, use it to move the plane to a new center point.
- `obj.SetCenter (double center[3])` - Set/Get the center of the plane. Works in conjunction with the plane normal to position the plane. Don't use this method to define the plane. Instead, use it to move the plane to a new center point.
- `double = obj. GetCenter ()` - Set/Get the center of the plane. Works in conjunction with the plane normal to position the plane. Don't use this method to define the plane. Instead, use it to move the plane to a new center point.
- `obj.SetNormal (double nx, double ny, double nz)` - Set/Get the plane normal. Works in conjunction with the plane center to orient the plane. Don't use this method to define the plane. Instead, use it to rotate the plane around the current center point.
- `obj.SetNormal (double n[3])` - Set/Get the plane normal. Works in conjunction with the plane center to orient the plane. Don't use this method to define the plane. Instead, use it to rotate the plane around the current center point.
- `double = obj. GetNormal ()` - Set/Get the plane normal. Works in conjunction with the plane center to orient the plane. Don't use this method to define the plane. Instead, use it to rotate the plane around the current center point.
- `obj.Push (double distance)` - Translate the plane in the direction of the normal by the distance specified. Negative values move the plane in the opposite direction.

33.160 vtkPlatonicSolidSource

33.160.1 Usage

vtkPlatonicSolidSource can generate each of the five Platonic solids: tetrahedron, cube, octahedron, icosahedron, and dodecahedron. Each of the solids is placed inside a sphere centered at the origin with radius 1.0. To use this class, simply specify the solid to create. Note that this source object creates cell scalars that are (integral value) face numbers.

To create an instance of class vtkPlatonicSolidSource, simply invoke its constructor as follows

```
obj = vtkPlatonicSolidSource
```

33.160.2 Methods

The class vtkPlatonicSolidSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPlatonicSolidSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPlatonicSolidSource = obj.NewInstance ()`

- `vtkPlatonicSolidSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetSolidType (int )` - Specify the type of PlatonicSolid solid to create.
- `int = obj.GetSolidTypeMinValue ()` - Specify the type of PlatonicSolid solid to create.
- `int = obj.GetSolidTypeMaxValue ()` - Specify the type of PlatonicSolid solid to create.
- `int = obj.GetSolidType ()` - Specify the type of PlatonicSolid solid to create.
- `obj.SetSolidTypeToTetrahedron ()` - Specify the type of PlatonicSolid solid to create.
- `obj.SetSolidTypeToCube ()` - Specify the type of PlatonicSolid solid to create.
- `obj.SetSolidTypeToOctahedron ()` - Specify the type of PlatonicSolid solid to create.
- `obj.SetSolidTypeToIcosahedron ()` - Specify the type of PlatonicSolid solid to create.
- `obj.SetSolidTypeToDodecahedron ()`

33.161 vtkPointDataToCellData

33.161.1 Usage

vtkPointDataToCellData is a filter that transforms point data (i.e., data specified per point) into cell data (i.e., data specified per cell). The method of transformation is based on averaging the data values of all points defining a particular cell. Optionally, the input point data can be passed through to the output as well.

To create an instance of class vtkPointDataToCellData, simply invoke its constructor as follows

```
obj = vtkPointDataToCellData
```

33.161.2 Methods

The class vtkPointDataToCellData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointDataToCellData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointDataToCellData = obj.NewInstance ()`
- `vtkPointDataToCellData = obj.SafeDownCast (vtkObject o)`
- `obj.SetPassPointData (int )` - Control whether the input point data is to be passed to the output. If on, then the input point data is passed through to the output; otherwise, only generated point data is placed into the output.
- `int = obj.GetPassPointData ()` - Control whether the input point data is to be passed to the output. If on, then the input point data is passed through to the output; otherwise, only generated point data is placed into the output.
- `obj.PassPointDataOn ()` - Control whether the input point data is to be passed to the output. If on, then the input point data is passed through to the output; otherwise, only generated point data is placed into the output.
- `obj.PassPointDataOff ()` - Control whether the input point data is to be passed to the output. If on, then the input point data is passed through to the output; otherwise, only generated point data is placed into the output.

33.162 vtkPointSource

33.162.1 Usage

vtkPointSource is a source object that creates a user-specified number of points within a specified radius about a specified center point. By default location of the points is random within the sphere. It is also possible to generate random points only on the surface of the sphere.

To create an instance of class vtkPointSource, simply invoke its constructor as follows

```
obj = vtkPointSource
```

33.162.2 Methods

The class vtkPointSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointSource = obj.NewInstance ()`
- `vtkPointSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfPoints (vtkIdType )` - Set the number of points to generate.
- `vtkIdType = obj.GetNumberOfPointsMinValue ()` - Set the number of points to generate.
- `vtkIdType = obj.GetNumberOfPointsMaxValue ()` - Set the number of points to generate.
- `vtkIdType = obj.GetNumberOfPoints ()` - Set the number of points to generate.
- `obj.SetCenter (double , double , double )` - Set the center of the point cloud.
- `obj.SetCenter (double  a[3])` - Set the center of the point cloud.
- `double = obj. GetCenter ()` - Set the center of the point cloud.
- `obj.SetRadius (double )` - Set the radius of the point cloud. If you are generating a Gaussian distribution, then this is the standard deviation for each of x, y, and z.
- `double = obj.GetRadiusMinValue ()` - Set the radius of the point cloud. If you are generating a Gaussian distribution, then this is the standard deviation for each of x, y, and z.
- `double = obj.GetRadiusMaxValue ()` - Set the radius of the point cloud. If you are generating a Gaussian distribution, then this is the standard deviation for each of x, y, and z.
- `double = obj.GetRadius ()` - Set the radius of the point cloud. If you are generating a Gaussian distribution, then this is the standard deviation for each of x, y, and z.
- `obj.SetDistribution (int )` - Specify the distribution to use. The default is a uniform distribution. The shell distribution produces random points on the surface of the sphere, none in the interior.
- `obj.SetDistributionToUniform ()` - Specify the distribution to use. The default is a uniform distribution. The shell distribution produces random points on the surface of the sphere, none in the interior.
- `obj.SetDistributionToShell ()` - Specify the distribution to use. The default is a uniform distribution. The shell distribution produces random points on the surface of the sphere, none in the interior.
- `int = obj.GetDistribution ()` - Specify the distribution to use. The default is a uniform distribution. The shell distribution produces random points on the surface of the sphere, none in the interior.

33.163 vtkPolyDataConnectivityFilter

33.163.1 Usage

vtkPolyDataConnectivityFilter is a filter that extracts cells that share common points and/or satisfy a scalar threshold criterion. (Such a group of cells is called a region.) The filter works in one of six ways: 1) extract the largest connected region in the dataset; 2) extract specified region numbers; 3) extract all regions sharing specified point ids; 4) extract all regions sharing specified cell ids; 5) extract the region closest to the specified point; or 6) extract all regions (used to color regions).

This filter is specialized for polygonal data. This means it runs a bit faster and is easier to construct visualization networks that process polygonal data.

The behavior of vtkPolyDataConnectivityFilter can be modified by turning on the boolean ivar ScalarConnectivity. If this flag is on, the connectivity algorithm is modified so that cells are considered connected only if 1) they are geometrically connected (share a point) and 2) the scalar values of one of the cell's points falls in the scalar range specified. This use of ScalarConnectivity is particularly useful for selecting cells for later processing.

To create an instance of class vtkPolyDataConnectivityFilter, simply invoke its constructor as follows

```
obj = vtkPolyDataConnectivityFilter
```

33.163.2 Methods

The class vtkPolyDataConnectivityFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataConnectivityFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataConnectivityFilter = obj.NewInstance ()`
- `vtkPolyDataConnectivityFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetScalarConnectivity (int )` - Turn on/off connectivity based on scalar value. If on, cells are connected only if they share points AND one of the cells scalar values falls in the scalar range specified.
- `int = obj.GetScalarConnectivity ()` - Turn on/off connectivity based on scalar value. If on, cells are connected only if they share points AND one of the cells scalar values falls in the scalar range specified.
- `obj.ScalarConnectivityOn ()` - Turn on/off connectivity based on scalar value. If on, cells are connected only if they share points AND one of the cells scalar values falls in the scalar range specified.
- `obj.ScalarConnectivityOff ()` - Turn on/off connectivity based on scalar value. If on, cells are connected only if they share points AND one of the cells scalar values falls in the scalar range specified.
- `obj.SetScalarRange (double , double )` - Set the scalar range to use to extract cells based on scalar connectivity.
- `obj.SetScalarRange (double  a[2])` - Set the scalar range to use to extract cells based on scalar connectivity.
- `double = obj. GetScalarRange ()` - Set the scalar range to use to extract cells based on scalar connectivity.
- `obj.SetExtractionMode (int )` - Control the extraction of connected surfaces.
- `int = obj.GetExtractionModeMinValue ()` - Control the extraction of connected surfaces.

- `int = obj.GetExtractionModeMaxValue ()` - Control the extraction of connected surfaces.
- `int = obj.GetExtractionMode ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToPointSeededRegions ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToCellSeededRegions ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToLargestRegion ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToSpecifiedRegions ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToClosestPointRegion ()` - Control the extraction of connected surfaces.
- `obj.SetExtractionModeToAllRegions ()` - Control the extraction of connected surfaces.
- `string = obj.GetExtractionModeAsString ()` - Control the extraction of connected surfaces.
- `obj.InitializeSeedList ()` - Initialize list of point ids/cell ids used to seed regions.
- `obj.AddSeed (int id)` - Add a seed id (point or cell id). Note: ids are 0-offset.
- `obj.DeleteSeed (int id)` - Delete a seed id (point or cell id). Note: ids are 0-offset.
- `obj.InitializeSpecifiedRegionList ()` - Initialize list of region ids to extract.
- `obj.AddSpecifiedRegion (int id)` - Add a region id to extract. Note: ids are 0-offset.
- `obj.DeleteSpecifiedRegion (int id)` - Delete a region id to extract. Note: ids are 0-offset.
- `obj.SetClosestPoint (double , double , double )` - Use to specify x-y-z point coordinates when extracting the region closest to a specified point.
- `obj.SetClosestPoint (double a[3])` - Use to specify x-y-z point coordinates when extracting the region closest to a specified point.
- `double = obj. GetClosestPoint ()` - Use to specify x-y-z point coordinates when extracting the region closest to a specified point.
- `int = obj.GetNumberOfExtractedRegions ()` - Obtain the number of connected regions.
- `obj.SetColorRegions (int )` - Turn on/off the coloring of connected regions.
- `int = obj.GetColorRegions ()` - Turn on/off the coloring of connected regions.
- `obj.ColorRegionsOn ()` - Turn on/off the coloring of connected regions.
- `obj.ColorRegionsOff ()` - Turn on/off the coloring of connected regions.

33.164 vtkPolyDataNormals

33.164.1 Usage

vtkPolyDataNormals is a filter that computes point normals for a polygonal mesh. The filter can reorder polygons to insure consistent orientation across polygon neighbors. Sharp edges can be split and points duplicated with separate normals to give crisp (rendered) surface definition. It is also possible to globally flip the normal orientation.

The algorithm works by determining normals for each polygon and then averaging them at shared points. When sharp edges are present, the edges are split and new points generated to prevent blurry edges (due to Gouraud shading).

To create an instance of class vtkPolyDataNormals, simply invoke its constructor as follows

```
obj = vtkPolyDataNormals
```

33.164.2 Methods

The class vtkPolyDataNormals has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataNormals class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataNormals = obj.NewInstance ()`
- `vtkPolyDataNormals = obj.SafeDownCast (vtkObject o)`
- `obj.SetFeatureAngle (double )` - Specify the angle that defines a sharp edge. If the difference in angle across neighboring polygons is greater than this value, the shared edge is considered "sharp".
- `double = obj.GetFeatureAngleMinValue ()` - Specify the angle that defines a sharp edge. If the difference in angle across neighboring polygons is greater than this value, the shared edge is considered "sharp".
- `double = obj.GetFeatureAngleMaxValue ()` - Specify the angle that defines a sharp edge. If the difference in angle across neighboring polygons is greater than this value, the shared edge is considered "sharp".
- `double = obj.GetFeatureAngle ()` - Specify the angle that defines a sharp edge. If the difference in angle across neighboring polygons is greater than this value, the shared edge is considered "sharp".
- `obj.SetSplitting (int )` - Turn on/off the splitting of sharp edges.
- `int = obj.GetSplitting ()` - Turn on/off the splitting of sharp edges.
- `obj.SplittingOn ()` - Turn on/off the splitting of sharp edges.
- `obj.SplittingOff ()` - Turn on/off the splitting of sharp edges.
- `obj.SetConsistency (int )` - Turn on/off the enforcement of consistent polygon ordering.
- `int = obj.GetConsistency ()` - Turn on/off the enforcement of consistent polygon ordering.
- `obj.ConsistencyOn ()` - Turn on/off the enforcement of consistent polygon ordering.
- `obj.ConsistencyOff ()` - Turn on/off the enforcement of consistent polygon ordering.
- `obj.SetAutoOrientNormals (int )` - Turn on/off the automatic determination of correct normal orientation. NOTE: This assumes a completely closed surface (i.e. no boundary edges) and no non-manifold edges. If these constraints do not hold, all bets are off. This option adds some computational complexity, and is useful if you don't want to have to inspect the rendered image to determine whether to turn on the FlipNormals flag. However, this flag can work with the FlipNormals flag, and if both are set, all the normals in the output will point "inward".
- `int = obj.GetAutoOrientNormals ()` - Turn on/off the automatic determination of correct normal orientation. NOTE: This assumes a completely closed surface (i.e. no boundary edges) and no non-manifold edges. If these constraints do not hold, all bets are off. This option adds some computational complexity, and is useful if you don't want to have to inspect the rendered image to determine whether to turn on the FlipNormals flag. However, this flag can work with the FlipNormals flag, and if both are set, all the normals in the output will point "inward".

- `obj.AutoOrientNormalsOn ()` - Turn on/off the automatic determination of correct normal orientation. NOTE: This assumes a completely closed surface (i.e. no boundary edges) and no non-manifold edges. If these constraints do not hold, all bets are off. This option adds some computational complexity, and is useful if you don't want to have to inspect the rendered image to determine whether to turn on the FlipNormals flag. However, this flag can work with the FlipNormals flag, and if both are set, all the normals in the output will point "inward".

- `obj.AutoOrientNormalsOff ()` - Turn on/off the automatic determination of correct normal orientation. NOTE: This assumes a completely closed surface (i.e. no boundary edges) and no non-manifold edges. If these constraints do not hold, all bets are off. This option adds some computational complexity, and is useful if you don't want to have to inspect the rendered image to determine whether to turn on the FlipNormals flag. However, this flag can work with the FlipNormals flag, and if both are set, all the normals in the output will point "inward".

- `obj.SetComputePointNormals (int )` - Turn on/off the computation of point normals.

- `int = obj.GetComputePointNormals ()` - Turn on/off the computation of point normals.

- `obj.ComputePointNormalsOn ()` - Turn on/off the computation of point normals.

- `obj.ComputePointNormalsOff ()` - Turn on/off the computation of point normals.

- `obj.SetComputeCellNormals (int )` - Turn on/off the computation of cell normals.

- `int = obj.GetComputeCellNormals ()` - Turn on/off the computation of cell normals.

- `obj.ComputeCellNormalsOn ()` - Turn on/off the computation of cell normals.

- `obj.ComputeCellNormalsOff ()` - Turn on/off the computation of cell normals.

- `obj.SetFlipNormals (int )` - Turn on/off the global flipping of normal orientation. Flipping reverves the meaning of front and back for Frontface and Backface culling in vtkProperty. Flipping modifies both the normal direction and the order of a cell's points.

- `int = obj.GetFlipNormals ()` - Turn on/off the global flipping of normal orientation. Flipping reverves the meaning of front and back for Frontface and Backface culling in vtkProperty. Flipping modifies both the normal direction and the order of a cell's points.

- `obj.FlipNormalsOn ()` - Turn on/off the global flipping of normal orientation. Flipping reverves the meaning of front and back for Frontface and Backface culling in vtkProperty. Flipping modifies both the normal direction and the order of a cell's points.

- `obj.FlipNormalsOff ()` - Turn on/off the global flipping of normal orientation. Flipping reverves the meaning of front and back for Frontface and Backface culling in vtkProperty. Flipping modifies both the normal direction and the order of a cell's points.

- `obj.SetNonManifoldTraversal (int )` - Turn on/off traversal across non-manifold edges. This will prevent problems where the consistency of polygonal ordering is corrupted due to topological loops.

- `int = obj.GetNonManifoldTraversal ()` - Turn on/off traversal across non-manifold edges. This will prevent problems where the consistency of polygonal ordering is corrupted due to topological loops.

- `obj.NonManifoldTraversalOn ()` - Turn on/off traversal across non-manifold edges. This will prevent problems where the consistency of polygonal ordering is corrupted due to topological loops.

- `obj.NonManifoldTraversalOff ()` - Turn on/off traversal across non-manifold edges. This will prevent problems where the consistency of polygonal ordering is corrupted due to topological loops.

33.165 vtkPolyDataPointSampler

33.165.1 Usage

vtkPolyDataPointSampler generates points from input vtkPolyData. The points are placed approximately a specified distance apart.

This filter functions as follows. First, it regurgitates all input points, then samples all lines, plus edges associated with the input polygons and triangle strips to produce edge points. Finally, the interiors of polygons and triangle strips are subsampled to produce points. All of these functiona can be enabled or disabled separately. Note that this algorithm only approximately generates points the specified distance apart. Generally the point density is finer than requested.

To create an instance of class vtkPolyDataPointSampler, simply invoke its constructor as follows

```
obj = vtkPolyDataPointSampler
```

33.165.2 Methods

The class vtkPolyDataPointSampler has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataPointSampler class.

- `string = obj.GetClassName ()` - Standard macros for type information and printing.
- `int = obj.IsA (string name)` - Standard macros for type information and printing.
- `vtkPolyDataPointSampler = obj.NewInstance ()` - Standard macros for type information and printing.
- `vtkPolyDataPointSampler = obj.SafeDownCast (vtkObject o)` - Standard macros for type information and printing.
- `obj.SetDistance (double )` - Set/Get the approximate distance between points. This is an absolute distance measure. The default is 0.01.
- `double = obj.GetDistanceMinValue ()` - Set/Get the approximate distance between points. This is an absolute distance measure. The default is 0.01.
- `double = obj.GetDistanceMaxValue ()` - Set/Get the approximate distance between points. This is an absolute distance measure. The default is 0.01.
- `double = obj.GetDistance ()` - Set/Get the approximate distance between points. This is an absolute distance measure. The default is 0.01.
- `int = obj.GetGenerateVertexPoints ()` - Specify/retrieve a boolean flag indicating whether cell vertex points should be output.
- `obj.SetGenerateVertexPoints (int )` - Specify/retrieve a boolean flag indicating whether cell vertex points should be output.
- `obj.GenerateVertexPointsOn ()` - Specify/retrieve a boolean flag indicating whether cell vertex points should be output.
- `obj.GenerateVertexPointsOff ()` - Specify/retrieve a boolean flag indicating whether cell vertex points should be output.
- `int = obj.GetGenerateEdgePoints ()` - Specify/retrieve a boolean flag indicating whether cell edges should be sampled to produce output points. The default is true.

- `obj.SetGenerateEdgePoints (int )` - Specify/retrieve a boolean flag indicating whether cell edges should be sampled to produce output points. The default is true.
- `obj.GenerateEdgePointsOn ()` - Specify/retrieve a boolean flag indicating whether cell edges should be sampled to produce output points. The default is true.
- `obj.GenerateEdgePointsOff ()` - Specify/retrieve a boolean flag indicating whether cell edges should be sampled to produce output points. The default is true.
- `int = obj.GetGenerateInteriorPoints ()` - Specify/retrieve a boolean flag indicating whether cell interiors should be sampled to produce output points. The default is true.
- `obj.SetGenerateInteriorPoints (int )` - Specify/retrieve a boolean flag indicating whether cell interiors should be sampled to produce output points. The default is true.
- `obj.GenerateInteriorPointsOn ()` - Specify/retrieve a boolean flag indicating whether cell interiors should be sampled to produce output points. The default is true.
- `obj.GenerateInteriorPointsOff ()` - Specify/retrieve a boolean flag indicating whether cell interiors should be sampled to produce output points. The default is true.
- `int = obj.GetGenerateVertices ()` - Specify/retrieve a boolean flag indicating whether cell vertices should be generated. Cell vertices are useful if you actually want to display the points (that is, for each point generated, a vertex is generated). Recall that VTK only renders vertices and not points. The default is true.
- `obj.SetGenerateVertices (int )` - Specify/retrieve a boolean flag indicating whether cell vertices should be generated. Cell vertices are useful if you actually want to display the points (that is, for each point generated, a vertex is generated). Recall that VTK only renders vertices and not points. The default is true.
- `obj.GenerateVerticesOn ()` - Specify/retrieve a boolean flag indicating whether cell vertices should be generated. Cell vertices are useful if you actually want to display the points (that is, for each point generated, a vertex is generated). Recall that VTK only renders vertices and not points. The default is true.
- `obj.GenerateVerticesOff ()` - Specify/retrieve a boolean flag indicating whether cell vertices should be generated. Cell vertices are useful if you actually want to display the points (that is, for each point generated, a vertex is generated). Recall that VTK only renders vertices and not points. The default is true.

33.166 vtkPolyDataStreamer

33.166.1 Usage

vtkPolyDataStreamer initiates streaming by requesting pieces from its single input it appends these pieces it to the requested output. Note that since vtkPolyDataStreamer uses an append filter, all the polygons generated have to be kept in memory before rendering. If these do not fit in the memory, it is possible to make the vtkPolyDataMapper stream. Since the mapper will render each piece separately, all the polygons do not have to stored in memory. .SECTION Note The output may be slightly different if the pipeline does not handle ghost cells properly (i.e. you might see seames between the pieces).

To create an instance of class vtkPolyDataStreamer, simply invoke its constructor as follows

```
obj = vtkPolyDataStreamer
```

33.166.2 Methods

The class vtkPolyDataStreamer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataStreamer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataStreamer = obj.NewInstance ()`
- `vtkPolyDataStreamer = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfStreamDivisions (int num)` - Set the number of pieces to divide the problem into.
- `int = obj.GetNumberOfStreamDivisions ()` - Set the number of pieces to divide the problem into.
- `obj.SetColorByPiece (int )` - By default, this option is off. When it is on, cell scalars are generated based on which piece they are in.
- `int = obj.GetColorByPiece ()` - By default, this option is off. When it is on, cell scalars are generated based on which piece they are in.
- `obj.ColorByPieceOn ()` - By default, this option is off. When it is on, cell scalars are generated based on which piece they are in.
- `obj.ColorByPieceOff ()` - By default, this option is off. When it is on, cell scalars are generated based on which piece they are in.

33.167 vtkProbeFilter

33.167.1 Usage

vtkProbeFilter is a filter that computes point attributes (e.g., scalars, vectors, etc.) at specified point positions. The filter has two inputs: the Input and Source. The Input geometric structure is passed through the filter. The point attributes are computed at the Input point positions by interpolating into the source data. For example, we can compute data values on a plane (plane specified as Input) from a volume (Source). The cell data of the source data is copied to the output based on in which source cell each input point is. If an array of the same name exists both in source's point and cell data, only the one from the point data is probed.

This filter can be used to resample data, or convert one dataset form into another. For example, an unstructured grid (vtkUnstructuredGrid) can be probed with a volume (three-dimensional vtkImageData), and then volume rendering techniques can be used to visualize the results. Another example: a line or curve can be used to probe data to produce x-y plots along that line or curve.

To create an instance of class vtkProbeFilter, simply invoke its constructor as follows

```
obj = vtkProbeFilter
```

33.167.2 Methods

The class vtkProbeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProbeFilter class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkProbeFilter = obj.NewInstance ()`
- `vtkProbeFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetSource (vtkDataObject source)` - Specify the point locations used to probe input. Any geometry can be used. Old style. Do not use unless for backwards compatibility.
- `vtkDataObject = obj.GetSource ()` - Specify the point locations used to probe input. Any geometry can be used. Old style. Do not use unless for backwards compatibility.
- `obj.SetSourceConnection (vtkAlgorithmOutput algOutput)` - Specify the point locations used to probe input. Any geometry can be used. New style. Equivalent to SetInputConnection(1, algOutput).
- `obj.SetSpatialMatch (int )` - This flag is used only when a piece is requested to update. By default the flag is off. Because no spatial correspondence between input pieces and source pieces is known, all of the source has to be requested no matter what piece of the output is requested. When there is a spatial correspondence, the user/application can set this flag. This hint allows the breakup of the probe operation to be much more efficient. When piece m of n is requested for update by the user, then only n of m needs to be requested of the source.
- `int = obj.GetSpatialMatch ()` - This flag is used only when a piece is requested to update. By default the flag is off. Because no spatial correspondence between input pieces and source pieces is known, all of the source has to be requested no matter what piece of the output is requested. When there is a spatial correspondence, the user/application can set this flag. This hint allows the breakup of the probe operation to be much more efficient. When piece m of n is requested for update by the user, then only n of m needs to be requested of the source.
- `obj.SpatialMatchOn ()` - This flag is used only when a piece is requested to update. By default the flag is off. Because no spatial correspondence between input pieces and source pieces is known, all of the source has to be requested no matter what piece of the output is requested. When there is a spatial correspondence, the user/application can set this flag. This hint allows the breakup of the probe operation to be much more efficient. When piece m of n is requested for update by the user, then only n of m needs to be requested of the source.
- `obj.SpatialMatchOff ()` - This flag is used only when a piece is requested to update. By default the flag is off. Because no spatial correspondence between input pieces and source pieces is known, all of the source has to be requested no matter what piece of the output is requested. When there is a spatial correspondence, the user/application can set this flag. This hint allows the breakup of the probe operation to be much more efficient. When piece m of n is requested for update by the user, then only n of m needs to be requested of the source.
- `vtkIdTypeArray = obj.GetValidPoints ()` - Get the list of point ids in the output that contain attribute data interpolated from the source.
- `obj.SetValidPointMaskArrayName (string )` - Returns the name of the char array added to the output with values 1 for valid points and 0 for invalid points. Set to "vtkValidPointMask" by default.
- `string = obj.GetValidPointMaskArrayName ()` - Returns the name of the char array added to the output with values 1 for valid points and 0 for invalid points. Set to "vtkValidPointMask" by default.

33.168 vtkProbeSelectedLocations

33.168.1 Usage

vtkProbeSelectedLocations is similar to vtkExtractSelectedLocations except that it interpolates the point attributes at the probe location. This is equivalent to the vtkProbeFilter except that the probe locations

are provided by a vtkSelection. The FieldType of the input vtkSelection is immaterial and is ignored. The ContentType of the input vtkSelection must be vtkSelection::LOCATIONS.

To create an instance of class vtkProbeSelectedLocations, simply invoke its constructor as follows

```
obj = vtkProbeSelectedLocations
```

33.168.2 Methods

The class vtkProbeSelectedLocations has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProbeSelectedLocations class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProbeSelectedLocations = obj.NewInstance ()`
- `vtkProbeSelectedLocations = obj.SafeDownCast (vtkObject o)`

33.169 vtkProgrammableAttributeDataFilter

33.169.1 Usage

vtkProgrammableAttributeDataFilter is a filter that allows you to write a custom procedure to manipulate attribute data - either point or cell data. For example, you could generate scalars based on a complex formula; convert vectors to normals; compute scalar values as a function of vectors, texture coords, and/or any other point data attribute; and so on. The filter takes multiple inputs (input plus an auxiliary input list), so you can write procedures that combine several dataset point attributes. Note that the output of the filter is the same type (topology/geometry) as the input.

The filter works as follows. It operates like any other filter (i.e., checking and managing modified and execution times, processing Update() and Execute() methods, managing release of data, etc.), but the difference is that the Execute() method simply invokes a user-specified function with an optional (void *) argument (typically the "this" pointer in C++). It is also possible to specify a function to delete the argument via ExecuteMethodArgDelete().

To use the filter, you write a procedure to process the input datasets, process the data, and generate output data. Typically, this means grabbing the input point or cell data (using GetInput() and maybe GetInputList()), operating on it (creating new point and cell attributes such as scalars, vectors, etc.), and then setting the point and/or cell attributes in the output dataset (you'll need to use GetOutput() to access the output). (Note: besides C++, it is possible to do the same thing in Tcl, Java, or other languages that wrap the C++ core.) Remember, proper filter protocol requires that you don't modify the input data - you create new output data from the input.

To create an instance of class vtkProgrammableAttributeDataFilter, simply invoke its constructor as follows

```
obj = vtkProgrammableAttributeDataFilter
```

33.169.2 Methods

The class vtkProgrammableAttributeDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProgrammableAttributeDataFilter class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkProgrammableAttributeDataFilter = obj.NewInstance ()`
- `vtkProgrammableAttributeDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.AddInput (vtkDataSet in)` - Add a dataset to the list of data to process.
- `obj.RemoveInput (vtkDataSet in)` - Remove a dataset from the list of data to process.
- `vtkDataSetCollection = obj.GetInputList ()` - Return the list of inputs.

33.170 vtkProgrammableDataObjectSource

33.170.1 Usage

vtkProgrammableDataObjectSource is a source object that is programmable by the user. The output of the filter is a data object (vtkDataObject) which represents data via an instance of field data. To use this object, you must specify a function that creates the output.

Example use of this filter includes reading tabular data and encoding it as vtkFieldData. You can then use filters like vtkDataObjectToDataSetFilter to convert the data object to a dataset and then visualize it. Another important use of this class is that it allows users of interpreters (e.g., Tcl or Java) the ability to write source objects without having to recompile C++ code or generate new libraries.

To create an instance of class vtkProgrammableDataObjectSource, simply invoke its constructor as follows

```
obj = vtkProgrammableDataObjectSource
```

33.170.2 Methods

The class vtkProgrammableDataObjectSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProgrammableDataObjectSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProgrammableDataObjectSource = obj.NewInstance ()`
- `vtkProgrammableDataObjectSource = obj.SafeDownCast (vtkObject o)`

33.171 vtkProgrammableFilter

33.171.1 Usage

vtkProgrammableFilter is a filter that can be programmed by the user. To use the filter you define a function that retrieves input of the correct type, creates data, and then manipulates the output of the filter. Using this filter avoids the need for subclassing - and the function can be defined in an interpreter wrapper language such as Tcl or Java.

The trickiest part of using this filter is that the input and output methods are unusual and cannot be compile-time type checked. Instead, as a user of this filter it is your responsibility to set and get the correct input and output types.

To create an instance of class vtkProgrammableFilter, simply invoke its constructor as follows

```
obj = vtkProgrammableFilter
```

33.171.2 Methods

The class vtkProgrammableFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProgrammableFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProgrammableFilter = obj.NewInstance ()`
- `vtkProgrammableFilter = obj.SafeDownCast (vtkObject o)`
- `vtkPolyData = obj.GetPolyDataInput ()` - Get the input as a concrete type. This method is typically used by the writer of the filter function to get the input as a particular type (i.e., it essentially does type casting). It is the users responsibility to know the correct type of the input data.
- `vtkStructuredPoints = obj.GetStructuredPointsInput ()` - Get the input as a concrete type.
- `vtkStructuredGrid = obj.GetStructuredGridInput ()` - Get the input as a concrete type.
- `vtkUnstructuredGrid = obj.GetUnstructuredGridInput ()` - Get the input as a concrete type.
- `vtkRectilinearGrid = obj.GetRectilinearGridInput ()` - Get the input as a concrete type.
- `vtkGraph = obj.GetGraphInput ()` - Get the input as a concrete type.
- `vtkTable = obj.GetTableInput ()` - Get the input as a concrete type.
- `obj.SetCopyArrays (bool )` - When CopyArrays is true, all arrays are copied to the output iff input and output are of the same type. False by default.
- `bool = obj.GetCopyArrays ()` - When CopyArrays is true, all arrays are copied to the output iff input and output are of the same type. False by default.
- `obj.CopyArraysOn ()` - When CopyArrays is true, all arrays are copied to the output iff input and output are of the same type. False by default.
- `obj.CopyArraysOff ()` - When CopyArrays is true, all arrays are copied to the output iff input and output are of the same type. False by default.

33.172 vtkProgrammableGlyphFilter

33.172.1 Usage

vtkProgrammableGlyphFilter is a filter that allows you to place a glyph at each input point in the dataset. In addition, the filter is programmable which means the user has control over the generation of the glyph. The glyphs can be controlled via the point data attributes (e.g., scalars, vectors, etc.) or any other information in the input dataset.

This is the way the filter works. You must define an input dataset which at a minimum contains points with associated attribute values. Also, the Source instance variable must be set which is of type vtkPolyData. Then, for each point in the input, the PointId is set to the current point id, and a user-defined function is called (i.e., GlyphMethod). In this method you can manipulate the Source data (including changing to a different Source object). After the GlyphMethod is called, vtkProgrammableGlyphFilter will invoke an Update() on its Source object, and then copy its data to the output of the vtkProgrammableGlyphFilter. Therefore the output of this filter is of type vtkPolyData.

Another option to this filter is the way you color the glyphs. You can use the scalar data from the input or the source. The instance variable ColorMode controls this behavior.

To create an instance of class vtkProgrammableGlyphFilter, simply invoke its constructor as follows

```
obj = vtkProgrammableGlyphFilter
```

33.172.2 Methods

The class vtkProgrammableGlyphFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProgrammableGlyphFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProgrammableGlyphFilter = obj.NewInstance ()`
- `vtkProgrammableGlyphFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetSource (vtkPolyData source)` - Set/Get the source to use for this glyph. Note: you can change the source during execution of this filter.
- `vtkPolyData = obj.GetSource ()` - Set/Get the source to use for this glyph. Note: you can change the source during execution of this filter.
- `vtkIdType = obj.GetPointId ()` - Get the current point id during processing. Value only valid during the Execute() method of this filter. (Meant to be called by the GlyphMethod().)
- `double = obj. GetPoint ()` - Get the current point coordinates during processing. Value only valid during the Execute() method of this filter. (Meant to be called by the GlyphMethod().)
- `vtkPointData = obj.GetPointData ()` - Get the set of point data attributes for the input. A convenience to the programmer to be used in the GlyphMethod(). Only valid during the Execute() method of this filter.
- `obj.SetColorMode (int )` - Either color by the input or source scalar data.
- `int = obj.GetColorMode ()` - Either color by the input or source scalar data.
- `obj.SetColorModeToColorByInput ()` - Either color by the input or source scalar data.
- `obj.SetColorModeToColorBySource ()` - Either color by the input or source scalar data.
- `string = obj.GetColorModeAsString ()` - Either color by the input or source scalar data.

33.173 vtkProgrammableSource

33.173.1 Usage

vtkProgrammableSource is a source object that is programmable by the user. To use this object, you must specify a function that creates the output. It is possible to generate an output dataset of any (concrete) type; it is up to the function to properly initialize and define the output. Typically, you use one of the methods to get a concrete output type (e.g., GetPolyDataOutput() or GetStructuredPointsOutput()), and then manipulate the output in the user-specified function.

Example use of this include writing a function to read a data file or interface to another system. (You might want to do this in favor of deriving a new class.) Another important use of this class is that it allows users of interpreters (e.g., Tcl or Java) the ability to write source objects without having to recompile C++ code or generate new libraries.

To create an instance of class vtkProgrammableSource, simply invoke its constructor as follows

```
obj = vtkProgrammableSource
```

33.173.2 Methods

The class vtkProgrammableSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProgrammableSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProgrammableSource = obj.NewInstance ()`
- `vtkProgrammableSource = obj.SafeDownCast (vtkObject o)`
- `vtkPolyData = obj.GetPolyDataOutput ()` - Get the output as a concrete type. This method is typically used by the writer of the source function to get the output as a particular type (i.e., it essentially does type casting). It is the users responsibility to know the correct type of the output data.
- `vtkStructuredPoints = obj.GetStructuredPointsOutput ()` - Get the output as a concrete type.
- `vtkStructuredGrid = obj.GetStructuredGridOutput ()` - Get the output as a concrete type.
- `vtkUnstructuredGrid = obj.GetUnstructuredGridOutput ()` - Get the output as a concrete type.
- `vtkRectilinearGrid = obj.GetRectilinearGridOutput ()` - Get the output as a concrete type.

33.174 vtkProjectedTexture

33.174.1 Usage

vtkProjectedTexture assigns texture coordinates to a dataset as if the texture was projected from a slide projected located somewhere in the scene. Methods are provided to position the projector and aim it at a location, to set the width of the projector's frustum, and to set the range of texture coordinates assigned to the dataset.

Objects in the scene that appear behind the projector are also assigned texture coordinates; the projected image is left-right and top-bottom flipped, much as a lens' focus flips the rays of light that pass through it. A warning is issued if a point in the dataset falls at the focus of the projector.

To create an instance of class vtkProjectedTexture, simply invoke its constructor as follows

```
obj = vtkProjectedTexture
```

33.174.2 Methods

The class vtkProjectedTexture has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProjectedTexture class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProjectedTexture = obj.NewInstance ()`
- `vtkProjectedTexture = obj.SafeDownCast (vtkObject o)`
- `obj.SetPosition (double , double , double )` - Set/Get the position of the focus of the projector.

- `obj.SetPosition (double  a[3])` - Set/Get the position of the focus of the projector.
- `double = obj. GetPosition ()` - Set/Get the position of the focus of the projector.
- `obj.SetFocalPoint (double focalPoint[3])` - Set/Get the focal point of the projector (a point that lies along the center axis of the projector's frustum).
- `obj.SetFocalPoint (double x, double y, double z)` - Set/Get the focal point of the projector (a point that lies along the center axis of the projector's frustum).
- `double = obj. GetFocalPoint ()` - Set/Get the focal point of the projector (a point that lies along the center axis of the projector's frustum).
- `obj.SetCameraMode (int )` - Set/Get the camera mode of the projection – pinhole projection or two mirror projection.
- `int = obj.GetCameraMode ()` - Set/Get the camera mode of the projection – pinhole projection or two mirror projection.
- `obj.SetCameraModeToPinhole ()` - Set/Get the camera mode of the projection – pinhole projection or two mirror projection.
- `obj.SetCameraModeToTwoMirror ()` - Set/Get the mirror separation for the two mirror system.
- `obj.SetMirrorSeparation (double )` - Set/Get the mirror separation for the two mirror system.
- `double = obj.GetMirrorSeparation ()` - Set/Get the mirror separation for the two mirror system.
- `double = obj. GetOrientation ()` - Get the normalized orientation vector of the projector.
- `obj.SetUp (double , double , double )`
- `obj.SetUp (double  a[3])`
- `double = obj. GetUp ()`
- `obj.SetAspectRatio (double , double , double )`
- `obj.SetAspectRatio (double  a[3])`
- `double = obj. GetAspectRatio ()`
- `obj.SetSRange (double , double )` - Specify s-coordinate range for texture s-t coordinate pair.
- `obj.SetSRange (double  a[2])` - Specify s-coordinate range for texture s-t coordinate pair.
- `double = obj. GetSRange ()` - Specify s-coordinate range for texture s-t coordinate pair.
- `obj.SetTRange (double , double )` - Specify t-coordinate range for texture s-t coordinate pair.
- `obj.SetTRange (double  a[2])` - Specify t-coordinate range for texture s-t coordinate pair.
- `double = obj. GetTRange ()` - Specify t-coordinate range for texture s-t coordinate pair.

33.175 vtkQuadraturePointInterpolator

33.175.1 Usage

Interpolates each scalar/vector field in a vtkUnstrcturedGrid on its input to a specific set of quadrature points. The set of quadrature points is specified per array via a dictionary (ie an instance of vtkInformationQuadratureSchemeDefinitionVectorKey). contained in the array. The interpolated fields are placed in FieldData along with a set of per cell indexes, that allow random access to a given cells quadrature points.

To create an instance of class vtkQuadraturePointInterpolator, simply invoke its constructor as follows

```
obj = vtkQuadraturePointInterpolator
```

33.175.2 Methods

The class vtkQuadraturePointInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraturePointInterpolator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraturePointInterpolator = obj.NewInstance ()`
- `vtkQuadraturePointInterpolator = obj.SafeDownCast (vtkObject o)`

33.176 vtkQuadraturePointsGenerator

33.176.1 Usage

Create a vtkPolyData on its output containing the vertices for the quadrature points for one of the vtkDataArrays present on its input vtkUnstructuredGrid. If the input data set has has FieldData generated by vtkQuadraturePointInterpolator then this will be set as point data. Note: Point sets are generated per field array. This is because each field array may contain its own dictionary.

.SECTION See also vtkQuadraturePointInterpolator, vtkQuadratureSchemeDefinition, vtkInformationQuadratureSchemeDefinitionVectorKey

To create an instance of class vtkQuadraturePointsGenerator, simply invoke its constructor as follows

```
obj = vtkQuadraturePointsGenerator
```

33.176.2 Methods

The class vtkQuadraturePointsGenerator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadraturePointsGenerator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadraturePointsGenerator = obj.NewInstance ()`
- `vtkQuadraturePointsGenerator = obj.SafeDownCast (vtkObject o)`

33.177 vtkQuadratureSchemeDictionaryGenerator

33.177.1 Usage

Given an unstructured grid on its input this filter generates for each data array in point data dictionary (ie an instance of vtkInformationQuadratureSchemeDefinitionVectorKey). This filter has been introduced to facilitate testing of the vtkQuadrature* classes as these cannot operate with the dictionary. This class is for testing and should not be used for application development.

.SECTION See also vtkQuadraturePointInterpolator, vtkQuadraturePointsGenerator, vtkQuadratureSchemeDefinition

To create an instance of class vtkQuadratureSchemeDictionaryGenerator, simply invoke its constructor as follows

```
obj = vtkQuadratureSchemeDictionaryGenerator
```

33.177.2 Methods

The class vtkQuadratureSchemeDictionaryGenerator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadratureSchemeDictionaryGenerator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadratureSchemeDictionaryGenerator = obj.NewInstance ()`
- `vtkQuadratureSchemeDictionaryGenerator = obj.SafeDownCast (vtkObject o)`

33.178 vtkQuadricClustering

33.178.1 Usage

vtkQuadricClustering is a filter to reduce the number of triangles in a triangle mesh, forming a good approximation to the original geometry. The input to vtkQuadricClustering is a vtkPolyData object, and all types of polygonal data are handled.

The algorithm used is the one described by Peter Lindstrom in his Siggraph 2000 paper, "Out-of-Core Simplification of Large Polygonal Models." The general approach of the algorithm is to cluster vertices in a uniform binning of space, accumulating the quadric of each triangle (pushed out to the triangles vertices) within each bin, and then determining an optimal position for a single vertex in a bin by using the accumulated quadric. In more detail, the algorithm first gets the bounds of the input poly data. It then breaks this bounding volume into a user-specified number of spatial bins. It then reads each triangle from the input and hashes its vertices into these bins. (If this is the first time a bin has been visited, initialize its quadric to the 0 matrix.) The algorithm computes the error quadric for this triangle and adds it to the existing quadric of the bin in which each vertex is contained. Then, if 2 or more vertices of the triangle fall in the same bin, the triangle is dicarded. If the triangle is not discarded, it adds the triangle to the list of output triangles as a list of vertex identifiers. (There is one vertex id per bin.) After all the triangles have been read, the representative vertex for each bin is computed (an optimal location is found) using the quadric for that bin. This determines the spatial location of the vertices of each of the triangles in the output.

To use this filter, specify the divisions defining the spatial subdivision in the x, y, and z directions. You must also specify an input vtkPolyData. Then choose to either 1) use the original points that minimize the quadric error to produce the output triangles or 2) compute an optimal position in each bin to produce the output triangles (recommended and default behavior).

This filter can take multiple inputs. To do this, the user must explicity call StartAppend, Append (once for each input), and EndAppend. StartAppend sets up the data structure to hold the quadric matrices. Append processes each triangle in the input poly data it was called on, hashes its vertices to the appropriate bins, determines whether to keep this triangle, and updates the appropriate quadric matrices. EndAppend determines the spatial location of each of the representative vertices for the visited bins. While this approach does not fit into the visualization architecture and requires manual control, it has the advantage that extremely large data can be processed in pieces and appended to the filter piece-by-piece.

To create an instance of class vtkQuadricClustering, simply invoke its constructor as follows

```
obj = vtkQuadricClustering
```

33.178.2 Methods

The class vtkQuadricClustering has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadricClustering class.

- `string = obj.GetClassName ()` - Standard instantition, type and print methods.
- `int = obj.IsA (string name)` - Standard instantition, type and print methods.
- `vtkQuadricClustering = obj.NewInstance ()` - Standard instantition, type and print methods.
- `vtkQuadricClustering = obj.SafeDownCast (vtkObject o)` - Standard instantition, type and print methods.
- `obj.SetNumberOfXDivisions (int num)` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.
- `obj.SetNumberOfYDivisions (int num)` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.
- `obj.SetNumberOfZDivisions (int num)` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.
- `int = obj.GetNumberOfXDivisions ()` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.
- `int = obj.GetNumberOfYDivisions ()` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.
- `int = obj.GetNumberOfZDivisions ()` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.
- `obj.SetNumberOfDivisions (int div[3])` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.
- `obj.SetNumberOfDivisions (int div0, int div1, int div2)` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.
- `int = obj.GetNumberOfDivisions ()` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.
- `obj.GetNumberOfDivisions (int div[3])` - Set/Get the number of divisions along each axis for the spatial bins. The number of spatial bins is NumberOfXDivisions*NumberOfYDivisions* NumberOfZDivisions. The filter may choose to ignore large numbers of divisions if the input has few points and AutoAdjustNumberOfDivisions is enabled.

- `obj.SetAutoAdjustNumberOfDivisions (int )` - Enable automatic adjustment of number of divisions. If off, the number of divisions specified by the user is always used (as long as it is valid). The default is On

- `int = obj.GetAutoAdjustNumberOfDivisions ()` - Enable automatic adjustment of number of divisions. If off, the number of divisions specified by the user is always used (as long as it is valid). The default is On

- `obj.AutoAdjustNumberOfDivisionsOn ()` - Enable automatic adjustment of number of divisions. If off, the number of divisions specified by the user is always used (as long as it is valid). The default is On

- `obj.AutoAdjustNumberOfDivisionsOff ()` - Enable automatic adjustment of number of divisions. If off, the number of divisions specified by the user is always used (as long as it is valid). The default is On

- `obj.SetDivisionOrigin (double x, double y, double z)` - This is an alternative way to set up the bins. If you are trying to match boundaries between pieces, then you should use these methods rather than SetNumberOfDivisions. To use these methods, specify the origin and spacing of the spatial binning.

- `obj.SetDivisionOrigin (double o[3])` - This is an alternative way to set up the bins. If you are trying to match boundaries between pieces, then you should use these methods rather than SetNumberOfDivisions. To use these methods, specify the origin and spacing of the spatial binning.

- `double = obj. GetDivisionOrigin ()` - This is an alternative way to set up the bins. If you are trying to match boundaries between pieces, then you should use these methods rather than SetNumberOfDivisions. To use these methods, specify the origin and spacing of the spatial binning.

- `obj.SetDivisionSpacing (double x, double y, double z)` - This is an alternative way to set up the bins. If you are trying to match boundaries between pieces, then you should use these methods rather than SetNumberOfDivisions. To use these methods, specify the origin and spacing of the spatial binning.

- `obj.SetDivisionSpacing (double s[3])` - This is an alternative way to set up the bins. If you are trying to match boundaries between pieces, then you should use these methods rather than SetNumberOfDivisions. To use these methods, specify the origin and spacing of the spatial binning.

- `double = obj. GetDivisionSpacing ()` - This is an alternative way to set up the bins. If you are trying to match boundaries between pieces, then you should use these methods rather than SetNumberOfDivisions. To use these methods, specify the origin and spacing of the spatial binning.

- `obj.SetUseInputPoints (int )` - Normally the point that minimizes the quadric error function is used as the output of the bin. When this flag is on, the bin point is forced to be one of the points from the input (the one with the smallest error). This option does not work (i.e., input points cannot be used) when the append methods (StartAppend(), Append(), EndAppend()) are being called directly.

- `int = obj.GetUseInputPoints ()` - Normally the point that minimizes the quadric error function is used as the output of the bin. When this flag is on, the bin point is forced to be one of the points from the input (the one with the smallest error). This option does not work (i.e., input points cannot be used) when the append methods (StartAppend(), Append(), EndAppend()) are being called directly.

- `obj.UseInputPointsOn ()` - Normally the point that minimizes the quadric error function is used as the output of the bin. When this flag is on, the bin point is forced to be one of the points from the input (the one with the smallest error). This option does not work (i.e., input points cannot be used) when the append methods (StartAppend(), Append(), EndAppend()) are being called directly.

- `obj.UseInputPointsOff ()` - Normally the point that minimizes the quadric error function is used as the output of the bin. When this flag is on, the bin point is forced to be one of the points from the input (the one with the smallest error). This option does not work (i.e., input points cannot be used) when the append methods (StartAppend(), Append(), EndAppend()) are being called directly.

- `obj.SetUseFeatureEdges (int )` - By default, this flag is off. When "UseFeatureEdges" is on, then quadrics are computed for boundary edges/feature edges. They influence the quadrics (position of points), but not the mesh. Which features to use can be controlled by the filter "FeatureEdges".

- `int = obj.GetUseFeatureEdges ()` - By default, this flag is off. When "UseFeatureEdges" is on, then quadrics are computed for boundary edges/feature edges. They influence the quadrics (position of points), but not the mesh. Which features to use can be controlled by the filter "FeatureEdges".

- `obj.UseFeatureEdgesOn ()` - By default, this flag is off. When "UseFeatureEdges" is on, then quadrics are computed for boundary edges/feature edges. They influence the quadrics (position of points), but not the mesh. Which features to use can be controlled by the filter "FeatureEdges".

- `obj.UseFeatureEdgesOff ()` - By default, this flag is off. When "UseFeatureEdges" is on, then quadrics are computed for boundary edges/feature edges. They influence the quadrics (position of points), but not the mesh. Which features to use can be controlled by the filter "FeatureEdges".

- `vtkFeatureEdges = obj.GetFeatureEdges ()` - By default, this flag is off. It only has an effect when "UseFeatureEdges" is also on. When "UseFeaturePoints" is on, then quadrics are computed for boundary / feature points used in the boundary / feature edges. They influence the quadrics (position of points), but not the mesh.

- `obj.SetUseFeaturePoints (int )` - By default, this flag is off. It only has an effect when "UseFeatureEdges" is also on. When "UseFeaturePoints" is on, then quadrics are computed for boundary / feature points used in the boundary / feature edges. They influence the quadrics (position of points), but not the mesh.

- `int = obj.GetUseFeaturePoints ()` - By default, this flag is off. It only has an effect when "UseFeatureEdges" is also on. When "UseFeaturePoints" is on, then quadrics are computed for boundary / feature points used in the boundary / feature edges. They influence the quadrics (position of points), but not the mesh.

- `obj.UseFeaturePointsOn ()` - By default, this flag is off. It only has an effect when "UseFeatureEdges" is also on. When "UseFeaturePoints" is on, then quadrics are computed for boundary / feature points used in the boundary / feature edges. They influence the quadrics (position of points), but not the mesh.

- `obj.UseFeaturePointsOff ()` - By default, this flag is off. It only has an effect when "UseFeatureEdges" is also on. When "UseFeaturePoints" is on, then quadrics are computed for boundary / feature points used in the boundary / feature edges. They influence the quadrics (position of points), but not the mesh.

- `obj.SetFeaturePointsAngle (double )` - Set/Get the angle to use in determining whether a point on a boundary / feature edge is a feature point.

- `double = obj.GetFeaturePointsAngleMinValue ()` - Set/Get the angle to use in determining whether a point on a boundary / feature edge is a feature point.

- `double = obj.GetFeaturePointsAngleMaxValue ()` - Set/Get the angle to use in determining whether a point on a boundary / feature edge is a feature point.

- `double = obj.GetFeaturePointsAngle ()` - Set/Get the angle to use in determining whether a point on a boundary / feature edge is a feature point.

- `obj.SetUseInternalTriangles (int )` - When this flag is on (and it is on by default), then triangles that are completely contained in a bin are added to the bin quadrics. When the the flag is off the filter operates faster, but the surface may not be as well behaved.
- `int = obj.GetUseInternalTriangles ()` - When this flag is on (and it is on by default), then triangles that are completely contained in a bin are added to the bin quadrics. When the the flag is off the filter operates faster, but the surface may not be as well behaved.
- `obj.UseInternalTrianglesOn ()` - When this flag is on (and it is on by default), then triangles that are completely contained in a bin are added to the bin quadrics. When the the flag is off the filter operates faster, but the surface may not be as well behaved.
- `obj.UseInternalTrianglesOff ()` - When this flag is on (and it is on by default), then triangles that are completely contained in a bin are added to the bin quadrics. When the the flag is off the filter operates faster, but the surface may not be as well behaved.
- `obj.StartAppend (double bounds)` - These methods provide an alternative way of executing the filter. PolyData can be added to the result in pieces (append). In this mode, the user must specify the bounds of the entire model as an argument to the "StartAppend" method.
- `obj.StartAppend (double x0, double x1, double y0, double y1, double z0, double z1)` - These methods provide an alternative way of executing the filter. PolyData can be added to the result in pieces (append). In this mode, the user must specify the bounds of the entire model as an argument to the "StartAppend" method.
- `obj.Append (vtkPolyData piece)` - These methods provide an alternative way of executing the filter. PolyData can be added to the result in pieces (append). In this mode, the user must specify the bounds of the entire model as an argument to the "StartAppend" method.
- `obj.EndAppend ()` - These methods provide an alternative way of executing the filter. PolyData can be added to the result in pieces (append). In this mode, the user must specify the bounds of the entire model as an argument to the "StartAppend" method.
- `obj.SetCopyCellData (int )` - This flag makes the filter copy cell data from input to output (the best it can). It uses input cells that trigger the addition of output cells (no averaging). This is off by default, and does not work when append is being called explicitely (non-pipeline usage).
- `int = obj.GetCopyCellData ()` - This flag makes the filter copy cell data from input to output (the best it can). It uses input cells that trigger the addition of output cells (no averaging). This is off by default, and does not work when append is being called explicitely (non-pipeline usage).
- `obj.CopyCellDataOn ()` - This flag makes the filter copy cell data from input to output (the best it can). It uses input cells that trigger the addition of output cells (no averaging). This is off by default, and does not work when append is being called explicitely (non-pipeline usage).
- `obj.CopyCellDataOff ()` - This flag makes the filter copy cell data from input to output (the best it can). It uses input cells that trigger the addition of output cells (no averaging). This is off by default, and does not work when append is being called explicitely (non-pipeline usage).
- `obj.SetPreventDuplicateCells (int )` - Specify a boolean indicating whether to remove duplicate cells (i.e. triangles). This is a little slower, and takes more memory, but in some cases can reduce the number of cells produced by an order of magnitude. By default, this flag is true.
- `int = obj.GetPreventDuplicateCells ()` - Specify a boolean indicating whether to remove duplicate cells (i.e. triangles). This is a little slower, and takes more memory, but in some cases can reduce the number of cells produced by an order of magnitude. By default, this flag is true.
- `obj.PreventDuplicateCellsOn ()` - Specify a boolean indicating whether to remove duplicate cells (i.e. triangles). This is a little slower, and takes more memory, but in some cases can reduce the number of cells produced by an order of magnitude. By default, this flag is true.

- `obj.PreventDuplicateCellsOff ()` - Specify a boolean indicating whether to remove duplicate cells (i.e. triangles). This is a little slower, and takes more memory, but in some cases can reduce the number of cells produced by an order of magnitude. By default, this flag is true.

33.179 vtkQuadricDecimation

33.179.1 Usage

vtkQuadricDecimation is a filter to reduce the number of triangles in a triangle mesh, forming a good approximation to the original geometry. The input to vtkQuadricDecimation is a vtkPolyData object, and only triangles are treated. If you desire to decimate polygonal meshes, first triangulate the polygons with vtkTriangleFilter.

The algorithm is based on repeated edge collapses until the requested mesh reduction is achieved. Edges are placed in a priority queue based on the "cost" to delete the edge. The cost is an approximate measure of error (distance to the original surface)–described by the so-called quadric error measure. The quadric error measure is associated with each vertex of the mesh and represents a matrix of planes incident on that vertex. The distance of the planes to the vertex is the error in the position of the vertex (originally the vertex error iz zero). As edges are deleted, the quadric error measure associated with the two end points of the edge are summed (this combines the plane equations) and an optimal collapse point can be computed. Edges connected to the collapse point are then reinserted into the queue after computing the new cost to delete them. The process continues until the desired reduction level is reached or topological constraints prevent further reduction. Note that this basic algorithm can be extended to higher dimensions by taking into account variation in attributes (i.e., scalars, vectors, and so on).

This paper is based on the work of Garland and Heckbert who first presented the quadric error measure at Siggraph '97 "Surface Simplification Using Quadric Error Metrics". For details of the algorithm Michael Garland's Ph.D. thesis is also recommended. Hughues Hoppe's Vis '99 paper, "New Quadric Metric for Simplifying Meshes with Appearance Attributes" is also a good take on the subject especially as it pertains to the error metric applied to attributes.

.SECTION Thanks Thanks to Bradley Lowekamp of the National Library of Medicine/NIH for contributing this class.

To create an instance of class vtkQuadricDecimation, simply invoke its constructor as follows

```
obj = vtkQuadricDecimation
```

33.179.2 Methods

The class vtkQuadricDecimation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadricDecimation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuadricDecimation = obj.NewInstance ()`
- `vtkQuadricDecimation = obj.SafeDownCast (vtkObject o)`
- `obj.SetTargetReduction (double )` - Set/Get the desired reduction (expressed as a fraction of the original number of triangles). The actual reduction may be less depending on triangulation and topological constraints.
- `double = obj.GetTargetReductionMinValue ()` - Set/Get the desired reduction (expressed as a fraction of the original number of triangles). The actual reduction may be less depending on triangulation and topological constraints.

- `double = obj.GetTargetReductionMaxValue ()` - Set/Get the desired reduction (expressed as a fraction of the original number of triangles). The actual reduction may be less depending on triangulation and topological constraints.

- `double = obj.GetTargetReduction ()` - Set/Get the desired reduction (expressed as a fraction of the original number of triangles). The actual reduction may be less depending on triangulation and topological constraints.

- `obj.SetAttributeErrorMetric (int )` - Decide whether to include data attributes in the error metric. If off, then only geometric error is used to control the decimation. By default the attribute errors are off.

- `int = obj.GetAttributeErrorMetric ()` - Decide whether to include data attributes in the error metric. If off, then only geometric error is used to control the decimation. By default the attribute errors are off.

- `obj.AttributeErrorMetricOn ()` - Decide whether to include data attributes in the error metric. If off, then only geometric error is used to control the decimation. By default the attribute errors are off.

- `obj.AttributeErrorMetricOff ()` - Decide whether to include data attributes in the error metric. If off, then only geometric error is used to control the decimation. By default the attribute errors are off.

- `obj.SetScalarsAttribute (int )` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.

- `int = obj.GetScalarsAttribute ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.

- `obj.ScalarsAttributeOn ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.

- `obj.ScalarsAttributeOff ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.

- `obj.SetVectorsAttribute (int )` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.

- `int = obj.GetVectorsAttribute ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.

- `obj.VectorsAttributeOn ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.

- `obj.VectorsAttributeOff ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.

- `obj.SetNormalsAttribute (int )` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.

- `int = obj.GetNormalsAttribute ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `obj.NormalsAttributeOn ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `obj.NormalsAttributeOff ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `obj.SetTCoordsAttribute (int )` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `int = obj.GetTCoordsAttribute ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `obj.TCoordsAttributeOn ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `obj.TCoordsAttributeOff ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `obj.SetTensorsAttribute (int )` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `int = obj.GetTensorsAttribute ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `obj.TensorsAttributeOn ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `obj.TensorsAttributeOff ()` - If attribute errors are to be included in the metric (i.e., AttributeErrorMetric is on), then the following flags control which attributes are to be included in the error calculation. By default all of these are on.
- `obj.SetScalarsWeight (double )` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.
- `obj.SetVectorsWeight (double )` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.
- `obj.SetNormalsWeight (double )` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.
- `obj.SetTCoordsWeight (double )` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.
- `obj.SetTensorsWeight (double )` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.

- `double = obj.GetScalarsWeight ()` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.
- `double = obj.GetVectorsWeight ()` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.
- `double = obj.GetNormalsWeight ()` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.
- `double = obj.GetTCoordsWeight ()` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.
- `double = obj.GetTensorsWeight ()` - Set/Get the scaling weight contribution of the attribute. These values are used to weight the contribution of the attributes towards the error metric.
- `double = obj.GetActualReduction ()` - Get the actual reduction. This value is only valid after the filter has executed.

33.180 vtkQuantizePolyDataPoints

33.180.1 Usage

vtkQuantizePolyDataPoints is a subclass of vtkCleanPolyData and inherits the functionality of vtkCleanPolyData with the addition that it quantizes the point coordinates before inserting into the point list. The user should set QFactor to a positive value (0.25 by default) and all x,y,z coordinates will be quantized to that grain size.

A tolerance of zero is expected, though positive values may be used, the quantization will take place before the tolerance is applied.

To create an instance of class vtkQuantizePolyDataPoints, simply invoke its constructor as follows

```
obj = vtkQuantizePolyDataPoints
```

33.180.2 Methods

The class vtkQuantizePolyDataPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuantizePolyDataPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQuantizePolyDataPoints = obj.NewInstance ()`
- `vtkQuantizePolyDataPoints = obj.SafeDownCast (vtkObject o)`
- `obj.SetQFactor (double )` - Specify quantization grain size. Default is 0.25
- `double = obj.GetQFactorMinValue ()` - Specify quantization grain size. Default is 0.25
- `double = obj.GetQFactorMaxValue ()` - Specify quantization grain size. Default is 0.25
- `double = obj.GetQFactor ()` - Specify quantization grain size. Default is 0.25
- `obj.OperateOnPoint (double in[3], double out[3])` - Perform quantization on a point
- `obj.OperateOnBounds (double in[6], double out[6])` - Perform quantization on bounds

33.181 vtkRandomAttributeGenerator

33.181.1 Usage

vtkRandomAttributeGenerator is a filter that creates random attributes including scalars, vectors, normals, tensors, texture coordinates and/or general data arrays. These attributes can be generated as point data, cell data or general field data. The generation of each component is normalized between a user-specified minimum and maximum value.

This filter provides that capability to specify the data type of the attributes, the range for each of the components, and the number of components. Note, however, that this flexibility only goes so far because some attributes (e.g., normals, vectors and tensors) are fixed in the number of components, and in the case of normals and tensors, are constrained in the values that some of the components can take (i.e., normals have magnitude one, and tensors are symmetric).

To create an instance of class vtkRandomAttributeGenerator, simply invoke its constructor as follows

```
obj = vtkRandomAttributeGenerator
```

33.181.2 Methods

The class vtkRandomAttributeGenerator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRandomAttributeGenerator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRandomAttributeGenerator = obj.NewInstance ()`
- `vtkRandomAttributeGenerator = obj.SafeDownCast (vtkObject o)`
- `obj.SetDataType (int )` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.
- `obj.SetDataTypeToBit ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.
- `obj.SetDataTypeToChar ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.
- `obj.SetDataTypeToUnsignedChar ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.
- `obj.SetDataTypeToShort ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.
- `obj.SetDataTypeToUnsignedShort ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.
- `obj.SetDataTypeToInt ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.
- `obj.SetDataTypeToUnsignedInt ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.
- `obj.SetDataTypeToLong ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.

- `obj.SetDataTypeToUnsignedLong ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.

- `obj.SetDataTypeToFloat ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.

- `obj.SetDataTypeToDouble ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.

- `int = obj.GetDataType ()` - Specify the type of array to create (all components of this array are of this type). This holds true for all arrays that are created.

- `obj.SetNumberOfComponents (int )` - Specify the number of components to generate. This value only applies to those attribute types that take a variable number of components. For example, a vector is only three components so the number of components is not applicable; whereas a scalar may support multiple, varying number of components.

- `int = obj.GetNumberOfComponentsMinValue ()` - Specify the number of components to generate. This value only applies to those attribute types that take a variable number of components. For example, a vector is only three components so the number of components is not applicable; whereas a scalar may support multiple, varying number of components.

- `int = obj.GetNumberOfComponentsMaxValue ()` - Specify the number of components to generate. This value only applies to those attribute types that take a variable number of components. For example, a vector is only three components so the number of components is not applicable; whereas a scalar may support multiple, varying number of components.

- `int = obj.GetNumberOfComponents ()` - Specify the number of components to generate. This value only applies to those attribute types that take a variable number of components. For example, a vector is only three components so the number of components is not applicable; whereas a scalar may support multiple, varying number of components.

- `obj.SetMinimumComponentValue (double )` - Set the minimum component value. This applies to all data that is generated, although normals and tensors have internal constraints that must be observed.

- `double = obj.GetMinimumComponentValue ()` - Set the minimum component value. This applies to all data that is generated, although normals and tensors have internal constraints that must be observed.

- `obj.SetMaximumComponentValue (double )` - Set the maximum component value. This applies to all data that is generated, although normals and tensors have internal constraints that must be observed.

- `double = obj.GetMaximumComponentValue ()` - Set the maximum component value. This applies to all data that is generated, although normals and tensors have internal constraints that must be observed.

- `obj.SetNumberOfTuples (vtkIdType )` - Specify the number of tuples to generate. This value only applies when creating general field data. In all other cases (i.e., point data or cell data), the number of tuples is controlled by the number of points and cells, respectively.

- `vtkIdType = obj.GetNumberOfTuplesMinValue ()` - Specify the number of tuples to generate. This value only applies when creating general field data. In all other cases (i.e., point data or cell data), the number of tuples is controlled by the number of points and cells, respectively.

- `vtkIdType = obj.GetNumberOfTuplesMaxValue ()` - Specify the number of tuples to generate. This value only applies when creating general field data. In all other cases (i.e., point data or cell data), the number of tuples is controlled by the number of points and cells, respectively.

- `vtkIdType = obj.GetNumberOfTuples ()` - Specify the number of tuples to generate. This value only applies when creating general field data. In all other cases (i.e., point data or cell data), the number of tuples is controlled by the number of points and cells, respectively.
- `obj.SetGeneratePointScalars (int )` - Indicate that point scalars are to be generated. Note that the specified number of components is used to create the scalar.
- `int = obj.GetGeneratePointScalars ()` - Indicate that point scalars are to be generated. Note that the specified number of components is used to create the scalar.
- `obj.GeneratePointScalarsOn ()` - Indicate that point scalars are to be generated. Note that the specified number of components is used to create the scalar.
- `obj.GeneratePointScalarsOff ()` - Indicate that point scalars are to be generated. Note that the specified number of components is used to create the scalar.
- `obj.SetGeneratePointVectors (int )` - Indicate that point vectors are to be generated. Note that the number of components is always equal to three.
- `int = obj.GetGeneratePointVectors ()` - Indicate that point vectors are to be generated. Note that the number of components is always equal to three.
- `obj.GeneratePointVectorsOn ()` - Indicate that point vectors are to be generated. Note that the number of components is always equal to three.
- `obj.GeneratePointVectorsOff ()` - Indicate that point vectors are to be generated. Note that the number of components is always equal to three.
- `obj.SetGeneratePointNormals (int )` - Indicate that point normals are to be generated. Note that the number of components is always equal to three.
- `int = obj.GetGeneratePointNormals ()` - Indicate that point normals are to be generated. Note that the number of components is always equal to three.
- `obj.GeneratePointNormalsOn ()` - Indicate that point normals are to be generated. Note that the number of components is always equal to three.
- `obj.GeneratePointNormalsOff ()` - Indicate that point normals are to be generated. Note that the number of components is always equal to three.
- `obj.SetGeneratePointTensors (int )` - Indicate that point tensors are to be generated. Note that the number of components is always equal to nine.
- `int = obj.GetGeneratePointTensors ()` - Indicate that point tensors are to be generated. Note that the number of components is always equal to nine.
- `obj.GeneratePointTensorsOn ()` - Indicate that point tensors are to be generated. Note that the number of components is always equal to nine.
- `obj.GeneratePointTensorsOff ()` - Indicate that point tensors are to be generated. Note that the number of components is always equal to nine.
- `obj.SetGeneratePointTCoords (int )` - Indicate that point texture coordinates are to be generated. Note that the specified number of components is used to create the texture coordinates (but must range between one and three).
- `int = obj.GetGeneratePointTCoords ()` - Indicate that point texture coordinates are to be generated. Note that the specified number of components is used to create the texture coordinates (but must range between one and three).

- `obj.GeneratePointTCoordsOn ()` - Indicate that point texture coordinates are to be generated. Note that the specified number of components is used to create the texture coordinates (but must range between one and three).
- `obj.GeneratePointTCoordsOff ()` - Indicate that point texture coordinates are to be generated. Note that the specified number of components is used to create the texture coordinates (but must range between one and three).
- `obj.SetGeneratePointArray (int )` - Indicate that an arbitrary point array is to be generated. Note that the specified number of components is used to create the array.
- `int = obj.GetGeneratePointArray ()` - Indicate that an arbitrary point array is to be generated. Note that the specified number of components is used to create the array.
- `obj.GeneratePointArrayOn ()` - Indicate that an arbitrary point array is to be generated. Note that the specified number of components is used to create the array.
- `obj.GeneratePointArrayOff ()` - Indicate that an arbitrary point array is to be generated. Note that the specified number of components is used to create the array.
- `obj.SetGenerateCellScalars (int )` - Indicate that cell scalars are to be generated. Note that the specified number of components is used to create the scalar.
- `int = obj.GetGenerateCellScalars ()` - Indicate that cell scalars are to be generated. Note that the specified number of components is used to create the scalar.
- `obj.GenerateCellScalarsOn ()` - Indicate that cell scalars are to be generated. Note that the specified number of components is used to create the scalar.
- `obj.GenerateCellScalarsOff ()` - Indicate that cell scalars are to be generated. Note that the specified number of components is used to create the scalar.
- `obj.SetGenerateCellVectors (int )` - Indicate that cell vectors are to be generated. Note that the number of components is always equal to three.
- `int = obj.GetGenerateCellVectors ()` - Indicate that cell vectors are to be generated. Note that the number of components is always equal to three.
- `obj.GenerateCellVectorsOn ()` - Indicate that cell vectors are to be generated. Note that the number of components is always equal to three.
- `obj.GenerateCellVectorsOff ()` - Indicate that cell vectors are to be generated. Note that the number of components is always equal to three.
- `obj.SetGenerateCellNormals (int )` - Indicate that cell normals are to be generated. Note that the number of components is always equal to three.
- `int = obj.GetGenerateCellNormals ()` - Indicate that cell normals are to be generated. Note that the number of components is always equal to three.
- `obj.GenerateCellNormalsOn ()` - Indicate that cell normals are to be generated. Note that the number of components is always equal to three.
- `obj.GenerateCellNormalsOff ()` - Indicate that cell normals are to be generated. Note that the number of components is always equal to three.
- `obj.SetGenerateCellTensors (int )` - Indicate that cell tensors are to be generated. Note that the number of components is always equal to nine.
- `int = obj.GetGenerateCellTensors ()` - Indicate that cell tensors are to be generated. Note that the number of components is always equal to nine.

- `obj.GenerateCellTensorsOn ()` - Indicate that cell tensors are to be generated. Note that the number of components is always equal to nine.
- `obj.GenerateCellTensorsOff ()` - Indicate that cell tensors are to be generated. Note that the number of components is always equal to nine.
- `obj.SetGenerateCellTCoords (int )` - Indicate that cell texture coordinates are to be generated. Note that the specified number of components is used to create the texture coordinates (but must range between one and three).
- `int = obj.GetGenerateCellTCoords ()` - Indicate that cell texture coordinates are to be generated. Note that the specified number of components is used to create the texture coordinates (but must range between one and three).
- `obj.GenerateCellTCoordsOn ()` - Indicate that cell texture coordinates are to be generated. Note that the specified number of components is used to create the texture coordinates (but must range between one and three).
- `obj.GenerateCellTCoordsOff ()` - Indicate that cell texture coordinates are to be generated. Note that the specified number of components is used to create the texture coordinates (but must range between one and three).
- `obj.SetGenerateCellArray (int )` - Indicate that an arbitrary cell array is to be generated. Note that the specified number of components is used to create the array.
- `int = obj.GetGenerateCellArray ()` - Indicate that an arbitrary cell array is to be generated. Note that the specified number of components is used to create the array.
- `obj.GenerateCellArrayOn ()` - Indicate that an arbitrary cell array is to be generated. Note that the specified number of components is used to create the array.
- `obj.GenerateCellArrayOff ()` - Indicate that an arbitrary cell array is to be generated. Note that the specified number of components is used to create the array.
- `obj.SetGenerateFieldArray (int )` - Indicate that an arbitrary field data array is to be generated. Note that the specified number of components is used to create the scalar.
- `int = obj.GetGenerateFieldArray ()` - Indicate that an arbitrary field data array is to be generated. Note that the specified number of components is used to create the scalar.
- `obj.GenerateFieldArrayOn ()` - Indicate that an arbitrary field data array is to be generated. Note that the specified number of components is used to create the scalar.
- `obj.GenerateFieldArrayOff ()` - Indicate that an arbitrary field data array is to be generated. Note that the specified number of components is used to create the scalar.
- `obj.GenerateAllPointDataOn ()` - Convenience methods for generating data: all data, all point data, or all cell data. For example, if all data is enabled, then all point, cell and field data is generated. If all point data is enabled, then point scalars, vectors, normals, tensors, tcoords, and a data array are produced.
- `obj.GenerateAllPointDataOff ()` - Convenience methods for generating data: all data, all point data, or all cell data. For example, if all data is enabled, then all point, cell and field data is generated. If all point data is enabled, then point scalars, vectors, normals, tensors, tcoords, and a data array are produced.
- `obj.GenerateAllCellDataOn ()` - Convenience methods for generating data: all data, all point data, or all cell data. For example, if all data is enabled, then all point, cell and field data is generated. If all point data is enabled, then point scalars, vectors, normals, tensors, tcoords, and a data array are produced.

- `obj.GenerateAllCellDataOff ()` - Convenience methods for generating data: all data, all point data, or all cell data. For example, if all data is enabled, then all point, cell and field data is generated. If all point data is enabled, then point scalars, vectors, normals, tensors, tcoords, and a data array are produced.
- `obj.GenerateAllDataOn ()` - Convenience methods for generating data: all data, all point data, or all cell data. For example, if all data is enabled, then all point, cell and field data is generated. If all point data is enabled, then point scalars, vectors, normals, tensors, tcoords, and a data array are produced.
- `obj.GenerateAllDataOff ()`

33.182 vtkRearrangeFields

33.182.1 Usage

vtkRearrangeFields is used to copy/move fields (vtkDataArrays) between data object's field data, point data and cell data. To specify which fields are copied/moved, the user adds operations. There are two types of operations: 1. the type which copies/moves an attribute's data (i.e. the field will be copied but will not be an attribute in the target), 2. the type which copies/moves fields by name. For example: @verbatim rf-¿AddOperation(vtkRearrangeFields::COPY, "foo", vtkRearrangeFields::DATA_OBJECT, vtkRearrangeFields::POINT_DATA); @endverbatim adds an operation which copies a field (data array) called foo from the data object's field data to point data. From Tcl, the same operation can be added as follows: @verbatim rf AddOperation COPY foo DATA_OBJECT POINT_DATA @endverbatim The same can be done using Python and Java bindings by passing strings as arguments. @verbatim Operation types: COPY, MOVE AttributeTypes: SCALARS, VECTORS, NORMALS, TCOORDS, TENSORS Field data locations: DATA_OBJECT, POINT_DATA, CELL_DATA @endverbatim

To create an instance of class vtkRearrangeFields, simply invoke its constructor as follows

```
obj = vtkRearrangeFields
```

33.182.2 Methods

The class vtkRearrangeFields has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRearrangeFields class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRearrangeFields = obj.NewInstance ()`
- `vtkRearrangeFields = obj.SafeDownCast (vtkObject o)`
- `int = obj.AddOperation (int operationType, int attributeType, int fromFieldLoc, int toFieldLoc)` - Add an operation which copies an attribute's field (data array) from one field data to another. Returns an operation id which can later be used to remove the operation.
- `int = obj.AddOperation (int operationType, string name, int fromFieldLoc, int toFieldLoc)` - Add an operation which copies a field (data array) from one field data to another. Returns an operation id which can later be used to remove the operation.
- `int = obj.AddOperation (string operationType, string attributeType, string fromFieldLoc, string toF` - Helper method used by other language bindings. Allows the caller to specify arguments as strings instead of enums.Returns an operation id which can later be used to remove the operation.

- `int = obj.RemoveOperation (int operationId)` - Remove an operation with the given id.
- `int = obj.RemoveOperation (int operationType, int attributeType, int fromFieldLoc, int toFieldLoc)` - Remove an operation with the given signature. See AddOperation for details.
- `int = obj.RemoveOperation (int operationType, string name, int fromFieldLoc, int toFieldLoc)` - Remove an operation with the given signature. See AddOperation for details.
- `int = obj.RemoveOperation (string operationType, string attributeType, string fromFieldLoc, string` - Remove an operation with the given signature. See AddOperation for details.
- `obj.RemoveAllOperations ()`

33.183 vtkRectangularButtonSource

33.183.1 Usage

vtkRectangularButtonSource creates a rectangular shaped button with texture coordinates suitable for application of a texture map. This provides a way to make nice looking 3D buttons. The buttons are represented as vtkPolyData that includes texture coordinates and normals. The button lies in the x-y plane.

To use this class you must define its width, height and length. These measurements are all taken with respect to the shoulder of the button. The shoulder is defined as follows. Imagine a box sitting on the floor. The distance from the floor to the top of the box is the depth; the other directions are the length (x-direction) and height (y-direction). In this particular widget the box can have a smaller bottom than top. The ratio in size between bottom and top is called the box ratio (by default=1.0). The ratio of the texture region to the shoulder region is the texture ratio. And finally the texture region may be out of plane compared to the shoulder. The texture height ratio controls this.

To create an instance of class vtkRectangularButtonSource, simply invoke its constructor as follows

```
obj = vtkRectangularButtonSource
```

33.183.2 Methods

The class vtkRectangularButtonSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectangularButtonSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectangularButtonSource = obj.NewInstance ()`
- `vtkRectangularButtonSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetWidth (double )` - Set/Get the width of the button.
- `double = obj.GetWidthMinValue ()` - Set/Get the width of the button.
- `double = obj.GetWidthMaxValue ()` - Set/Get the width of the button.
- `double = obj.GetWidth ()` - Set/Get the width of the button.
- `obj.SetHeight (double )` - Set/Get the height of the button.
- `double = obj.GetHeightMinValue ()` - Set/Get the height of the button.
- `double = obj.GetHeightMaxValue ()` - Set/Get the height of the button.

- `double = obj.GetHeight ()` - Set/Get the height of the button.
- `obj.SetDepth (double )` - Set/Get the depth of the button (the z-eliipsoid axis length).
- `double = obj.GetDepthMinValue ()` - Set/Get the depth of the button (the z-eliipsoid axis length).
- `double = obj.GetDepthMaxValue ()` - Set/Get the depth of the button (the z-eliipsoid axis length).
- `double = obj.GetDepth ()` - Set/Get the depth of the button (the z-eliipsoid axis length).
- `obj.SetBoxRatio (double )` - Set/Get the ratio of the bottom of the button with the shoulder region. Numbers greater than one produce buttons with a wider bottom than shoulder; ratios less than one produce buttons that have a wider shoulder than bottom.
- `double = obj.GetBoxRatioMinValue ()` - Set/Get the ratio of the bottom of the button with the shoulder region. Numbers greater than one produce buttons with a wider bottom than shoulder; ratios less than one produce buttons that have a wider shoulder than bottom.
- `double = obj.GetBoxRatioMaxValue ()` - Set/Get the ratio of the bottom of the button with the shoulder region. Numbers greater than one produce buttons with a wider bottom than shoulder; ratios less than one produce buttons that have a wider shoulder than bottom.
- `double = obj.GetBoxRatio ()` - Set/Get the ratio of the bottom of the button with the shoulder region. Numbers greater than one produce buttons with a wider bottom than shoulder; ratios less than one produce buttons that have a wider shoulder than bottom.
- `obj.SetTextureRatio (double )` - Set/Get the ratio of the texture region to the shoulder region. This number must be 0¡=tr¡=1. If the texture style is to fit the image, then satisfying the texture ratio may only be possible in one of the two directions (length or width) depending on the dimensions of the texture.
- `double = obj.GetTextureRatioMinValue ()` - Set/Get the ratio of the texture region to the shoulder region. This number must be 0¡=tr¡=1. If the texture style is to fit the image, then satisfying the texture ratio may only be possible in one of the two directions (length or width) depending on the dimensions of the texture.
- `double = obj.GetTextureRatioMaxValue ()` - Set/Get the ratio of the texture region to the shoulder region. This number must be 0¡=tr¡=1. If the texture style is to fit the image, then satisfying the texture ratio may only be possible in one of the two directions (length or width) depending on the dimensions of the texture.
- `double = obj.GetTextureRatio ()` - Set/Get the ratio of the texture region to the shoulder region. This number must be 0¡=tr¡=1. If the texture style is to fit the image, then satisfying the texture ratio may only be possible in one of the two directions (length or width) depending on the dimensions of the texture.
- `obj.SetTextureHeightRatio (double )` - Set/Get the ratio of the height of the texture region to the shoulder height. Values greater than 1.0 yield convex buttons with the texture region raised above the shoulder. Values less than 1.0 yield concave buttons with the texture region below the shoulder.
- `double = obj.GetTextureHeightRatioMinValue ()` - Set/Get the ratio of the height of the texture region to the shoulder height. Values greater than 1.0 yield convex buttons with the texture region raised above the shoulder. Values less than 1.0 yield concave buttons with the texture region below the shoulder.
- `double = obj.GetTextureHeightRatioMaxValue ()` - Set/Get the ratio of the height of the texture region to the shoulder height. Values greater than 1.0 yield convex buttons with the texture region raised above the shoulder. Values less than 1.0 yield concave buttons with the texture region below the shoulder.

- `double = obj.GetTextureHeightRatio ()` - Set/Get the ratio of the height of the texture region to the shoulder height. Values greater than 1.0 yield convex buttons with the texture region raised above the shoulder. Values less than 1.0 yield concave buttons with the texture region below the shoulder.

33.184 vtkRectilinearGridClip

33.184.1 Usage

vtkRectilinearGridClip will make an image smaller. The output must have an image extent which is the subset of the input. The filter has two modes of operation: 1: By default, the data is not copied in this filter. Only the whole extent is modified. 2: If ClipDataOn is set, then you will get no more that the clipped extent.

To create an instance of class vtkRectilinearGridClip, simply invoke its constructor as follows

```
obj = vtkRectilinearGridClip
```

33.184.2 Methods

The class vtkRectilinearGridClip has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGridClip class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGridClip = obj.NewInstance ()`
- `vtkRectilinearGridClip = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutputWholeExtent (int extent[6], vtkInformation outInfo)` - The whole extent of the output has to be set explicitly.
- `obj.SetOutputWholeExtent (int minX, int maxX, int minY, int maxY, int minZ, int maxZ)` - The whole extent of the output has to be set explicitly.
- `obj.GetOutputWholeExtent (int extent[6])` - The whole extent of the output has to be set explicitly.
- `obj.ResetOutputWholeExtent ()`
- `obj.SetClipData (int )` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `int = obj.GetClipData ()` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `obj.ClipDataOn ()` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `obj.ClipDataOff ()` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `obj.SetOutputWholeExtent (int piece, int numPieces)` - Hack set output by piece

33.185 vtkRectilinearGridGeometryFilter

33.185.1 Usage

vtkRectilinearGridGeometryFilter is a filter that extracts geometry from a rectilinear grid. By specifying appropriate i-j-k indices, it is possible to extract a point, a curve, a surface, or a "volume". The volume is actually a (n x m x o) region of points.

The extent specification is zero-offset. That is, the first k-plane in a 50x50x50 rectilinear grid is given by (0,49, 0,49, 0,0).

To create an instance of class vtkRectilinearGridGeometryFilter, simply invoke its constructor as follows

```
obj = vtkRectilinearGridGeometryFilter
```

33.185.2 Methods

The class vtkRectilinearGridGeometryFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGridGeometryFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGridGeometryFilter = obj.NewInstance ()`
- `vtkRectilinearGridGeometryFilter = obj.SafeDownCast (vtkObject o)`
- `int = obj. GetExtent ()` - Get the extent in topological coordinate range (imin,imax, jmin,jmax, kmin,kmax).
- `obj.SetExtent (int iMin, int iMax, int jMin, int jMax, int kMin, int kMax)` - Specify (imin,imax, jmin,jmax, kmin,kmax) indices.
- `obj.SetExtent (int extent[6])` - Specify (imin,imax, jmin,jmax, kmin,kmax) indices in array form.

33.186 vtkRectilinearGridToTetrahedra

33.186.1 Usage

vtkRectilinearGridToTetrahedra forms a mesh of Tetrahedra from a vtkRectilinearGrid. The tetrahedra can be 5 per cell, 6 per cell, or a mixture of 5 or 12 per cell. The resulting mesh is consistent, meaning that there are no edge crossings and that each tetrahedron face is shared by two tetrahedra, except those tetrahedra on the boundary. All tetrahedra are right handed.

Note that 12 tetrahedra per cell means adding a point in the center of the cell.

In order to subdivide some cells into 5 and some cells into 12 tetrahedra: SetTetraPerCellTo5And12(); Set the Scalars of the Input RectilinearGrid to be 5 or 12 depending on what you want per cell of the RectilinearGrid.

If you set RememberVoxelId, the scalars of the tetrahedron will be set to the Id of the Cell in the RectilinearGrid from which the tetrahedron came.

.SECTION Thanks This class was developed by Samson J. Timoner of the MIT Artificial Intelligence Laboratory

To create an instance of class vtkRectilinearGridToTetrahedra, simply invoke its constructor as follows

```
obj = vtkRectilinearGridToTetrahedra
```

33.186.2 Methods

The class vtkRectilinearGridToTetrahedra has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGridToTetrahedra class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGridToTetrahedra = obj.NewInstance ()`
- `vtkRectilinearGridToTetrahedra = obj.SafeDownCast (vtkObject o)`
- `obj.SetTetraPerCellTo5 ()` - Set the method to divide each cell (voxel) in the RectilinearGrid into tetrahedra.
- `obj.SetTetraPerCellTo6 ()` - Set the method to divide each cell (voxel) in the RectilinearGrid into tetrahedra.
- `obj.SetTetraPerCellTo12 ()` - Set the method to divide each cell (voxel) in the RectilinearGrid into tetrahedra.
- `obj.SetTetraPerCellTo5And12 ()` - Set the method to divide each cell (voxel) in the RectilinearGrid into tetrahedra.
- `obj.SetTetraPerCell (int )` - Set the method to divide each cell (voxel) in the RectilinearGrid into tetrahedra.
- `int = obj.GetTetraPerCell ()` - Set the method to divide each cell (voxel) in the RectilinearGrid into tetrahedra.
- `obj.SetRememberVoxelId (int )` - Should the tetrahedra have scalar data indicating which Voxel they came from in the vtkRectilinearGrid?
- `int = obj.GetRememberVoxelId ()` - Should the tetrahedra have scalar data indicating which Voxel they came from in the vtkRectilinearGrid?
- `obj.RememberVoxelIdOn ()` - Should the tetrahedra have scalar data indicating which Voxel they came from in the vtkRectilinearGrid?
- `obj.RememberVoxelIdOff ()` - Should the tetrahedra have scalar data indicating which Voxel they came from in the vtkRectilinearGrid?
- `obj.SetInput (double Extent[3], double Spacing[3], double tol)` - This function for convenience for creating a Rectilinear Grid If Spacing does not fit evenly into extent, the last cell will have a different width (or height or depth). If Extent[i]/Spacing[i] is within tol of an integer, then assume the programmer meant an integer for direction i.
- `obj.SetInput (double ExtentX, double ExtentY, double ExtentZ, double SpacingX, double SpacingY, dou` - This version of the function for the wrappers

33.187 vtkRectilinearSynchronizedTemplates

33.187.1 Usage

vtkRectilinearSynchronizedTemplates is a 3D implementation (for rectilinear grids) of the synchronized template algorithm. Note that vtkContourFilter will automatically use this class when appropriate.

To create an instance of class vtkRectilinearSynchronizedTemplates, simply invoke its constructor as follows

```
obj = vtkRectilinearSynchronizedTemplates
```

33.187.2 Methods

The class vtkRectilinearSynchronizedTemplates has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearSynchronizedTemplates class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearSynchronizedTemplates = obj.NewInstance ()`
- `vtkRectilinearSynchronizedTemplates = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Because we delegate to vtkContourValues
- `obj.SetComputeNormals (int )` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeNormals ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOn ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOff ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeGradients (int )` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeGradients ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOn ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOff ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeScalars (int )` - Set/Get the computation of scalars.
- `int = obj.GetComputeScalars ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOn ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOff ()` - Set/Get the computation of scalars.

- `obj.SetValue (int i, double value)` - Get the ith contour value.
- `double = obj.GetValue (int i)` - Get a pointer to an array of contour values. There will be GetNumberOfContours() values in the list.
- `obj.GetValues (double contourValues)` - Set the number of contours to place into the list. You only really need to use this method to reduce list size. The method SetValue() will automatically increase list size as needed.
- `obj.SetNumberOfContours (int number)` - Get the number of contours in the list of contour values.
- `int = obj.GetNumberOfContours ()` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double range[2])` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Needed by templated functions.
- `obj.SetArrayComponent (int )` - Set/get which component of the scalar array to contour on; defaults to 0.
- `int = obj.GetArrayComponent ()` - Set/get which component of the scalar array to contour on; defaults to 0.
- `obj.ComputeSpacing (vtkRectilinearGrid data, int i, int j, int k, int extent[6], double spacing[6])` - Compute the spacing between this point and its 6 neighbors. This method needs to be public so it can be accessed from a templated function.

33.188 vtkRecursiveDividingCubes

33.188.1 Usage

vtkRecursiveDividingCubes is a filter that generates points lying on a surface of constant scalar value (i.e., an isosurface). Dense point clouds (i.e., at screen resolution) will appear as a surface. Less dense clouds can be used as a source to generate streamlines or to generate "transparent" surfaces.

This implementation differs from vtkDividingCubes in that it uses a recursive procedure. In many cases this can result in generating more points than the procedural implementation of vtkDividingCubes. This is because the recursive procedure divides voxels by multiples of powers of two. This can over-constrain subdivision. One of the advantages of the recursive technique is that the recursion is terminated earlier, which in some cases can be more efficient.

To create an instance of class vtkRecursiveDividingCubes, simply invoke its constructor as follows

```
obj = vtkRecursiveDividingCubes
```

33.188.2 Methods

The class vtkRecursiveDividingCubes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRecursiveDividingCubes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRecursiveDividingCubes = obj.NewInstance ()`

- `vtkRecursiveDividingCubes = obj.SafeDownCast (vtkObject o)`
- `obj.SetValue (double )` - Set isosurface value.
- `double = obj.GetValue ()` - Set isosurface value.
- `obj.SetDistance (double )` - Specify sub-voxel size at which to generate point.
- `double = obj.GetDistanceMinValue ()` - Specify sub-voxel size at which to generate point.
- `double = obj.GetDistanceMaxValue ()` - Specify sub-voxel size at which to generate point.
- `double = obj.GetDistance ()` - Specify sub-voxel size at which to generate point.
- `obj.SetIncrement (int )` - Every "Increment" point is added to the list of points. This parameter, if set to a large value, can be used to limit the number of points while retaining good accuracy.
- `int = obj.GetIncrementMinValue ()` - Every "Increment" point is added to the list of points. This parameter, if set to a large value, can be used to limit the number of points while retaining good accuracy.
- `int = obj.GetIncrementMaxValue ()` - Every "Increment" point is added to the list of points. This parameter, if set to a large value, can be used to limit the number of points while retaining good accuracy.
- `int = obj.GetIncrement ()` - Every "Increment" point is added to the list of points. This parameter, if set to a large value, can be used to limit the number of points while retaining good accuracy.

33.189 vtkReflectionFilter

33.189.1 Usage

The vtkReflectionFilter reflects a data set across one of the planes formed by the data set's bounding box. Since it converts data sets into unstructured grids, it is not effeicient for structured data sets.

To create an instance of class vtkReflectionFilter, simply invoke its constructor as follows

```
obj = vtkReflectionFilter
```

33.189.2 Methods

The class vtkReflectionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkReflectionFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkReflectionFilter = obj.NewInstance ()`
- `vtkReflectionFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetPlane (int )` - Set the normal of the plane to use as mirror.
- `int = obj.GetPlaneMinValue ()` - Set the normal of the plane to use as mirror.
- `int = obj.GetPlaneMaxValue ()` - Set the normal of the plane to use as mirror.
- `int = obj.GetPlane ()` - Set the normal of the plane to use as mirror.

- `obj.SetPlaneToX ()` - Set the normal of the plane to use as mirror.
- `obj.SetPlaneToY ()` - Set the normal of the plane to use as mirror.
- `obj.SetPlaneToZ ()` - Set the normal of the plane to use as mirror.
- `obj.SetPlaneToXMin ()` - Set the normal of the plane to use as mirror.
- `obj.SetPlaneToYMin ()` - Set the normal of the plane to use as mirror.
- `obj.SetPlaneToZMin ()` - Set the normal of the plane to use as mirror.
- `obj.SetPlaneToXMax ()` - Set the normal of the plane to use as mirror.
- `obj.SetPlaneToYMax ()` - Set the normal of the plane to use as mirror.
- `obj.SetPlaneToZMax ()` - Set the normal of the plane to use as mirror.
- `obj.SetCenter (double )` - If the reflection plane is set to X, Y or Z, this variable is use to set the position of the plane.
- `double = obj.GetCenter ()` - If the reflection plane is set to X, Y or Z, this variable is use to set the position of the plane.
- `obj.SetCopyInput (int )` - If on (the default), copy the input geometry to the output. If off, the output will only contain the reflection.
- `int = obj.GetCopyInput ()` - If on (the default), copy the input geometry to the output. If off, the output will only contain the reflection.
- `obj.CopyInputOn ()` - If on (the default), copy the input geometry to the output. If off, the output will only contain the reflection.
- `obj.CopyInputOff ()` - If on (the default), copy the input geometry to the output. If off, the output will only contain the reflection.

33.190 vtkRegularPolygonSource

33.190.1 Usage

vtkRegularPolygonSource is a source object that creates a single n-sided polygon and/or polyline. The polygon is centered at a specified point, orthogonal to a specified normal, and with a circumscribing radius set by the user. The user can also specify the number of sides of the polygon ranging from [3,N].

This object can be used for seeding streamlines or defining regions for clipping/cutting.

To create an instance of class vtkRegularPolygonSource, simply invoke its constructor as follows

```
obj = vtkRegularPolygonSource
```

33.190.2 Methods

The class vtkRegularPolygonSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRegularPolygonSource class.

- `string = obj.GetClassName ()` - Standard methods for instantiation, obtaining type and printing instance values.
- `int = obj.IsA (string name)` - Standard methods for instantiation, obtaining type and printing instance values.

- `vtkRegularPolygonSource = obj.NewInstance ()` - Standard methods for instantiation, obtaining type and printing instance values.
- `vtkRegularPolygonSource = obj.SafeDownCast (vtkObject o)` - Standard methods for instantiation, obtaining type and printing instance values.
- `obj.SetNumberOfSides (int )` - Set/Get the number of sides of the polygon. By default, the number of sides is set to six.
- `int = obj.GetNumberOfSidesMinValue ()` - Set/Get the number of sides of the polygon. By default, the number of sides is set to six.
- `int = obj.GetNumberOfSidesMaxValue ()` - Set/Get the number of sides of the polygon. By default, the number of sides is set to six.
- `int = obj.GetNumberOfSides ()` - Set/Get the number of sides of the polygon. By default, the number of sides is set to six.
- `obj.SetCenter (double , double , double )` - Set/Get the center of the polygon. By default, the center is set at the origin (0,0,0).
- `obj.SetCenter (double  a[3])` - Set/Get the center of the polygon. By default, the center is set at the origin (0,0,0).
- `double = obj. GetCenter ()` - Set/Get the center of the polygon. By default, the center is set at the origin (0,0,0).
- `obj.SetNormal (double , double , double )` - Set/Get the normal to the polygon. The ordering of the polygon will be counter-clockwise around the normal (i.e., using the right-hand rule). By default, the normal is set to (0,0,1).
- `obj.SetNormal (double  a[3])` - Set/Get the normal to the polygon. The ordering of the polygon will be counter-clockwise around the normal (i.e., using the right-hand rule). By default, the normal is set to (0,0,1).
- `double = obj. GetNormal ()` - Set/Get the normal to the polygon. The ordering of the polygon will be counter-clockwise around the normal (i.e., using the right-hand rule). By default, the normal is set to (0,0,1).
- `obj.SetRadius (double )` - Set/Get the radius of the polygon. By default, the radius is set to 0.5.
- `double = obj.GetRadius ()` - Set/Get the radius of the polygon. By default, the radius is set to 0.5.
- `obj.SetGeneratePolygon (int )` - Control whether a polygon is produced. By default, GeneratePolygon is enabled.
- `int = obj.GetGeneratePolygon ()` - Control whether a polygon is produced. By default, GeneratePolygon is enabled.
- `obj.GeneratePolygonOn ()` - Control whether a polygon is produced. By default, GeneratePolygon is enabled.
- `obj.GeneratePolygonOff ()` - Control whether a polygon is produced. By default, GeneratePolygon is enabled.
- `obj.SetGeneratePolyline (int )` - Control whether a polyline is produced. By default, GeneratePolyline is enabled.
- `int = obj.GetGeneratePolyline ()` - Control whether a polyline is produced. By default, GeneratePolyline is enabled.

- `obj.GeneratePolylineOn ()` - Control whether a polyline is produced. By default, GeneratePolyline is enabled.
- `obj.GeneratePolylineOff ()` - Control whether a polyline is produced. By default, GeneratePolyline is enabled.

33.191 vtkReverseSense

33.191.1 Usage

vtkReverseSense is a filter that reverses the order of polygonal cells and/or reverses the direction of point and cell normals. Two flags are used to control these operations. Cell reversal means reversing the order of indices in the cell connectivity list. Normal reversal means multiplying the normal vector by -1 (both point and cell normals, if present).

To create an instance of class vtkReverseSense, simply invoke its constructor as follows

```
obj = vtkReverseSense
```

33.191.2 Methods

The class vtkReverseSense has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkReverseSense class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkReverseSense = obj.NewInstance ()`
- `vtkReverseSense = obj.SafeDownCast (vtkObject o)`
- `obj.SetReverseCells (int )` - Flag controls whether to reverse cell ordering.
- `int = obj.GetReverseCells ()` - Flag controls whether to reverse cell ordering.
- `obj.ReverseCellsOn ()` - Flag controls whether to reverse cell ordering.
- `obj.ReverseCellsOff ()` - Flag controls whether to reverse cell ordering.
- `obj.SetReverseNormals (int )` - Flag controls whether to reverse normal orientation.
- `int = obj.GetReverseNormals ()` - Flag controls whether to reverse normal orientation.
- `obj.ReverseNormalsOn ()` - Flag controls whether to reverse normal orientation.
- `obj.ReverseNormalsOff ()` - Flag controls whether to reverse normal orientation.

33.192 vtkRibbonFilter

33.192.1 Usage

vtkRibbonFilter is a filter to create oriented ribbons from lines defined in polygonal dataset. The orientation of the ribbon is along the line segments and perpendicular to "projected" line normals. Projected line normals are the original line normals projected to be perpendicular to the local line segment. An offset angle can be specified to rotate the ribbon with respect to the normal.

To create an instance of class vtkRibbonFilter, simply invoke its constructor as follows

```
obj = vtkRibbonFilter
```

33.192.2 Methods

The class vtkRibbonFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRibbonFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRibbonFilter = obj.NewInstance ()`
- `vtkRibbonFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetWidth (double )` - Set the "half" width of the ribbon. If the width is allowed to vary, this is the minimum width. The default is 0.5
- `double = obj.GetWidthMinValue ()` - Set the "half" width of the ribbon. If the width is allowed to vary, this is the minimum width. The default is 0.5
- `double = obj.GetWidthMaxValue ()` - Set the "half" width of the ribbon. If the width is allowed to vary, this is the minimum width. The default is 0.5
- `double = obj.GetWidth ()` - Set the "half" width of the ribbon. If the width is allowed to vary, this is the minimum width. The default is 0.5
- `obj.SetAngle (double )` - Set the offset angle of the ribbon from the line normal. (The angle is expressed in degrees.) The default is 0.0
- `double = obj.GetAngleMinValue ()` - Set the offset angle of the ribbon from the line normal. (The angle is expressed in degrees.) The default is 0.0
- `double = obj.GetAngleMaxValue ()` - Set the offset angle of the ribbon from the line normal. (The angle is expressed in degrees.) The default is 0.0
- `double = obj.GetAngle ()` - Set the offset angle of the ribbon from the line normal. (The angle is expressed in degrees.) The default is 0.0
- `obj.SetVaryWidth (int )` - Turn on/off the variation of ribbon width with scalar value. The default is Off
- `int = obj.GetVaryWidth ()` - Turn on/off the variation of ribbon width with scalar value. The default is Off
- `obj.VaryWidthOn ()` - Turn on/off the variation of ribbon width with scalar value. The default is Off
- `obj.VaryWidthOff ()` - Turn on/off the variation of ribbon width with scalar value. The default is Off
- `obj.SetWidthFactor (double )` - Set the maximum ribbon width in terms of a multiple of the minimum width. The default is 2.0
- `double = obj.GetWidthFactor ()` - Set the maximum ribbon width in terms of a multiple of the minimum width. The default is 2.0
- `obj.SetDefaultNormal (double , double , double )` - Set the default normal to use if no normals are supplied, and DefaultNormalOn is set. The default is (0,0,1)
- `obj.SetDefaultNormal (double  a[3])` - Set the default normal to use if no normals are supplied, and DefaultNormalOn is set. The default is (0,0,1)

- `double = obj. GetDefaultNormal ()` - Set the default normal to use if no normals are supplied, and DefaultNormalOn is set. The default is (0,0,1)
- `obj.SetUseDefaultNormal (int )` - Set a boolean to control whether to use default normals. The default is Off
- `int = obj.GetUseDefaultNormal ()` - Set a boolean to control whether to use default normals. The default is Off
- `obj.UseDefaultNormalOn ()` - Set a boolean to control whether to use default normals. The default is Off
- `obj.UseDefaultNormalOff ()` - Set a boolean to control whether to use default normals. The default is Off
- `obj.SetGenerateTCoords (int )` - Control whether and how texture coordinates are produced. This is useful for striping the ribbon with time textures, etc.
- `int = obj.GetGenerateTCoordsMinValue ()` - Control whether and how texture coordinates are produced. This is useful for striping the ribbon with time textures, etc.
- `int = obj.GetGenerateTCoordsMaxValue ()` - Control whether and how texture coordinates are produced. This is useful for striping the ribbon with time textures, etc.
- `int = obj.GetGenerateTCoords ()` - Control whether and how texture coordinates are produced. This is useful for striping the ribbon with time textures, etc.
- `obj.SetGenerateTCoordsToOff ()` - Control whether and how texture coordinates are produced. This is useful for striping the ribbon with time textures, etc.
- `obj.SetGenerateTCoordsToNormalizedLength ()` - Control whether and how texture coordinates are produced. This is useful for striping the ribbon with time textures, etc.
- `obj.SetGenerateTCoordsToUseLength ()` - Control whether and how texture coordinates are produced. This is useful for striping the ribbon with time textures, etc.
- `obj.SetGenerateTCoordsToUseScalars ()` - Control whether and how texture coordinates are produced. This is useful for striping the ribbon with time textures, etc.
- `string = obj.GetGenerateTCoordsAsString ()` - Control whether and how texture coordinates are produced. This is useful for striping the ribbon with time textures, etc.
- `obj.SetTextureLength (double )` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space. The default is 1.0
- `double = obj.GetTextureLengthMinValue ()` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space. The default is 1.0
- `double = obj.GetTextureLengthMaxValue ()` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space. The default is 1.0
- `double = obj.GetTextureLength ()` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space. The default is 1.0

33.193 vtkRotationalExtrusionFilter

33.193.1 Usage

vtkRotationalExtrusionFilter is a modeling filter. It takes polygonal data as input and generates polygonal data on output. The input dataset is swept around the z-axis to create new polygonal primitives. These primitives form a "skirt" or swept surface. For example, sweeping a line results in a cylindrical shell, and sweeping a circle creates a torus.

There are a number of control parameters for this filter. You can control whether the sweep of a 2D object (i.e., polygon or triangle strip) is capped with the generating geometry via the "Capping" instance variable. Also, you can control the angle of rotation, and whether translation along the z-axis is performed along with the rotation. (Translation is useful for creating "springs".) You also can adjust the radius of the generating geometry using the "DeltaRotation" instance variable.

The skirt is generated by locating certain topological features. Free edges (edges of polygons or triangle strips only used by one polygon or triangle strips) generate surfaces. This is true also of lines or polylines. Vertices generate lines.

This filter can be used to model axisymmetric objects like cylinders, bottles, and wine glasses; or translational/rotational symmetric objects like springs or corkscrews.

To create an instance of class vtkRotationalExtrusionFilter, simply invoke its constructor as follows

```
obj = vtkRotationalExtrusionFilter
```

33.193.2 Methods

The class vtkRotationalExtrusionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRotationalExtrusionFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRotationalExtrusionFilter = obj.NewInstance ()`
- `vtkRotationalExtrusionFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetResolution (int )` - Set/Get resolution of sweep operation. Resolution controls the number of intermediate node points.
- `int = obj.GetResolutionMinValue ()` - Set/Get resolution of sweep operation. Resolution controls the number of intermediate node points.
- `int = obj.GetResolutionMaxValue ()` - Set/Get resolution of sweep operation. Resolution controls the number of intermediate node points.
- `int = obj.GetResolution ()` - Set/Get resolution of sweep operation. Resolution controls the number of intermediate node points.
- `obj.SetCapping (int )` - Turn on/off the capping of the skirt.
- `int = obj.GetCapping ()` - Turn on/off the capping of the skirt.
- `obj.CappingOn ()` - Turn on/off the capping of the skirt.
- `obj.CappingOff ()` - Turn on/off the capping of the skirt.
- `obj.SetAngle (double )` - Set/Get angle of rotation.
- `double = obj.GetAngle ()` - Set/Get angle of rotation.

- `obj.SetTranslation (double )` - Set/Get total amount of translation along the z-axis.
- `double = obj.GetTranslation ()` - Set/Get total amount of translation along the z-axis.
- `obj.SetDeltaRadius (double )` - Set/Get change in radius during sweep process.
- `double = obj.GetDeltaRadius ()` - Set/Get change in radius during sweep process.

33.194 vtkRotationFilter

33.194.1 Usage

The vtkRotationFilter duplicates a data set by rotation about one of the 3 axis of the dataset's reference. Since it converts data sets into unstructured grids, it is not efficient for structured data sets.

.SECTION Thanks Theophane Foggia of The Swiss National Supercomputing Centre (CSCS) for creating and contributing this filter

To create an instance of class vtkRotationFilter, simply invoke its constructor as follows

```
obj = vtkRotationFilter
```

33.194.2 Methods

The class vtkRotationFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRotationFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRotationFilter = obj.NewInstance ()`
- `vtkRotationFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetAxis (int )` - Set the axis of rotation to use. It is set by default to Z.
- `int = obj.GetAxisMinValue ()` - Set the axis of rotation to use. It is set by default to Z.
- `int = obj.GetAxisMaxValue ()` - Set the axis of rotation to use. It is set by default to Z.
- `int = obj.GetAxis ()` - Set the axis of rotation to use. It is set by default to Z.
- `obj.SetAxisToX ()` - Set the axis of rotation to use. It is set by default to Z.
- `obj.SetAxisToY ()` - Set the axis of rotation to use. It is set by default to Z.
- `obj.SetAxisToZ ()` - Set the axis of rotation to use. It is set by default to Z.
- `obj.SetAngle (double )` - Set the rotation angle to use.
- `double = obj.GetAngle ()` - Set the rotation angle to use.
- `obj.SetCenter (double , double , double )` - Set the rotation center coordinates.
- `obj.SetCenter (double  a[3])` - Set the rotation center coordinates.
- `double = obj. GetCenter ()` - Set the rotation center coordinates.
- `obj.SetNumberOfCopies (int )` - Set the number of copies to create. The source will be rotated N times and a new polydata copy of the original created at each angular position All copies will be appended to form a single output

- `int = obj.GetNumberOfCopies ()` - Set the number of copies to create. The source will be rotated N times and a new polydata copy of the original created at each angular position All copies will be appended to form a single output
- `obj.SetCopyInput (int )` - If on (the default), copy the input geometry to the output. If off, the output will only contain the rotation.
- `int = obj.GetCopyInput ()` - If on (the default), copy the input geometry to the output. If off, the output will only contain the rotation.
- `obj.CopyInputOn ()` - If on (the default), copy the input geometry to the output. If off, the output will only contain the rotation.
- `obj.CopyInputOff ()` - If on (the default), copy the input geometry to the output. If off, the output will only contain the rotation.

33.195 vtkRuledSurfaceFilter

33.195.1 Usage

vtkRuledSurfaceFilter is a filter that generates a surface from a set of lines. The lines are assumed to be "parallel" in the sense that they do not intersect and remain somewhat close to one another. A surface is generated by connecting the points defining each pair of lines with straight lines. This creates a strip for each pair of lines (i.e., a triangulation is created from two generating lines). The filter can handle an arbitrary number of lines, with lines i and i+1 assumed connected. Note that there are several different approaches for creating the ruled surface, the method for creating the surface can either use the input points or resample from the polylines (using a user-specified resolution).

This filter offers some other important features. A DistanceFactor ivar is used to decide when two lines are too far apart to connect. (The factor is a multiple of the distance between the first two points of the two lines defining the strip.) If the distance between the two generating lines becomes too great, then the surface is not generated in that region. (Note: if the lines separate and then merge, then a hole can be generated in the surface.) In addition, the Offset and OnRation ivars can be used to create nifty striped surfaces. Closed surfaces (e.g., tubes) can be created by setting the CloseSurface ivar. (The surface can be closed in the other direction by repeating the first and last point in the polylines defining the surface.)

An important use of this filter is to combine it with vtkStreamLine to generate stream surfaces. It can also be used to create surfaces from contours.

To create an instance of class vtkRuledSurfaceFilter, simply invoke its constructor as follows

```
obj = vtkRuledSurfaceFilter
```

33.195.2 Methods

The class vtkRuledSurfaceFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRuledSurfaceFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRuledSurfaceFilter = obj.NewInstance ()`
- `vtkRuledSurfaceFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetDistanceFactor (double )` - Set/Get the factor that controls tearing of the surface.

- `double = obj.GetDistanceFactorMinValue ()` - Set/Get the factor that controls tearing of the surface.
- `double = obj.GetDistanceFactorMaxValue ()` - Set/Get the factor that controls tearing of the surface.
- `double = obj.GetDistanceFactor ()` - Set/Get the factor that controls tearing of the surface.
- `obj.SetOnRatio (int )` - Control the striping of the ruled surface. If OnRatio is greater than 1, then every nth strip is turned on, beginning with the Offset strip.
- `int = obj.GetOnRatioMinValue ()` - Control the striping of the ruled surface. If OnRatio is greater than 1, then every nth strip is turned on, beginning with the Offset strip.
- `int = obj.GetOnRatioMaxValue ()` - Control the striping of the ruled surface. If OnRatio is greater than 1, then every nth strip is turned on, beginning with the Offset strip.
- `int = obj.GetOnRatio ()` - Control the striping of the ruled surface. If OnRatio is greater than 1, then every nth strip is turned on, beginning with the Offset strip.
- `obj.SetOffset (int )` - Control the striping of the ruled surface. The offset sets the first stripe that is visible. Offset is generally used with OnRatio to create nifty striping effects.
- `int = obj.GetOffsetMinValue ()` - Control the striping of the ruled surface. The offset sets the first stripe that is visible. Offset is generally used with OnRatio to create nifty striping effects.
- `int = obj.GetOffsetMaxValue ()` - Control the striping of the ruled surface. The offset sets the first stripe that is visible. Offset is generally used with OnRatio to create nifty striping effects.
- `int = obj.GetOffset ()` - Control the striping of the ruled surface. The offset sets the first stripe that is visible. Offset is generally used with OnRatio to create nifty striping effects.
- `obj.SetCloseSurface (int )` - Indicate whether the surface is to be closed. If this boolean is on, then the first and last polyline are used to generate a stripe that closes the surface. (Note: to close the surface in the other direction, repeat the first point in the polyline as the last point in the polyline.)
- `int = obj.GetCloseSurface ()` - Indicate whether the surface is to be closed. If this boolean is on, then the first and last polyline are used to generate a stripe that closes the surface. (Note: to close the surface in the other direction, repeat the first point in the polyline as the last point in the polyline.)
- `obj.CloseSurfaceOn ()` - Indicate whether the surface is to be closed. If this boolean is on, then the first and last polyline are used to generate a stripe that closes the surface. (Note: to close the surface in the other direction, repeat the first point in the polyline as the last point in the polyline.)
- `obj.CloseSurfaceOff ()` - Indicate whether the surface is to be closed. If this boolean is on, then the first and last polyline are used to generate a stripe that closes the surface. (Note: to close the surface in the other direction, repeat the first point in the polyline as the last point in the polyline.)
- `obj.SetRuledMode (int )` - Set the mode by which to create the ruled surface. (Dramatically different results are possible depending on the chosen mode.) The resample mode evenly resamples the polylines (based on length) and generates triangle strips. The point walk mode uses the existing points and walks around the polyline using existing points.
- `int = obj.GetRuledModeMinValue ()` - Set the mode by which to create the ruled surface. (Dramatically different results are possible depending on the chosen mode.) The resample mode evenly resamples the polylines (based on length) and generates triangle strips. The point walk mode uses the existing points and walks around the polyline using existing points.

- `int = obj.GetRuledModeMaxValue ()` - Set the mode by which to create the ruled surface. (Dramatically different results are possible depending on the chosen mode.) The resample mode evenly resamples the polylines (based on length) and generates triangle strips. The point walk mode uses the existing points and walks around the polyline using existing points.
- `int = obj.GetRuledMode ()` - Set the mode by which to create the ruled surface. (Dramatically different results are possible depending on the chosen mode.) The resample mode evenly resamples the polylines (based on length) and generates triangle strips. The point walk mode uses the existing points and walks around the polyline using existing points.
- `obj.SetRuledModeToResample ()` - Set the mode by which to create the ruled surface. (Dramatically different results are possible depending on the chosen mode.) The resample mode evenly resamples the polylines (based on length) and generates triangle strips. The point walk mode uses the existing points and walks around the polyline using existing points.
- `obj.SetRuledModeToPointWalk ()` - Set the mode by which to create the ruled surface. (Dramatically different results are possible depending on the chosen mode.) The resample mode evenly resamples the polylines (based on length) and generates triangle strips. The point walk mode uses the existing points and walks around the polyline using existing points.
- `string = obj.GetRuledModeAsString ()` - Set the mode by which to create the ruled surface. (Dramatically different results are possible depending on the chosen mode.) The resample mode evenly resamples the polylines (based on length) and generates triangle strips. The point walk mode uses the existing points and walks around the polyline using existing points.
- `obj.SetResolution (int , int )` - If the ruled surface generation mode is RESAMPLE, then these parameters are used to determine the resample rate. Resolution[0] defines the resolution in the direction of the polylines; Resolution[1] defines the resolution across the polylines (i.e., direction orthogonal to Resolution[0]).
- `obj.SetResolution (int a[2])` - If the ruled surface generation mode is RESAMPLE, then these parameters are used to determine the resample rate. Resolution[0] defines the resolution in the direction of the polylines; Resolution[1] defines the resolution across the polylines (i.e., direction orthogonal to Resolution[0]).
- `int = obj. GetResolution ()` - If the ruled surface generation mode is RESAMPLE, then these parameters are used to determine the resample rate. Resolution[0] defines the resolution in the direction of the polylines; Resolution[1] defines the resolution across the polylines (i.e., direction orthogonal to Resolution[0]).
- `obj.SetPassLines (int )` - Indicate whether the generating lines are to be passed to the output. By defualt lines are not passed to the output.
- `int = obj.GetPassLines ()` - Indicate whether the generating lines are to be passed to the output. By defualt lines are not passed to the output.
- `obj.PassLinesOn ()` - Indicate whether the generating lines are to be passed to the output. By defualt lines are not passed to the output.
- `obj.PassLinesOff ()` - Indicate whether the generating lines are to be passed to the output. By defualt lines are not passed to the output.
- `obj.SetOrientLoops (int )` - Indicate whether the starting points of the loops need to be determined. If set to 0, then its assumes that the 0th point of each loop should be always connected By defualt the loops are not oriented.
- `int = obj.GetOrientLoops ()` - Indicate whether the starting points of the loops need to be determined. If set to 0, then its assumes that the 0th point of each loop should be always connected By defualt the loops are not oriented.

- `obj.OrientLoopsOn ()` - Indicate whether the starting points of the loops need to be determined. If set to 0, then its assumes that the 0th point of each loop should be always connected By defualt the loops are not oriented.
- `obj.OrientLoopsOff ()` - Indicate whether the starting points of the loops need to be determined. If set to 0, then its assumes that the 0th point of each loop should be always connected By defualt the loops are not oriented.

33.196 vtkSectorSource

33.196.1 Usage

vtkSectorSource creates a sector of a polygonal disk. The disk has zero height. The user can specify the inner and outer radius of the disk, the z-coordinate, and the radial and circumferential resolution of the polygonal representation.

To create an instance of class vtkSectorSource, simply invoke its constructor as follows

```
obj = vtkSectorSource
```

33.196.2 Methods

The class vtkSectorSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSectorSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSectorSource = obj.NewInstance ()`
- `vtkSectorSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetInnerRadius (double )` - Specify inner radius of the sector.
- `double = obj.GetInnerRadiusMinValue ()` - Specify inner radius of the sector.
- `double = obj.GetInnerRadiusMaxValue ()` - Specify inner radius of the sector.
- `double = obj.GetInnerRadius ()` - Specify inner radius of the sector.
- `obj.SetOuterRadius (double )` - Specify outer radius of the sector.
- `double = obj.GetOuterRadiusMinValue ()` - Specify outer radius of the sector.
- `double = obj.GetOuterRadiusMaxValue ()` - Specify outer radius of the sector.
- `double = obj.GetOuterRadius ()` - Specify outer radius of the sector.
- `obj.SetZCoord (double )` - Specify the z coordinate of the sector.
- `double = obj.GetZCoordMinValue ()` - Specify the z coordinate of the sector.
- `double = obj.GetZCoordMaxValue ()` - Specify the z coordinate of the sector.
- `double = obj.GetZCoord ()` - Specify the z coordinate of the sector.
- `obj.SetRadialResolution (int )` - Set the number of points in radius direction.
- `int = obj.GetRadialResolutionMinValue ()` - Set the number of points in radius direction.

- `int = obj.GetRadialResolutionMaxValue ()` - Set the number of points in radius direction.
- `int = obj.GetRadialResolution ()` - Set the number of points in radius direction.
- `obj.SetCircumferentialResolution (int )` - Set the number of points in circumferential direction.
- `int = obj.GetCircumferentialResolutionMinValue ()` - Set the number of points in circumferential direction.
- `int = obj.GetCircumferentialResolutionMaxValue ()` - Set the number of points in circumferential direction.
- `int = obj.GetCircumferentialResolution ()` - Set the number of points in circumferential direction.
- `obj.SetStartAngle (double )` - Set the start angle of the sector.
- `double = obj.GetStartAngleMinValue ()` - Set the start angle of the sector.
- `double = obj.GetStartAngleMaxValue ()` - Set the start angle of the sector.
- `double = obj.GetStartAngle ()` - Set the start angle of the sector.
- `obj.SetEndAngle (double )` - Set the end angle of the sector.
- `double = obj.GetEndAngleMinValue ()` - Set the end angle of the sector.
- `double = obj.GetEndAngleMaxValue ()` - Set the end angle of the sector.
- `double = obj.GetEndAngle ()` - Set the end angle of the sector.

33.197 vtkSelectEnclosedPoints

33.197.1 Usage

vtkSelectEnclosedPoints is a filter that evaluates all the input points to determine whether they are in an enclosed surface. The filter produces a (0,1) mask (in the form of a vtkDataArray) that indicates whether points are outside (mask value=0) or inside (mask value=1) a provided surface. (The name of the output vtkDataArray is "SelectedPointsArray".)

After running the filter, it is possible to query it as to whether a point is inside/outside by invoking the IsInside(ptId) method.

To create an instance of class vtkSelectEnclosedPoints, simply invoke its constructor as follows

```
obj = vtkSelectEnclosedPoints
```

33.197.2 Methods

The class vtkSelectEnclosedPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSelectEnclosedPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSelectEnclosedPoints = obj.NewInstance ()`
- `vtkSelectEnclosedPoints = obj.SafeDownCast (vtkObject o)`

- `obj.SetSurface (vtkPolyData pd)` - Set the surface to be used to test for containment. Two methods are provided: one directly for vtkPolyData, and one for the output of a filter.
- `obj.SetSurfaceConnection (vtkAlgorithmOutput algOutput)` - Set the surface to be used to test for containment. Two methods are provided: one directly for vtkPolyData, and one for the output of a filter.
- `vtkPolyData = obj.GetSurface ()` - Return a pointer to the enclosing surface.
- `vtkPolyData = obj.GetSurface (vtkInformationVector sourceInfo)` - Return a pointer to the enclosing surface.
- `obj.SetInsideOut (int )` - By default, points inside the surface are marked inside or sent to the output. If InsideOut is on, then the points outside the surface are marked inside.
- `obj.InsideOutOn ()` - By default, points inside the surface are marked inside or sent to the output. If InsideOut is on, then the points outside the surface are marked inside.
- `obj.InsideOutOff ()` - By default, points inside the surface are marked inside or sent to the output. If InsideOut is on, then the points outside the surface are marked inside.
- `int = obj.GetInsideOut ()` - By default, points inside the surface are marked inside or sent to the output. If InsideOut is on, then the points outside the surface are marked inside.
- `obj.SetCheckSurface (int )` - Specify whether to check the surface for closure. If on, then the algorithm first checks to see if the surface is closed and manifold.
- `obj.CheckSurfaceOn ()` - Specify whether to check the surface for closure. If on, then the algorithm first checks to see if the surface is closed and manifold.
- `obj.CheckSurfaceOff ()` - Specify whether to check the surface for closure. If on, then the algorithm first checks to see if the surface is closed and manifold.
- `int = obj.GetCheckSurface ()` - Specify whether to check the surface for closure. If on, then the algorithm first checks to see if the surface is closed and manifold.
- `int = obj.IsInside (vtkIdType inputPtId)` - Query an input point id as to whether it is inside or outside. Note that the result requires that the filter execute first.
- `obj.SetTolerance (double )` - Specify the tolerance on the intersection. The tolerance is expressed as a fraction of the bounding box of the enclosing surface.
- `double = obj.GetToleranceMinValue ()` - Specify the tolerance on the intersection. The tolerance is expressed as a fraction of the bounding box of the enclosing surface.
- `double = obj.GetToleranceMaxValue ()` - Specify the tolerance on the intersection. The tolerance is expressed as a fraction of the bounding box of the enclosing surface.
- `double = obj.GetTolerance ()` - Specify the tolerance on the intersection. The tolerance is expressed as a fraction of the bounding box of the enclosing surface.
- `obj.Initialize (vtkPolyData surface)` - This is a backdoor that can be used to test many points for containment. First initialize the instance, then repeated calls to IsInsideSurface() can be used without rebuilding the search structures. The complete method releases memory.
- `int = obj.IsInsideSurface (double x, double y, double z)` - This is a backdoor that can be used to test many points for containment. First initialize the instance, then repeated calls to IsInsideSurface() can be used without rebuilding the search structures. The complete method releases memory.

- `int = obj.IsInsideSurface (double x[3])` - This is a backdoor that can be used to test many points for containment. First initialize the instance, then repeated calls to IsInsideSurface() can be used without rebuilding the search structures. The complete method releases memory.

- `obj.Complete ()` - This is a backdoor that can be used to test many points for containment. First initialize the instance, then repeated calls to IsInsideSurface() can be used without rebuilding the search structures. The complete method releases memory.

33.198 vtkSelectPolyData

33.198.1 Usage

vtkSelectPolyData is a filter that selects polygonal data based on defining a "loop" and indicating the region inside of the loop. The mesh within the loop consists of complete cells (the cells are not cut). Alternatively, this filter can be used to generate scalars. These scalar values, which are a distance measure to the loop, can be used to clip, contour. or extract data (i.e., anything that an implicit function can do).

The loop is defined by an array of x-y-z point coordinates. (Coordinates should be in the same coordinate space as the input polygonal data.) The loop can be concave and non-planar, but not self-intersecting. The input to the filter is a polygonal mesh (only surface primitives such as triangle strips and polygons); the output is either a) a portion of the original mesh laying within the selection loop (GenerateSelectionScalarsOff); or b) the same polygonal mesh with the addition of scalar values (GenerateSelectionScalarsOn).

The algorithm works as follows. For each point coordinate in the loop, the closest point in the mesh is found. The result is a loop of closest point ids from the mesh. Then, the edges in the mesh connecting the closest points (and laying along the lines forming the loop) are found. A greedy edge tracking procedure is used as follows. At the current point, the mesh edge oriented in the direction of and whose end point is closest to the line is chosen. The edge is followed to the new end point, and the procedure is repeated. This process continues until the entire loop has been created.

To determine what portion of the mesh is inside and outside of the loop, three options are possible. 1) the smallest connected region, 2) the largest connected region, and 3) the connected region closest to a user specified point. (Set the ivar SelectionMode.)

Once the loop is computed as above, the GenerateSelectionScalars controls the output of the filter. If on, then scalar values are generated based on distance to the loop lines. Otherwise, the cells laying inside the selection loop are output. By default, the mesh lying within the loop is output; however, if InsideOut is on, then the portion of the mesh lying outside of the loop is output.

The filter can be configured to generate the unselected portions of the mesh as output by setting GenerateUnselectedOutput. Use the method GetUnselectedOutput to access this output. (Note: this flag is pertinent only when GenerateSelectionScalars is off.)

To create an instance of class vtkSelectPolyData, simply invoke its constructor as follows

```
obj = vtkSelectPolyData
```

33.198.2 Methods

The class vtkSelectPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSelectPolyData class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkSelectPolyData = obj.NewInstance ()`

- `vtkSelectPolyData = obj.SafeDownCast (vtkObject o)`

- `obj.SetGenerateSelectionScalars (int )` - Set/Get the flag to control behavior of the filter. If GenerateSelectionScalars is on, then the output of the filter is the same as the input, except that scalars are generated. If off, the filter outputs the cells laying inside the loop, and does not generate scalars.
- `int = obj.GetGenerateSelectionScalars ()` - Set/Get the flag to control behavior of the filter. If GenerateSelectionScalars is on, then the output of the filter is the same as the input, except that scalars are generated. If off, the filter outputs the cells laying inside the loop, and does not generate scalars.
- `obj.GenerateSelectionScalarsOn ()` - Set/Get the flag to control behavior of the filter. If GenerateSelectionScalars is on, then the output of the filter is the same as the input, except that scalars are generated. If off, the filter outputs the cells laying inside the loop, and does not generate scalars.
- `obj.GenerateSelectionScalarsOff ()` - Set/Get the flag to control behavior of the filter. If GenerateSelectionScalars is on, then the output of the filter is the same as the input, except that scalars are generated. If off, the filter outputs the cells laying inside the loop, and does not generate scalars.
- `obj.SetInsideOut (int )` - Set/Get the InsideOut flag. When off, the mesh within the loop is extracted. When on, the mesh outside the loop is extracted.
- `int = obj.GetInsideOut ()` - Set/Get the InsideOut flag. When off, the mesh within the loop is extracted. When on, the mesh outside the loop is extracted.
- `obj.InsideOutOn ()` - Set/Get the InsideOut flag. When off, the mesh within the loop is extracted. When on, the mesh outside the loop is extracted.
- `obj.InsideOutOff ()` - Set/Get the InsideOut flag. When off, the mesh within the loop is extracted. When on, the mesh outside the loop is extracted.
- `obj.SetLoop (vtkPoints )` - Set/Get the array of point coordinates defining the loop. There must be at least three points used to define a loop.
- `vtkPoints = obj.GetLoop ()` - Set/Get the array of point coordinates defining the loop. There must be at least three points used to define a loop.
- `obj.SetSelectionMode (int )` - Control how inside/outside of loop is defined.
- `int = obj.GetSelectionModeMinValue ()` - Control how inside/outside of loop is defined.
- `int = obj.GetSelectionModeMaxValue ()` - Control how inside/outside of loop is defined.
- `int = obj.GetSelectionMode ()` - Control how inside/outside of loop is defined.
- `obj.SetSelectionModeToSmallestRegion ()` - Control how inside/outside of loop is defined.
- `obj.SetSelectionModeToLargestRegion ()` - Control how inside/outside of loop is defined.
- `obj.SetSelectionModeToClosestPointRegion ()` - Control how inside/outside of loop is defined.
- `string = obj.GetSelectionModeAsString ()` - Control how inside/outside of loop is defined.
- `obj.SetGenerateUnselectedOutput (int )` - Control whether a second output is generated. The second output contains the polygonal data that's not been selected.
- `int = obj.GetGenerateUnselectedOutput ()` - Control whether a second output is generated. The second output contains the polygonal data that's not been selected.
- `obj.GenerateUnselectedOutputOn ()` - Control whether a second output is generated. The second output contains the polygonal data that's not been selected.
- `obj.GenerateUnselectedOutputOff ()` - Control whether a second output is generated. The second output contains the polygonal data that's not been selected.

- `vtkPolyData = obj.GetUnselectedOutput ()` - Return output that hasn't been selected (if GenreateUnselectedOutput is enabled).
- `vtkPolyData = obj.GetSelectionEdges ()` - Return the (mesh) edges of the selection region.
- `long = obj.GetMTime ()`

33.199 vtkShrinkFilter

33.199.1 Usage

vtkShrinkFilter shrinks cells composing an arbitrary data set towards their centroid. The centroid of a cell is computed as the average position of the cell points. Shrinking results in disconnecting the cells from one another. The output of this filter is of general dataset type vtkUnstructuredGrid.

To create an instance of class vtkShrinkFilter, simply invoke its constructor as follows

```
obj = vtkShrinkFilter
```

33.199.2 Methods

The class vtkShrinkFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkShrinkFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkShrinkFilter = obj.NewInstance ()`
- `vtkShrinkFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetShrinkFactor (double )` - Get/Set the fraction of shrink for each cell. The default is 0.5.
- `double = obj.GetShrinkFactorMinValue ()` - Get/Set the fraction of shrink for each cell. The default is 0.5.
- `double = obj.GetShrinkFactorMaxValue ()` - Get/Set the fraction of shrink for each cell. The default is 0.5.
- `double = obj.GetShrinkFactor ()` - Get/Set the fraction of shrink for each cell. The default is 0.5.

33.200 vtkShrinkPolyData

33.200.1 Usage

vtkShrinkPolyData shrinks cells composing a polygonal dataset (e.g., vertices, lines, polygons, and triangle strips) towards their centroid. The centroid of a cell is computed as the average position of the cell points. Shrinking results in disconnecting the cells from one another. The output dataset type of this filter is polygonal data.

During execution the filter passes its input cell data to its output. Point data attributes are copied to the points created during the shrinking process.

To create an instance of class vtkShrinkPolyData, simply invoke its constructor as follows

```
obj = vtkShrinkPolyData
```

33.200.2 Methods

The class vtkShrinkPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkShrinkPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkShrinkPolyData = obj.NewInstance ()`
- `vtkShrinkPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetShrinkFactor (double )` - Set the fraction of shrink for each cell.
- `double = obj.GetShrinkFactorMinValue ()` - Set the fraction of shrink for each cell.
- `double = obj.GetShrinkFactorMaxValue ()` - Set the fraction of shrink for each cell.
- `double = obj.GetShrinkFactor ()` - Get the fraction of shrink for each cell.

33.201 vtkSimpleElevationFilter

33.201.1 Usage

vtkSimpleElevationFilter is a filter to generate scalar values from a dataset. The scalar values are generated by dotting a user-specified vector against a vector defined from the input dataset points to the origin.

To create an instance of class vtkSimpleElevationFilter, simply invoke its constructor as follows

```
obj = vtkSimpleElevationFilter
```

33.201.2 Methods

The class vtkSimpleElevationFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSimpleElevationFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSimpleElevationFilter = obj.NewInstance ()`
- `vtkSimpleElevationFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetVector (double , double , double )` - Define one end of the line (small scalar values).
- `obj.SetVector (double  a[3])` - Define one end of the line (small scalar values).
- `double = obj. GetVector ()` - Define one end of the line (small scalar values).

33.202 vtkSliceCubes

33.202.1 Usage

vtkSliceCubes is a special version of the marching cubes filter. Instead of ingesting an entire volume at once it processes only four slices at a time. This way, it can generate isosurfaces from huge volumes. Also, the output of this object is written to a marching cubes triangle file. That way, output triangles do not need to be held in memory.

To use vtkSliceCubes you must specify an instance of vtkVolumeReader to read the data. Set this object up with the proper file prefix, image range, data origin, data dimensions, header size, data mask, and swap bytes flag. The vtkSliceCubes object will then take over and read slices as necessary. You also will need to specify the name of an output marching cubes triangle file.

To create an instance of class vtkSliceCubes, simply invoke its constructor as follows

```
obj = vtkSliceCubes
```

33.202.2 Methods

The class vtkSliceCubes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSliceCubes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSliceCubes = obj.NewInstance ()`
- `vtkSliceCubes = obj.SafeDownCast (vtkObject o)`
- `obj.Write ()`
- `obj.Update ()`
- `obj.SetReader (vtkVolumeReader )` - Set/get object to read slices.
- `vtkVolumeReader = obj.GetReader ()` - Set/get object to read slices.
- `obj.SetFileName (string )` - Specify file name of marching cubes output file.
- `string = obj.GetFileName ()` - Specify file name of marching cubes output file.
- `obj.SetValue (double )` - Set/get isosurface contour value.
- `double = obj.GetValue ()` - Set/get isosurface contour value.
- `obj.SetLimitsFileName (string )` - Specify file name of marching cubes limits file. The limits file speeds up subsequent reading of output triangle file.
- `string = obj.GetLimitsFileName ()` - Specify file name of marching cubes limits file. The limits file speeds up subsequent reading of output triangle file.

33.203 vtkSmoothPolyDataFilter

33.203.1 Usage

vtkSmoothPolyDataFilter is a filter that adjusts point coordinates using Laplacian smoothing. The effect is to "relax" the mesh, making the cells better shaped and the vertices more evenly distributed. Note that this filter operates on the lines, polygons, and triangle strips composing an instance of vtkPolyData. Vertex or poly-vertex cells are never modified.

The algorithm proceeds as follows. For each vertex v, a topological and geometric analysis is performed to determine which vertices are connected to v, and which cells are connected to v. Then, a connectivity array is constructed for each vertex. (The connectivity array is a list of lists of vertices that directly attach to each vertex.) Next, an iteration phase begins over all vertices. For each vertex v, the coordinates of v are modified according to an average of the connected vertices. (A relaxation factor is available to control the amount of displacement of v). The process repeats for each vertex. This pass over the list of vertices is a single iteration. Many iterations (generally around 20 or so) are repeated until the desired result is obtained.

There are some special instance variables used to control the execution of this filter. (These ivars basically control what vertices can be smoothed, and the creation of the connectivity array.) The BoundarySmoothing ivar enables/disables the smoothing operation on vertices that are on the "boundary" of the mesh. A boundary vertex is one that is surrounded by a semi-cycle of polygons (or used by a single line).

Another important ivar is FeatureEdgeSmoothing. If this ivar is enabled, then interior vertices are classified as either "simple", "interior edge", or "fixed", and smoothed differently. (Interior vertices are manifold vertices surrounded by a cycle of polygons; or used by two line cells.) The classification is based on the number of feature edges attached to v. A feature edge occurs when the angle between the two surface normals of a polygon sharing an edge is greater than the FeatureAngle ivar. Then, vertices used by no feature edges are classified "simple", vertices used by exactly two feature edges are classified "interior edge", and all others are "fixed" vertices.

Once the classification is known, the vertices are smoothed differently. Corner (i.e., fixed) vertices are not smoothed at all. Simple vertices are smoothed as before (i.e., average of connected vertex coordinates). Interior edge vertices are smoothed only along their two connected edges, and only if the angle between the edges is less than the EdgeAngle ivar.

The total smoothing can be controlled by using two ivars. The NumberOfIterations is a cap on the maximum number of smoothing passes. The Convergence ivar is a limit on the maximum point motion. If the maximum motion during an iteration is less than Convergence, then the smoothing process terminates. (Convergence is expressed as a fraction of the diagonal of the bounding box.)

There are two instance variables that control the generation of error data. If the ivar GenerateErrorScalars is on, then a scalar value indicating the distance of each vertex from its original position is computed. If the ivar GenerateErrorVectors is on, then a vector representing change in position is computed.

Optionally you can further control the smoothing process by defining a second input: the Source. If defined, the input mesh is constrained to lie on the surface defined by the Source ivar.

To create an instance of class vtkSmoothPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkSmoothPolyDataFilter
```

33.203.2 Methods

The class vtkSmoothPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSmoothPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSmoothPolyDataFilter = obj.NewInstance ()`
- `vtkSmoothPolyDataFilter = obj.SafeDownCast (vtkObject o)`

- `obj.SetConvergence (double )` - Specify a convergence criterion for the iteration process. Smaller numbers result in more smoothing iterations.
- `double = obj.GetConvergenceMinValue ()` - Specify a convergence criterion for the iteration process. Smaller numbers result in more smoothing iterations.
- `double = obj.GetConvergenceMaxValue ()` - Specify a convergence criterion for the iteration process. Smaller numbers result in more smoothing iterations.
- `double = obj.GetConvergence ()` - Specify a convergence criterion for the iteration process. Smaller numbers result in more smoothing iterations.
- `obj.SetNumberOfIterations (int )` - Specify the number of iterations for Laplacian smoothing,
- `int = obj.GetNumberOfIterationsMinValue ()` - Specify the number of iterations for Laplacian smoothing,
- `int = obj.GetNumberOfIterationsMaxValue ()` - Specify the number of iterations for Laplacian smoothing,
- `int = obj.GetNumberOfIterations ()` - Specify the number of iterations for Laplacian smoothing,
- `obj.SetRelaxationFactor (double )` - Specify the relaxation factor for Laplacian smoothing. As in all iterative methods, the stability of the process is sensitive to this parameter. In general, small relaxation factors and large numbers of iterations are more stable than larger relaxation factors and smaller numbers of iterations.
- `double = obj.GetRelaxationFactor ()` - Specify the relaxation factor for Laplacian smoothing. As in all iterative methods, the stability of the process is sensitive to this parameter. In general, small relaxation factors and large numbers of iterations are more stable than larger relaxation factors and smaller numbers of iterations.
- `obj.SetFeatureEdgeSmoothing (int )` - Turn on/off smoothing along sharp interior edges.
- `int = obj.GetFeatureEdgeSmoothing ()` - Turn on/off smoothing along sharp interior edges.
- `obj.FeatureEdgeSmoothingOn ()` - Turn on/off smoothing along sharp interior edges.
- `obj.FeatureEdgeSmoothingOff ()` - Turn on/off smoothing along sharp interior edges.
- `obj.SetFeatureAngle (double )` - Specify the feature angle for sharp edge identification.
- `double = obj.GetFeatureAngleMinValue ()` - Specify the feature angle for sharp edge identification.
- `double = obj.GetFeatureAngleMaxValue ()` - Specify the feature angle for sharp edge identification.
- `double = obj.GetFeatureAngle ()` - Specify the feature angle for sharp edge identification.
- `obj.SetEdgeAngle (double )` - Specify the edge angle to control smoothing along edges (either interior or boundary).
- `double = obj.GetEdgeAngleMinValue ()` - Specify the edge angle to control smoothing along edges (either interior or boundary).
- `double = obj.GetEdgeAngleMaxValue ()` - Specify the edge angle to control smoothing along edges (either interior or boundary).
- `double = obj.GetEdgeAngle ()` - Specify the edge angle to control smoothing along edges (either interior or boundary).
- `obj.SetBoundarySmoothing (int )` - Turn on/off the smoothing of vertices on the boundary of the mesh.

- `int = obj.GetBoundarySmoothing ()` - Turn on/off the smoothing of vertices on the boundary of the mesh.
- `obj.BoundarySmoothingOn ()` - Turn on/off the smoothing of vertices on the boundary of the mesh.
- `obj.BoundarySmoothingOff ()` - Turn on/off the smoothing of vertices on the boundary of the mesh.
- `obj.SetGenerateErrorScalars (int )` - Turn on/off the generation of scalar distance values.
- `int = obj.GetGenerateErrorScalars ()` - Turn on/off the generation of scalar distance values.
- `obj.GenerateErrorScalarsOn ()` - Turn on/off the generation of scalar distance values.
- `obj.GenerateErrorScalarsOff ()` - Turn on/off the generation of scalar distance values.
- `obj.SetGenerateErrorVectors (int )` - Turn on/off the generation of error vectors.
- `int = obj.GetGenerateErrorVectors ()` - Turn on/off the generation of error vectors.
- `obj.GenerateErrorVectorsOn ()` - Turn on/off the generation of error vectors.
- `obj.GenerateErrorVectorsOff ()` - Turn on/off the generation of error vectors.
- `obj.SetSource (vtkPolyData source)` - Specify the source object which is used to constrain smoothing. The source defines a surface that the input (as it is smoothed) is constrained to lie upon.
- `vtkPolyData = obj.GetSource ()` - Specify the source object which is used to constrain smoothing. The source defines a surface that the input (as it is smoothed) is constrained to lie upon.

33.204 vtkSpatialRepresentationFilter

33.204.1 Usage

vtkSpatialRepresentationFilter generates an polygonal representation of a spatial search (vtkLocator) object. The representation varies depending upon the nature of the spatial search object. For example, the representation for vtkOBBTree is a collection of oriented bounding boxes. Ths input to this filter is a dataset of any type, and the output is polygonal data. You must also specify the spatial search object to use.

Generally spatial search objects are used for collision detection and other geometric operations, but in this filter one or more levels of spatial searchers can be generated to form a geometric approximation to the input data. This is a form of data simplification, generally used to accelerate the rendering process. Or, this filter can be used as a debugging/ visualization aid for spatial search objects.

This filter can generate one or more output vtkPolyData corresponding to different levels in the spatial search tree. The output data is retrieved using the GetOutput(id) method, where id ranges from 0 (root level) to Level. Note that the output for level "id" is not computed unless a GetOutput(id) method is issued. Thus, if you desire three levels of output (say 2,4,7), you would have to invoke GetOutput(2), GetOutput(4), and GetOutput(7). (Also note that the Level ivar is computed automatically depending on the size and nature of the input data.) There is also another GetOutput() method that takes no parameters. This method returns the leafs of the spatial search tree, which may be at different levels.

To create an instance of class vtkSpatialRepresentationFilter, simply invoke its constructor as follows

```
obj = vtkSpatialRepresentationFilter
```

33.204.2 Methods

The class vtkSpatialRepresentationFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSpatialRepresentationFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSpatialRepresentationFilter = obj.NewInstance ()`
- `vtkSpatialRepresentationFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetSpatialRepresentation (vtkLocator )` - Set/Get the locator that will be used to generate the representation.
- `vtkLocator = obj.GetSpatialRepresentation ()` - Set/Get the locator that will be used to generate the representation.
- `int = obj.GetLevel ()` - Get the maximum number of outputs actually available.
- `vtkPolyData = obj.GetOutput (int level)` - A special form of the GetOutput() method that returns multiple outputs.
- `vtkPolyData = obj.GetOutput ()` - Output of terminal nodes/leaves.
- `obj.ResetOutput ()` - Reset requested output levels
- `obj.SetInput (vtkDataSet input)` - Set / get the input data or filter.
- `vtkDataSet = obj.GetInput ()` - Set / get the input data or filter.

33.205 vtkSpherePuzzle

33.205.1 Usage

vtkSpherePuzzle creates

To create an instance of class vtkSpherePuzzle, simply invoke its constructor as follows

```
obj = vtkSpherePuzzle
```

33.205.2 Methods

The class vtkSpherePuzzle has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSpherePuzzle class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSpherePuzzle = obj.NewInstance ()`
- `vtkSpherePuzzle = obj.SafeDownCast (vtkObject o)`
- `obj.Reset ()` - Reset the state of this puzzle back to its original state.

- `obj.MoveHorizontal (int section, int percentage, int rightFlag)` - Move the top/bottom half one segment either direction.

- `obj.MoveVertical (int section, int percentage, int rightFlag)` - Rotate vertical half of sphere along one of the longitude lines.

- `int = obj.SetPoint (double x, double y, double z)` - SetPoint will be called as the mouse moves over the screen. The output will change to indicate the pending move. SetPoint returns zero if move is not activated by point. Otherwise it encodes the move into a unique integer so that the caller can determine if the move state has changed. This will answer the question, "Should I render."

- `obj.MovePoint (int percentage)` - Move actually implements the pending move. When percentage is 100, the pending move becomes inactive, and SetPoint will have to be called again to setup another move.

33.206 vtkSpherePuzzleArrows

33.206.1 Usage

vtkSpherePuzzleArrows creates

To create an instance of class vtkSpherePuzzleArrows, simply invoke its constructor as follows

```
obj = vtkSpherePuzzleArrows
```

33.206.2 Methods

The class vtkSpherePuzzleArrows has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSpherePuzzleArrows class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkSpherePuzzleArrows = obj.NewInstance ()`

- `vtkSpherePuzzleArrows = obj.SafeDownCast (vtkObject o)`

- `obj.SetPermutation (int [32])` - Permutation is an array of puzzle piece ids. Arrows will be generated for any id that does not contain itself. Permutation[3] = 3 will produce no arrow. Permutation[3] = 10 will draw an arrow from location 3 to 10.

- `int = obj. GetPermutation ()` - Permutation is an array of puzzle piece ids. Arrows will be generated for any id that does not contain itself. Permutation[3] = 3 will produce no arrow. Permutation[3] = 10 will draw an arrow from location 3 to 10.

- `obj.SetPermutationComponent (int comp, int val)` - Permutation is an array of puzzle piece ids. Arrows will be generated for any id that does not contain itself. Permutation[3] = 3 will produce no arrow. Permutation[3] = 10 will draw an arrow from location 3 to 10.

- `obj.SetPermutation (vtkSpherePuzzle puz)` - Permutation is an array of puzzle piece ids. Arrows will be generated for any id that does not contain itself. Permutation[3] = 3 will produce no arrow. Permutation[3] = 10 will draw an arrow from location 3 to 10.

33.207 vtkSphereSource

33.207.1 Usage

vtkSphereSource creates a sphere (represented by polygons) of specified radius centered at the origin. The resolution (polygonal discretization) in both the latitude (phi) and longitude (theta) directions can be specified. It also is possible to create partial spheres by specifying maximum phi and theta angles. By default, the surface tessellation of the sphere uses triangles; however you can set LatLongTessellation to produce a tessellation using quadrilaterals.

To create an instance of class vtkSphereSource, simply invoke its constructor as follows

```
obj = vtkSphereSource
```

33.207.2 Methods

The class vtkSphereSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSphereSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSphereSource = obj.NewInstance ()`
- `vtkSphereSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetRadius (double )` - Set radius of sphere. Default is .5.
- `double = obj.GetRadiusMinValue ()` - Set radius of sphere. Default is .5.
- `double = obj.GetRadiusMaxValue ()` - Set radius of sphere. Default is .5.
- `double = obj.GetRadius ()` - Set radius of sphere. Default is .5.
- `obj.SetCenter (double , double , double )` - Set the center of the sphere. Default is 0,0,0.
- `obj.SetCenter (double  a[3])` - Set the center of the sphere. Default is 0,0,0.
- `double = obj. GetCenter ()` - Set the center of the sphere. Default is 0,0,0.
- `obj.SetThetaResolution (int )` - Set the number of points in the longitude direction (ranging from StartTheta to EndTheta).
- `int = obj.GetThetaResolutionMinValue ()` - Set the number of points in the longitude direction (ranging from StartTheta to EndTheta).
- `int = obj.GetThetaResolutionMaxValue ()` - Set the number of points in the longitude direction (ranging from StartTheta to EndTheta).
- `int = obj.GetThetaResolution ()` - Set the number of points in the longitude direction (ranging from StartTheta to EndTheta).
- `obj.SetPhiResolution (int )` - Set the number of points in the latitude direction (ranging from StartPhi to EndPhi).
- `int = obj.GetPhiResolutionMinValue ()` - Set the number of points in the latitude direction (ranging from StartPhi to EndPhi).
- `int = obj.GetPhiResolutionMaxValue ()` - Set the number of points in the latitude direction (ranging from StartPhi to EndPhi).

- `int = obj.GetPhiResolution ()` - Set the number of points in the latitude direction (ranging from StartPhi to EndPhi).
- `obj.SetStartTheta (double )` - Set the starting longitude angle. By default StartTheta=0 degrees.
- `double = obj.GetStartThetaMinValue ()` - Set the starting longitude angle. By default StartTheta=0 degrees.
- `double = obj.GetStartThetaMaxValue ()` - Set the starting longitude angle. By default StartTheta=0 degrees.
- `double = obj.GetStartTheta ()` - Set the starting longitude angle. By default StartTheta=0 degrees.
- `obj.SetEndTheta (double )` - Set the ending longitude angle. By default EndTheta=360 degrees.
- `double = obj.GetEndThetaMinValue ()` - Set the ending longitude angle. By default EndTheta=360 degrees.
- `double = obj.GetEndThetaMaxValue ()` - Set the ending longitude angle. By default EndTheta=360 degrees.
- `double = obj.GetEndTheta ()` - Set the ending longitude angle. By default EndTheta=360 degrees.
- `obj.SetStartPhi (double )` - Set the starting latitude angle (0 is at north pole). By default StartPhi=0 degrees.
- `double = obj.GetStartPhiMinValue ()` - Set the starting latitude angle (0 is at north pole). By default StartPhi=0 degrees.
- `double = obj.GetStartPhiMaxValue ()` - Set the starting latitude angle (0 is at north pole). By default StartPhi=0 degrees.
- `double = obj.GetStartPhi ()` - Set the starting latitude angle (0 is at north pole). By default StartPhi=0 degrees.
- `obj.SetEndPhi (double )` - Set the ending latitude angle. By default EndPhi=180 degrees.
- `double = obj.GetEndPhiMinValue ()` - Set the ending latitude angle. By default EndPhi=180 degrees.
- `double = obj.GetEndPhiMaxValue ()` - Set the ending latitude angle. By default EndPhi=180 degrees.
- `double = obj.GetEndPhi ()` - Set the ending latitude angle. By default EndPhi=180 degrees.
- `obj.SetLatLongTessellation (int )` - Cause the sphere to be tessellated with edges along the latitude and longitude lines. If off, triangles are generated at non-polar regions, which results in edges that are not parallel to latitude and longitude lines. If on, quadrilaterals are generated everywhere except at the poles. This can be useful for generating a wireframe sphere with natural latitude and longitude lines.
- `int = obj.GetLatLongTessellation ()` - Cause the sphere to be tessellated with edges along the latitude and longitude lines. If off, triangles are generated at non-polar regions, which results in edges that are not parallel to latitude and longitude lines. If on, quadrilaterals are generated everywhere except at the poles. This can be useful for generating a wireframe sphere with natural latitude and longitude lines.

- `obj.LatLongTessellationOn ()` - Cause the sphere to be tessellated with edges along the latitude and longitude lines. If off, triangles are generated at non-polar regions, which results in edges that are not parallel to latitude and longitude lines. If on, quadrilaterals are generated everywhere except at the poles. This can be useful for generating a wireframe sphere with natural latitude and longitude lines.

- `obj.LatLongTessellationOff ()` - Cause the sphere to be tessellated with edges along the latitude and longitude lines. If off, triangles are generated at non-polar regions, which results in edges that are not parallel to latitude and longitude lines. If on, quadrilaterals are generated everywhere except at the poles. This can be useful for generating a wireframe sphere with natural latitude and longitude lines.

33.208 vtkSplineFilter

33.208.1 Usage

vtkSplineFilter is a filter that generates an output polylines from an input set of polylines. The polylines are uniformly subdivided and produced with the help of a vtkSpline class that the user can specify (by default a vtkCardinalSpline is used). The number of subdivisions of the line can be controlled in several ways. The user can either specify the number of subdivisions or a length of each subdivision can be provided (and the class will figure out how many subdivisions is required over the whole polyline). The maximum number of subdivisions can also be set.

The output of this filter is a polyline per input polyline (or line). New points and texture coordinates are created. Point data is interpolated and cell data passed on. Any polylines with less than two points, or who have coincident points, are ignored.

To create an instance of class vtkSplineFilter, simply invoke its constructor as follows

```
obj = vtkSplineFilter
```

33.208.2 Methods

The class vtkSplineFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSplineFilter class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkSplineFilter = obj.NewInstance ()`

- `vtkSplineFilter = obj.SafeDownCast (vtkObject o)`

- `obj.SetMaximumNumberOfSubdivisions (int )` - Set the maximum number of subdivisions that are created for each polyline.

- `int = obj.GetMaximumNumberOfSubdivisionsMinValue ()` - Set the maximum number of subdivisions that are created for each polyline.

- `int = obj.GetMaximumNumberOfSubdivisionsMaxValue ()` - Set the maximum number of subdivisions that are created for each polyline.

- `int = obj.GetMaximumNumberOfSubdivisions ()` - Set the maximum number of subdivisions that are created for each polyline.

- `obj.SetSubdivide (int )` - Specify how the number of subdivisions is determined.

- `int = obj.GetSubdivideMinValue ()` - Specify how the number of subdivisions is determined.
- `int = obj.GetSubdivideMaxValue ()` - Specify how the number of subdivisions is determined.
- `int = obj.GetSubdivide ()` - Specify how the number of subdivisions is determined.
- `obj.SetSubdivideToSpecified ()` - Specify how the number of subdivisions is determined.
- `obj.SetSubdivideToLength ()` - Specify how the number of subdivisions is determined.
- `string = obj.GetSubdivideAsString ()` - Specify how the number of subdivisions is determined.
- `obj.SetNumberOfSubdivisions (int )` - Set the number of subdivisions that are created for the polyline. This method only has effect if Subdivisions is set to SetSubdivisionsToSpecify().
- `int = obj.GetNumberOfSubdivisionsMinValue ()` - Set the number of subdivisions that are created for the polyline. This method only has effect if Subdivisions is set to SetSubdivisionsToSpecify().
- `int = obj.GetNumberOfSubdivisionsMaxValue ()` - Set the number of subdivisions that are created for the polyline. This method only has effect if Subdivisions is set to SetSubdivisionsToSpecify().
- `int = obj.GetNumberOfSubdivisions ()` - Set the number of subdivisions that are created for the polyline. This method only has effect if Subdivisions is set to SetSubdivisionsToSpecify().
- `obj.SetLength (double )` - Control the number of subdivisions that are created for the polyline based on an absolute length. The length of the spline is divided by this length to determine the number of subdivisions.
- `double = obj.GetLengthMinValue ()` - Control the number of subdivisions that are created for the polyline based on an absolute length. The length of the spline is divided by this length to determine the number of subdivisions.
- `double = obj.GetLengthMaxValue ()` - Control the number of subdivisions that are created for the polyline based on an absolute length. The length of the spline is divided by this length to determine the number of subdivisions.
- `double = obj.GetLength ()` - Control the number of subdivisions that are created for the polyline based on an absolute length. The length of the spline is divided by this length to determine the number of subdivisions.
- `obj.SetSpline (vtkSpline )` - Specify an instance of vtkSpline to use to perform the interpolation.
- `vtkSpline = obj.GetSpline ()` - Specify an instance of vtkSpline to use to perform the interpolation.
- `obj.SetGenerateTCoords (int )` - Control whether and how texture coordinates are produced. This is useful for striping the output polyline. The texture coordinates can be generated in three ways: a normalized (0,1) generation; based on the length (divided by the texture length); and by using the input scalar values.
- `int = obj.GetGenerateTCoordsMinValue ()` - Control whether and how texture coordinates are produced. This is useful for striping the output polyline. The texture coordinates can be generated in three ways: a normalized (0,1) generation; based on the length (divided by the texture length); and by using the input scalar values.
- `int = obj.GetGenerateTCoordsMaxValue ()` - Control whether and how texture coordinates are produced. This is useful for striping the output polyline. The texture coordinates can be generated in three ways: a normalized (0,1) generation; based on the length (divided by the texture length); and by using the input scalar values.

- `int = obj.GetGenerateTCoords ()` - Control whether and how texture coordinates are produced. This is useful for striping the output polyline. The texture coordinates can be generated in three ways: a normalized (0,1) generation; based on the length (divided by the texture length); and by using the input scalar values.

- `obj.SetGenerateTCoordsToOff ()` - Control whether and how texture coordinates are produced. This is useful for striping the output polyline. The texture coordinates can be generated in three ways: a normalized (0,1) generation; based on the length (divided by the texture length); and by using the input scalar values.

- `obj.SetGenerateTCoordsToNormalizedLength ()` - Control whether and how texture coordinates are produced. This is useful for striping the output polyline. The texture coordinates can be generated in three ways: a normalized (0,1) generation; based on the length (divided by the texture length); and by using the input scalar values.

- `obj.SetGenerateTCoordsToUseLength ()` - Control whether and how texture coordinates are produced. This is useful for striping the output polyline. The texture coordinates can be generated in three ways: a normalized (0,1) generation; based on the length (divided by the texture length); and by using the input scalar values.

- `obj.SetGenerateTCoordsToUseScalars ()` - Control whether and how texture coordinates are produced. This is useful for striping the output polyline. The texture coordinates can be generated in three ways: a normalized (0,1) generation; based on the length (divided by the texture length); and by using the input scalar values.

- `string = obj.GetGenerateTCoordsAsString ()` - Control whether and how texture coordinates are produced. This is useful for striping the output polyline. The texture coordinates can be generated in three ways: a normalized (0,1) generation; based on the length (divided by the texture length); and by using the input scalar values.

- `obj.SetTextureLength (double )` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space.

- `double = obj.GetTextureLengthMinValue ()` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space.

- `double = obj.GetTextureLengthMaxValue ()` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space.

- `double = obj.GetTextureLength ()` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space.

33.209 vtkSplitField

33.209.1 Usage

vtkSplitField is used to split a multi-component field (vtkDataArray) into multiple single component fields. The new fields are put in the same field data as the original field. The output arrays are of the same type as the input array. Example: @verbatim sf-¿SetInputField("gradient", vtkSplitField::POINT_DATA); sf-¿Split(0, "firstcomponent"); @endverbatim tells vtkSplitField to extract the first component of the field called gradient and create an array called firstcomponent (the new field will be in the output's point data). The same can be done from Tcl: @verbatim sf SetInputField gradient POINT_DATA sf Split 0 firstcomponent

AttributeTypes: SCALARS, VECTORS, NORMALS, TCOORDS, TENSORS Field locations: DATA_OBJECT, POINT_DATA, CELL_DATA @endverbatim Note that, by default, the original array is also passed through.

To create an instance of class vtkSplitField, simply invoke its constructor as follows

```
obj = vtkSplitField
```

33.209.2 Methods

The class vtkSplitField has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSplitField class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSplitField = obj.NewInstance ()`
- `vtkSplitField = obj.SafeDownCast (vtkObject o)`
- `obj.SetInputField (int attributeType, int fieldLoc)` - Use the given attribute in the field data given by fieldLoc as input.
- `obj.SetInputField (string name, int fieldLoc)` - Use the array with given name in the field data given by fieldLoc as input.
- `obj.SetInputField (string name, string fieldLoc)` - Helper method used by other language bindings. Allows the caller to specify arguments as strings instead of enums.
- `obj.Split (int component, string arrayName)` - Create a new array with the given component.

33.210 vtkStreamer

33.210.1 Usage

vtkStreamer is a filter that integrates a massless particle through a vector field. The integration is performed using second order Runge-Kutta method. vtkStreamer often serves as a base class for other classes that perform numerical integration through a vector field (e.g., vtkStreamLine).

Note that vtkStreamer can integrate both forward and backward in time, or in both directions. The length of the streamer is controlled by specifying an elapsed time. (The elapsed time is the time each particle travels.) Otherwise, the integration terminates after exiting the dataset or if the particle speed is reduced to a value less than the terminal speed.

vtkStreamer integrates through any type of dataset. As a result, if the dataset contains 2D cells such as polygons or triangles, the integration is constrained to lie on the surface defined by the 2D cells.

The starting point of streamers may be defined in three different ways. Starting from global x-y-z "position" allows you to start a single streamer at a specified x-y-z coordinate. Starting from "location" allows you to start at a specified cell, subId, and parametric coordinate. Finally, you may specify a source object to start multiple streamers. If you start streamers using a source object, for each point in the source that is inside the dataset a streamer is created.

vtkStreamer implements the integration process in the Integrate() method. Because vtkStreamer does not implement the Execute() method that its superclass (i.e., Filter) requires, it is an abstract class. Its subclasses implement the execute method and use the Integrate() method, and then build their own representation of the integration path (i.e., lines, dashed lines, points, etc.).

To create an instance of class vtkStreamer, simply invoke its constructor as follows

```
obj = vtkStreamer
```

33.210.2 Methods

The class vtkStreamer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStreamer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStreamer = obj.NewInstance ()`
- `vtkStreamer = obj.SafeDownCast (vtkObject o)`
- `obj.SetStartLocation (vtkIdType cellId, int subId, double pcoords[3])` - Specify the start of the streamline in the cell coordinate system. That is, cellId and subId (if composite cell), and parametric coordinates.
- `obj.SetStartLocation (vtkIdType cellId, int subId, double r, double s, double t)` - Specify the start of the streamline in the cell coordinate system. That is, cellId and subId (if composite cell), and parametric coordinates.
- `obj.SetStartPosition (double x[3])` - Specify the start of the streamline in the global coordinate system. Search must be performed to find initial cell to start integration from.
- `obj.SetStartPosition (double x, double y, double z)` - Specify the start of the streamline in the global coordinate system. Search must be performed to find initial cell to start integration from.
- `double = obj.GetStartPosition ()` - Get the start position in global x-y-z coordinates.
- `obj.SetSource (vtkDataSet source)` - Specify the source object used to generate starting points.
- `vtkDataSet = obj.GetSource ()` - Specify the source object used to generate starting points.
- `obj.SetMaximumPropagationTime (double )` - Specify the maximum length of the Streamer expressed in elapsed time.
- `double = obj.GetMaximumPropagationTimeMinValue ()` - Specify the maximum length of the Streamer expressed in elapsed time.
- `double = obj.GetMaximumPropagationTimeMaxValue ()` - Specify the maximum length of the Streamer expressed in elapsed time.
- `double = obj.GetMaximumPropagationTime ()` - Specify the maximum length of the Streamer expressed in elapsed time.
- `obj.SetIntegrationDirection (int )` - Specify the direction in which to integrate the Streamer.
- `int = obj.GetIntegrationDirectionMinValue ()` - Specify the direction in which to integrate the Streamer.
- `int = obj.GetIntegrationDirectionMaxValue ()` - Specify the direction in which to integrate the Streamer.
- `int = obj.GetIntegrationDirection ()` - Specify the direction in which to integrate the Streamer.
- `obj.SetIntegrationDirectionToForward ()` - Specify the direction in which to integrate the Streamer.
- `obj.SetIntegrationDirectionToBackward ()` - Specify the direction in which to integrate the Streamer.
- `obj.SetIntegrationDirectionToIntegrateBothDirections ()` - Specify the direction in which to integrate the Streamer.

- `string = obj.GetIntegrationDirectionAsString ()` - Specify the direction in which to integrate the Streamer.
- `obj.SetIntegrationStepLength (double )` - Specify a nominal integration step size (expressed as a fraction of the size of each cell). This value can be larger than 1.
- `double = obj.GetIntegrationStepLengthMinValue ()` - Specify a nominal integration step size (expressed as a fraction of the size of each cell). This value can be larger than 1.
- `double = obj.GetIntegrationStepLengthMaxValue ()` - Specify a nominal integration step size (expressed as a fraction of the size of each cell). This value can be larger than 1.
- `double = obj.GetIntegrationStepLength ()` - Specify a nominal integration step size (expressed as a fraction of the size of each cell). This value can be larger than 1.
- `obj.SetSpeedScalars (int )` - Turn on/off the creation of scalar data from velocity magnitude. If off, and input dataset has scalars, input dataset scalars are used.
- `int = obj.GetSpeedScalars ()` - Turn on/off the creation of scalar data from velocity magnitude. If off, and input dataset has scalars, input dataset scalars are used.
- `obj.SpeedScalarsOn ()` - Turn on/off the creation of scalar data from velocity magnitude. If off, and input dataset has scalars, input dataset scalars are used.
- `obj.SpeedScalarsOff ()` - Turn on/off the creation of scalar data from velocity magnitude. If off, and input dataset has scalars, input dataset scalars are used.
- `obj.SetOrientationScalars (int )` - Turn on/off the creation of scalar data from vorticity information. The scalar information is currently the orientation value "theta" used in rotating stream tubes. If off, and input dataset has scalars, then input dataset scalars are used, unless SpeedScalars is also on. SpeedScalars takes precedence over OrientationScalars.
- `int = obj.GetOrientationScalars ()` - Turn on/off the creation of scalar data from vorticity information. The scalar information is currently the orientation value "theta" used in rotating stream tubes. If off, and input dataset has scalars, then input dataset scalars are used, unless SpeedScalars is also on. SpeedScalars takes precedence over OrientationScalars.
- `obj.OrientationScalarsOn ()` - Turn on/off the creation of scalar data from vorticity information. The scalar information is currently the orientation value "theta" used in rotating stream tubes. If off, and input dataset has scalars, then input dataset scalars are used, unless SpeedScalars is also on. SpeedScalars takes precedence over OrientationScalars.
- `obj.OrientationScalarsOff ()` - Turn on/off the creation of scalar data from vorticity information. The scalar information is currently the orientation value "theta" used in rotating stream tubes. If off, and input dataset has scalars, then input dataset scalars are used, unless SpeedScalars is also on. SpeedScalars takes precedence over OrientationScalars.
- `obj.SetTerminalSpeed (double )` - Set/get terminal speed (i.e., speed is velocity magnitude). Terminal speed is speed at which streamer will terminate propagation.
- `double = obj.GetTerminalSpeedMinValue ()` - Set/get terminal speed (i.e., speed is velocity magnitude). Terminal speed is speed at which streamer will terminate propagation.
- `double = obj.GetTerminalSpeedMaxValue ()` - Set/get terminal speed (i.e., speed is velocity magnitude). Terminal speed is speed at which streamer will terminate propagation.
- `double = obj.GetTerminalSpeed ()` - Set/get terminal speed (i.e., speed is velocity magnitude). Terminal speed is speed at which streamer will terminate propagation.

- `obj.SetVorticity (int )` - Turn on/off the computation of vorticity. Vorticity is an indication of the rotation of the flow. In combination with vtkStreamLine and vtkTubeFilter can be used to create rotated tubes. If vorticity is turned on, in the output, the velocity vectors are replaced by vorticity vectors.
- `int = obj.GetVorticity ()` - Turn on/off the computation of vorticity. Vorticity is an indication of the rotation of the flow. In combination with vtkStreamLine and vtkTubeFilter can be used to create rotated tubes. If vorticity is turned on, in the output, the velocity vectors are replaced by vorticity vectors.
- `obj.VorticityOn ()` - Turn on/off the computation of vorticity. Vorticity is an indication of the rotation of the flow. In combination with vtkStreamLine and vtkTubeFilter can be used to create rotated tubes. If vorticity is turned on, in the output, the velocity vectors are replaced by vorticity vectors.
- `obj.VorticityOff ()` - Turn on/off the computation of vorticity. Vorticity is an indication of the rotation of the flow. In combination with vtkStreamLine and vtkTubeFilter can be used to create rotated tubes. If vorticity is turned on, in the output, the velocity vectors are replaced by vorticity vectors.
- `obj.SetNumberOfThreads (int )`
- `int = obj.GetNumberOfThreads ()`
- `obj.SetSavePointInterval (double )`
- `double = obj.GetSavePointInterval ()`
- `obj.SetIntegrator (vtkInitialValueProblemSolver )` - Set/get the integrator type to be used in the stream line calculation. The object passed is not actually used but is cloned with NewInstance by each thread/process in the process of integration (prototype pattern). The default is 2nd order Runge Kutta.
- `vtkInitialValueProblemSolver = obj.GetIntegrator ()` - Set/get the integrator type to be used in the stream line calculation. The object passed is not actually used but is cloned with NewInstance by each thread/process in the process of integration (prototype pattern). The default is 2nd order Runge Kutta.
- `obj.SetEpsilon (double )` - A positive value, as small as possible for numerical comparison. The initial value is 1E-12.
- `double = obj.GetEpsilon ()` - A positive value, as small as possible for numerical comparison. The initial value is 1E-12.

33.211 vtkStreamingTessellator

33.211.1 Usage

This class is a simple algorithm that takes a single starting simplex – a tetrahedron, triangle, or line segment – and calls a function you pass it with (possibly many times) tetrahedra, triangles, or lines adaptively sampled from the one you specified. It uses an algorithm you specify to control the level of adaptivity.

This class does not create vtkUnstructuredGrid output because it is intended for use in mappers as well as filters. Instead, it calls the registered function with simplices as they are created.

The subdivision algorithm should change the vertex coordinates (it must change both geometric and, if desired, parametric coordinates) of the midpoint. These coordinates need not be changed unless the EvaluateEdge() member returns true. The vtkStreamingTessellator itself has no way of creating a more accurate midpoint vertex.

Here's how to use this class: - Call AdaptivelySample1Facet, AdaptivelySample2Facet, or AdaptivelySample3Facet, with an edge, triangle, or tetrahedron you want tessellated. - The adaptive tessellator classifies each edge by passing the midpoint values to the vtkEdgeSubdivisionCriterion. - After each edge is classified, the tessellator subdivides edges as required until the subdivision criterion is satisfied or the maximum subdivision depth has been reached. - Edges, triangles, or tetrahedra connecting the vertices generated by the subdivision algorithm are processed by calling the user-defined callback functions (set with SetTetrahedronCallback(), SetTriangleCallback(), or SetEdgeCallback()).

.SECTION Warning Note that the vertices passed to AdaptivelySample3Facet, AdaptivelySample2Facet, or AdaptivelySample1Facet must be at least 6, 5, or 4 entries long, respectively! This is because the <r,s,t>, <r,s>, or <r> parametric coordinates of the vertices are maintained as the facet is subdivided. This information is often required by the subdivision algorithm in order to compute an error metric. You may change the number of parametric coordinates associated with each vertex using vtkStreamingTessellator::SetEmbeddingDimension().

.SECTION Interpolating Field Values If you wish, you may also use vtkStreamingTessellator to interpolate field values at newly created vertices. Interpolated field values are stored just beyond the parametric coordinates associated with a vertex. They will always be double values; it does not make sense to interpolate a boolean or string value and your output and subdivision subroutines may always cast to a float or use floor() to truncate an interpolated value to an integer.

To create an instance of class vtkStreamingTessellator, simply invoke its constructor as follows

```
obj = vtkStreamingTessellator
```

33.211.2 Methods

The class vtkStreamingTessellator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStreamingTessellator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStreamingTessellator = obj.NewInstance ()`
- `vtkStreamingTessellator = obj.SafeDownCast (vtkObject o)`
- `obj.SetSubdivisionAlgorithm (vtkEdgeSubdivisionCriterion )` - Get/Set the algorithm used to determine whether an edge should be subdivided or left as-is. This is used once for each call to AdaptivelySample1Facet (which is recursive and will call itself resulting in additional edges to be checked) or three times for each call to AdaptivelySample2Facet (also recursive).
- `vtkEdgeSubdivisionCriterion = obj.GetSubdivisionAlgorithm ()` - Get/Set the algorithm used to determine whether an edge should be subdivided or left as-is. This is used once for each call to AdaptivelySample1Facet (which is recursive and will call itself resulting in additional edges to be checked) or three times for each call to AdaptivelySample2Facet (also recursive).
- `obj.SetEmbeddingDimension (int k, int d)` - Get/Set the number of parameter-space coordinates associated with each input and output point. The default is k for k -facets. You may specify a different dimension, d, for each type of k -facet to be processed. For example, SetEmbeddingDimension(2, 3) would associate r, s, and t coordinates with each input and output point generated by AdaptivelySample2Facet but does not say anything about input or output points generated by AdaptivelySample1Facet. Call SetEmbeddingDimension(-1, d) to specify the same dimension for all possible k values. d may not exceed 8, as that would be plain silly.
- `int = obj.GetEmbeddingDimension (int k) const` - Get/Set the number of parameter-space coordinates associated with each input and output point. The default is k for k -facets. You may specify

a different dimension, d, for each type of k -facet to be processed. For example, SetEmbeddingDimension(2, 3) would associate r, s, and t coordinates with each input and output point generated by AdaptivelySample2Facet but does not say anything about input or output points generated by AdaptivelySample1Facet. Call SetEmbeddingDimension(-1, d) to specify the same dimension for all possible k values. d may not exceed 8, as that would be plain silly.

- `obj.SetFieldSize (int k, int s)` - Get/Set the number of field value coordinates associated with each input and output point. The default is 0; no field values are interpolated. You may specify a different size, s, for each type of k -facet to be processed. For example, SetFieldSize(2, 3) would associate 3 field value coordinates with each input and output point of an AdaptivelySample2Facet call, but does not say anything about input or output points of AdaptivelySample1Facet. Call SetFieldSize(-1, s) to specify the same dimension for all possible k values. s may not exceed vtkStreamingTessellator::MaxFieldSize. This is a compile-time constant that defaults to 18, which is large enough for a scalar, vector, tensor, normal, and texture coordinate to be included at each point.

  Normally, you will not call SetFieldSize() directly; instead, subclasses of vtkEdgeSubdivisionCriterion, such as vtkShoeMeshSubdivisionAlgorithm, will call it for you.

  In any event, setting FieldSize to a non-zero value means you must pass field values to the AdaptivelySamplekFacet routines; For example, @verbatim vtkStreamingTessellator* t = vtkStreamingTessellator::New(); t-¿SetFieldSize(1, 3); t-¿SetEmbeddingDimension(1, 1); // not really required, this is the default double p0[3+1+3] = x0, y0, z0, r0, fx0, fy0, fz0 ; double p1[3+1+3] = x1, y1, z1, r1, fx1, fy1, fz1 ; t-¿AdaptivelySample1Facet(p0, p1); @endverbatim This would adaptively sample an curve (1-facet) with geometry and a vector field at every output point on the curve.

- `int = obj.GetFieldSize (int k) const` - Get/Set the number of field value coordinates associated with each input and output point. The default is 0; no field values are interpolated. You may specify a different size, s, for each type of k -facet to be processed. For example, SetFieldSize(2, 3) would associate 3 field value coordinates with each input and output point of an AdaptivelySample2Facet call, but does not say anything about input or output points of AdaptivelySample1Facet. Call SetFieldSize(-1, s) to specify the same dimension for all possible k values. s may not exceed vtkStreamingTessellator::MaxFieldSize. This is a compile-time constant that defaults to 18, which is large enough for a scalar, vector, tensor, normal, and texture coordinate to be included at each point.

  Normally, you will not call SetFieldSize() directly; instead, subclasses of vtkEdgeSubdivisionCriterion, such as vtkShoeMeshSubdivisionAlgorithm, will call it for you.

  In any event, setting FieldSize to a non-zero value means you must pass field values to the AdaptivelySamplekFacet routines; For example, @verbatim vtkStreamingTessellator* t = vtkStreamingTessellator::New(); t-¿SetFieldSize(1, 3); t-¿SetEmbeddingDimension(1, 1); // not really required, this is the default double p0[3+1+3] = x0, y0, z0, r0, fx0, fy0, fz0 ; double p1[3+1+3] = x1, y1, z1, r1, fx1, fy1, fz1 ; t-¿AdaptivelySample1Facet(p0, p1); @endverbatim This would adaptively sample an curve (1-facet) with geometry and a vector field at every output point on the curve.

- `obj.SetMaximumNumberOfSubdivisions (int num\_subdiv\_in)` - Get/Set the maximum number of subdivisions that may occur.

- `int = obj.GetMaximumNumberOfSubdivisions ()` - Get/Set the maximum number of subdivisions that may occur.

- `obj.AdaptivelySample3Facet (double v1, double v2, double v3, double v4) const` - This will adaptively subdivide the tetrahedron (3-facet), triangle (2-facet), or edge (1-facet) until the subdivision algorithm returns false for every edge or the maximum recursion depth is reached.

  Use SetMaximumNumberOfSubdivisions to change the maximum recursion depth.

  The AdaptivelySample0Facet method is provided as a convenience. Obviously, there is no way to adaptively subdivide a vertex. Instead the input vertex is passed unchanged to the output via a call to the registered VertexProcessorFunction callback.

.SECTION Warning This assumes that you have called SetSubdivisionAlgorithm(), SetEdgeCallback(), SetTriangleCallback(), and SetTetrahedronCallback() with valid values!

- `obj.AdaptivelySample2Facet (double v1, double v2, double v3) const` - This will adaptively subdivide the tetrahedron (3-facet), triangle (2-facet), or edge (1-facet) until the subdivision algorithm returns false for every edge or the maximum recursion depth is reached.

  Use SetMaximumNumberOfSubdivisions to change the maximum recursion depth.

  The AdaptivelySample0Facet method is provided as a convenience. Obviously, there is no way to adaptively subdivide a vertex. Instead the input vertex is passed unchanged to the output via a call to the registered VertexProcessorFunction callback.

  .SECTION Warning This assumes that you have called SetSubdivisionAlgorithm(), SetEdgeCallback(), SetTriangleCallback(), and SetTetrahedronCallback() with valid values!

- `obj.AdaptivelySample1Facet (double v1, double v2) const` - This will adaptively subdivide the tetrahedron (3-facet), triangle (2-facet), or edge (1-facet) until the subdivision algorithm returns false for every edge or the maximum recursion depth is reached.

  Use SetMaximumNumberOfSubdivisions to change the maximum recursion depth.

  The AdaptivelySample0Facet method is provided as a convenience. Obviously, there is no way to adaptively subdivide a vertex. Instead the input vertex is passed unchanged to the output via a call to the registered VertexProcessorFunction callback.

  .SECTION Warning This assumes that you have called SetSubdivisionAlgorithm(), SetEdgeCallback(), SetTriangleCallback(), and SetTetrahedronCallback() with valid values!

- `obj.AdaptivelySample0Facet (double v1) const` - This will adaptively subdivide the tetrahedron (3-facet), triangle (2-facet), or edge (1-facet) until the subdivision algorithm returns false for every edge or the maximum recursion depth is reached.

  Use SetMaximumNumberOfSubdivisions to change the maximum recursion depth.

  The AdaptivelySample0Facet method is provided as a convenience. Obviously, there is no way to adaptively subdivide a vertex. Instead the input vertex is passed unchanged to the output via a call to the registered VertexProcessorFunction callback.

  .SECTION Warning This assumes that you have called SetSubdivisionAlgorithm(), SetEdgeCallback(), SetTriangleCallback(), and SetTetrahedronCallback() with valid values!

- `obj.ResetCounts ()` - Reset/access the histogram of subdivision cases encountered. The histogram may be used to examine coverage during testing as well as characterizing the tessellation algorithm's performance. You should call ResetCounts() once, at the beginning of a stream of tetrahedra. It must be called before AdaptivelySample3Facet() to prevent uninitialized memory reads.

  These functions have no effect (and return 0) when PARAVIEW_DEBUG_TESSELLATOR has not been defined. By default, PARAVIEW_DEBUG_TESSELLATOR is not defined, and your code will be fast and efficient. Really!

- `vtkIdType = obj.GetCaseCount (int c)` - Reset/access the histogram of subdivision cases encountered. The histogram may be used to examine coverage during testing as well as characterizing the tessellation algorithm's performance. You should call ResetCounts() once, at the beginning of a stream of tetrahedra. It must be called before AdaptivelySample3Facet() to prevent uninitialized memory reads.

  These functions have no effect (and return 0) when PARAVIEW_DEBUG_TESSELLATOR has not been defined. By default, PARAVIEW_DEBUG_TESSELLATOR is not defined, and your code will be fast and efficient. Really!

- `vtkIdType = obj.GetSubcaseCount (int casenum, int sub)`

33.212 vtkStreamLine

33.212.1 Usage

vtkStreamLine is a filter that generates a streamline for an arbitrary dataset. A streamline is a line that is everywhere tangent to the vector field. Scalar values also are calculated along the streamline and can be used to color the line. Streamlines are calculated by integrating from a starting point through the vector field. Integration can be performed forward in time (see where the line goes), backward in time (see where the line came from), or in both directions. It also is possible to compute vorticity along the streamline. Vorticity is the projection (i.e., dot product) of the flow rotation on the velocity vector, i.e., the rotation of flow around the streamline.

vtkStreamLine defines the instance variable StepLength. This parameter controls the time increment used to generate individual points along the streamline(s). Smaller values result in more line primitives but smoother streamlines. The StepLength instance variable is defined in terms of time (i.e., the distance that the particle travels in the specified time period). Thus, the line segments will be smaller in areas of low velocity and larger in regions of high velocity. (NOTE: This is different than the IntegrationStepLength defined by the superclass vtkStreamer. IntegrationStepLength is used to control integration step size and is expressed as a fraction of the cell length.) The StepLength instance variable is important because subclasses of vtkStreamLine (e.g., vtkDashedStreamLine) depend on this value to build their representation.

To create an instance of class vtkStreamLine, simply invoke its constructor as follows

```
obj = vtkStreamLine
```

33.212.2 Methods

The class vtkStreamLine has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStreamLine class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStreamLine = obj.NewInstance ()`
- `vtkStreamLine = obj.SafeDownCast (vtkObject o)`
- `obj.SetStepLength (double )` - Specify the length of a line segment. The length is expressed in terms of elapsed time. Smaller values result in smoother appearing streamlines, but greater numbers of line primitives.
- `double = obj.GetStepLengthMinValue ()` - Specify the length of a line segment. The length is expressed in terms of elapsed time. Smaller values result in smoother appearing streamlines, but greater numbers of line primitives.
- `double = obj.GetStepLengthMaxValue ()` - Specify the length of a line segment. The length is expressed in terms of elapsed time. Smaller values result in smoother appearing streamlines, but greater numbers of line primitives.
- `double = obj.GetStepLength ()` - Specify the length of a line segment. The length is expressed in terms of elapsed time. Smaller values result in smoother appearing streamlines, but greater numbers of line primitives.

33.213 vtkStreamPoints

33.213.1 Usage

vtkStreamPoints is a filter that generates points along a streamer. The points are separated by a constant time increment. The resulting visual effect (especially when coupled with vtkGlyph3D) is an indication of particle speed.

To create an instance of class vtkStreamPoints, simply invoke its constructor as follows

```
obj = vtkStreamPoints
```

33.213.2 Methods

The class vtkStreamPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStreamPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStreamPoints = obj.NewInstance ()`
- `vtkStreamPoints = obj.SafeDownCast (vtkObject o)`
- `obj.SetTimeIncrement (double )` - Specify the separation of points in terms of absolute time.
- `double = obj.GetTimeIncrementMinValue ()` - Specify the separation of points in terms of absolute time.
- `double = obj.GetTimeIncrementMaxValue ()` - Specify the separation of points in terms of absolute time.
- `double = obj.GetTimeIncrement ()` - Specify the separation of points in terms of absolute time.

33.214 vtkStreamTracer

33.214.1 Usage

vtkStreamTracer is a filter that integrates a vector field to generate streamlines. The integration is performed using a specified integrator, by default Runge-Kutta2.

vtkStreamTracer produces polylines as the output, with each cell (i.e., polyline) representing a streamline. The attribute values associated with each streamline are stored in the cell data, whereas those associated with streamline-points are stored in the point data.

vtkStreamTracer supports forward (the default), backward, and combined (i.e., BOTH) integration. The length of a streamline is governed by specifying a maximum value either in physical arc length or in (local) cell length. Otherwise, the integration terminates upon exiting the flow field domain, or if the particle speed is reduced to a value less than a specified terminal speed, or when a maximum number of steps is completed. The specific reason for the termination is stored in a cell array named ReasonForTermination.

Note that normalized vectors are adopted in streamline integration, which achieves high numerical accuracy/smoothness of flow lines that is particularly guranteed for Runge-Kutta45 with adaptive step size and error control). In support of this feature, the underlying step size is ALWAYS in arc length unit (LENGTH_UNIT) while the 'real' time interval (virtual for steady flows) that a particle actually takes to trave in a single step is obtained by dividing the arc length by the LOCAL speed. The overall elapsed time (i.e., the life span) of the particle is the sum of those individual step-wise time intervals.

The quality of streamline integration can be controlled by setting the initial integration step (InitialIntegrationStep), particularly for Runge-Kutta2 and Runge-Kutta4 (with a fixed step size), and in the case of

Runge-Kutta45 (with an adaptive step size and error control) the minimum integration step, the maximum integration step, and the maximum error. These steps are in either LENGTH_UNIT or CELL_LENGTH_UNIT while the error is in physical arc length. For the former two integrators, there is a trade-off between integration speed and streamline quality.

The integration time, vorticity, rotation and angular velocity are stored in point data arrays named "IntegrationTime", "Vorticity", "Rotation" and "AngularVelocity", respectively (vorticity, rotation and angular velocity are computed only when ComputeVorticity is on). All point data attributes in the source dataset are interpolated on the new streamline points.

vtkStreamTracer supports integration through any type of dataset. Thus if the dataset contains 2D cells like polygons or triangles, the integration is constrained to lie on the surface defined by 2D cells.

The starting point, or the so-called 'seed', of a streamline may be set in two different ways. Starting from global x-y-z "position" allows you to start a single trace at a specified x-y-z coordinate. If you specify a source object, traces will be generated from each point in the source that is inside the dataset.

To create an instance of class vtkStreamTracer, simply invoke its constructor as follows

```
obj = vtkStreamTracer
```

33.214.2 Methods

The class vtkStreamTracer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStreamTracer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStreamTracer = obj.NewInstance ()`
- `vtkStreamTracer = obj.SafeDownCast (vtkObject o)`
- `obj.SetStartPosition (double , double , double )` - Specify the starting point (seed) of a streamline in the global coordinate system. Search must be performed to find the initial cell from which to start integration.
- `obj.SetStartPosition (double a[3])` - Specify the starting point (seed) of a streamline in the global coordinate system. Search must be performed to find the initial cell from which to start integration.
- `double = obj. GetStartPosition ()` - Specify the starting point (seed) of a streamline in the global coordinate system. Search must be performed to find the initial cell from which to start integration.
- `obj.SetSource (vtkDataSet source)` - Specify the source object used to generate starting points (seeds). Old style. Do not use.
- `vtkDataSet = obj.GetSource ()` - Specify the source object used to generate starting points (seeds). Old style. Do not use.
- `obj.SetSourceConnection (vtkAlgorithmOutput algOutput)` - Specify the source object used to generate starting points (seeds). New style.
- `obj.SetIntegrator (vtkInitialValueProblemSolver )` - Set/get the integrator type to be used for streamline generation. The object passed is not actually used but is cloned with NewInstance in the process of integration (prototype pattern). The default is Runge-Kutta2. The integrator can also be changed using SetIntegratorType. The recognized solvers are: RUNGE_KUTTA2 = 0 RUNGE_KUTTA4 = 1 RUNGE_KUTTA45 = 2

- `vtkInitialValueProblemSolver = obj.GetIntegrator ()` - Set/get the integrator type to be used for streamline generation. The object passed is not actually used but is cloned with NewInstance in the process of integration (prototype pattern). The default is Runge-Kutta2. The integrator can also be changed using SetIntegratorType. The recognized solvers are: RUNGE_KUTTA2 = 0 RUNGE_KUTTA4 = 1 RUNGE_KUTTA45 = 2

- `obj.SetIntegratorType (int type)` - Set/get the integrator type to be used for streamline generation. The object passed is not actually used but is cloned with NewInstance in the process of integration (prototype pattern). The default is Runge-Kutta2. The integrator can also be changed using SetIntegratorType. The recognized solvers are: RUNGE_KUTTA2 = 0 RUNGE_KUTTA4 = 1 RUNGE_KUTTA45 = 2

- `int = obj.GetIntegratorType ()` - Set/get the integrator type to be used for streamline generation. The object passed is not actually used but is cloned with NewInstance in the process of integration (prototype pattern). The default is Runge-Kutta2. The integrator can also be changed using SetIntegratorType. The recognized solvers are: RUNGE_KUTTA2 = 0 RUNGE_KUTTA4 = 1 RUNGE_KUTTA45 = 2

- `obj.SetIntegratorTypeToRungeKutta2 ()` - Set/get the integrator type to be used for streamline generation. The object passed is not actually used but is cloned with NewInstance in the process of integration (prototype pattern). The default is Runge-Kutta2. The integrator can also be changed using SetIntegratorType. The recognized solvers are: RUNGE_KUTTA2 = 0 RUNGE_KUTTA4 = 1 RUNGE_KUTTA45 = 2

- `obj.SetIntegratorTypeToRungeKutta4 ()` - Set/get the integrator type to be used for streamline generation. The object passed is not actually used but is cloned with NewInstance in the process of integration (prototype pattern). The default is Runge-Kutta2. The integrator can also be changed using SetIntegratorType. The recognized solvers are: RUNGE_KUTTA2 = 0 RUNGE_KUTTA4 = 1 RUNGE_KUTTA45 = 2

- `obj.SetIntegratorTypeToRungeKutta45 ()` - Set/get the integrator type to be used for streamline generation. The object passed is not actually used but is cloned with NewInstance in the process of integration (prototype pattern). The default is Runge-Kutta2. The integrator can also be changed using SetIntegratorType. The recognized solvers are: RUNGE_KUTTA2 = 0 RUNGE_KUTTA4 = 1 RUNGE_KUTTA45 = 2

- `obj.SetInterpolatorTypeToDataSetPointLocator ()` - Set the velocity field interpolator type to the one involving a dataset point locator.

- `obj.SetInterpolatorTypeToCellLocator ()` - Set the velocity field interpolator type to the one involving a cell locator.

- `obj.SetMaximumPropagation (double max)` - Specify the maximum length of a streamline expressed in LENGTH_UNIT.

- `double = obj.GetMaximumPropagation ()` - Specify a uniform integration step unit for MinimumIntegrationStep, InitialIntegrationStep, and MaximumIntegrationStep. NOTE: The valid unit is now limited to only LENGTH_UNIT (1) and CELL_LENGTH_UNIT (2), EXCLUDING the previously-supported TIME_UNIT.

- `obj.SetIntegrationStepUnit (int unit)` - Specify a uniform integration step unit for MinimumIntegrationStep, InitialIntegrationStep, and MaximumIntegrationStep. NOTE: The valid unit is now limited to only LENGTH_UNIT (1) and CELL_LENGTH_UNIT (2), EXCLUDING the previously-supported TIME_UNIT.

- `int = obj.GetIntegrationStepUnit ()` - Specify the Initial step size used for line integration, expressed in: LENGTH_UNIT = 1 CELL_LENGTH_UNIT = 2 (either the starting size for an adaptive integrator, e.g., RK45, or the constant / fixed size for non-adaptive ones, i.e., RK2 and RK4)

- `obj.SetInitialIntegrationStep (double step)` - Specify the Initial step size used for line integration, expressed in: LENGTH_UNIT = 1 CELL_LENGTH_UNIT = 2 (either the starting size for an adaptive integrator, e.g., RK45, or the constant / fixed size for non-adaptive ones, i.e., RK2 and RK4)
- `double = obj.GetInitialIntegrationStep ()` - Specify the Minimum step size used for line integration, expressed in: LENGTH_UNIT = 1 CELL_LENGTH_UNIT = 2 (Only valid for an adaptive integrator, e.g., RK45)
- `obj.SetMinimumIntegrationStep (double step)` - Specify the Minimum step size used for line integration, expressed in: LENGTH_UNIT = 1 CELL_LENGTH_UNIT = 2 (Only valid for an adaptive integrator, e.g., RK45)
- `double = obj.GetMinimumIntegrationStep ()` - Specify the Maximum step size used for line integration, expressed in: LENGTH_UNIT = 1 CELL_LENGTH_UNIT = 2 (Only valid for an adaptive integrator, e.g., RK45)
- `obj.SetMaximumIntegrationStep (double step)` - Specify the Maximum step size used for line integration, expressed in: LENGTH_UNIT = 1 CELL_LENGTH_UNIT = 2 (Only valid for an adaptive integrator, e.g., RK45)
- `double = obj.GetMaximumIntegrationStep ()`
- `obj.SetMaximumError (double )`
- `double = obj.GetMaximumError ()`
- `obj.SetMaximumNumberOfSteps (vtkIdType )`
- `vtkIdType = obj.GetMaximumNumberOfSteps ()`
- `obj.SetTerminalSpeed (double )`
- `double = obj.GetTerminalSpeed ()`
- `obj.SetIntegrationDirection (int )` - Specify whether the streamline is integrated in the upstream or downstream direction.
- `int = obj.GetIntegrationDirectionMinValue ()` - Specify whether the streamline is integrated in the upstream or downstream direction.
- `int = obj.GetIntegrationDirectionMaxValue ()` - Specify whether the streamline is integrated in the upstream or downstream direction.
- `int = obj.GetIntegrationDirection ()` - Specify whether the streamline is integrated in the upstream or downstream direction.
- `obj.SetIntegrationDirectionToForward ()` - Specify whether the streamline is integrated in the upstream or downstream direction.
- `obj.SetIntegrationDirectionToBackward ()` - Specify whether the streamline is integrated in the upstream or downstream direction.
- `obj.SetIntegrationDirectionToBoth ()` - Specify whether the streamline is integrated in the upstream or downstream direction.
- `obj.SetComputeVorticity (bool )`
- `bool = obj.GetComputeVorticity ()`
- `obj.SetRotationScale (double )`
- `double = obj.GetRotationScale ()`

- `obj.SetInterpolatorPrototype (vtkAbstractInterpolatedVelocityField ivf)` - The object used to interpolate the velocity field during integration is of the same class as this prototype.
- `obj.SetInterpolatorType (int interpType)` - Set the type of the velocity field interpolator to determine whether vtkInterpolatedVelocityField (INTERPOLATOR_WITH_DATASET_POINT_LOCATOR) or vtkCellLocatorInterpolatedVelocityField (INTERPOLATOR_WITH_CELL_LOCATOR) is employed for locating cells during streamline integration. The latter (adopting vtkAbstractCellLocator subclasses such as vtkCellLocator and vtkModifiedBSPTree) is more robust then the former (through vtkDataSet / vtkPointSet::FindCell() coupled with vtkPointLocator).

33.215 vtkStripper

33.215.1 Usage

To create an instance of class vtkStripper, simply invoke its constructor as follows

```
obj = vtkStripper
```

33.215.2 Methods

The class vtkStripper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStripper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStripper = obj.NewInstance ()`
- `vtkStripper = obj.SafeDownCast (vtkObject o)`
- `obj.SetMaximumLength (int )` - Specify the maximum number of triangles in a triangle strip, and/or the maximum number of lines in a poly-line.
- `int = obj.GetMaximumLengthMinValue ()` - Specify the maximum number of triangles in a triangle strip, and/or the maximum number of lines in a poly-line.
- `int = obj.GetMaximumLengthMaxValue ()` - Specify the maximum number of triangles in a triangle strip, and/or the maximum number of lines in a poly-line.
- `int = obj.GetMaximumLength ()` - Specify the maximum number of triangles in a triangle strip, and/or the maximum number of lines in a poly-line.
- `obj.PassCellDataAsFieldDataOn ()` - Enable/Disable passing of the CellData in the input to the output as FieldData. Note the field data is tranformed.
- `obj.PassCellDataAsFieldDataOff ()` - Enable/Disable passing of the CellData in the input to the output as FieldData. Note the field data is tranformed.
- `obj.SetPassCellDataAsFieldData (int )` - Enable/Disable passing of the CellData in the input to the output as FieldData. Note the field data is tranformed.
- `int = obj.GetPassCellDataAsFieldData ()` - Enable/Disable passing of the CellData in the input to the output as FieldData. Note the field data is tranformed.
- `obj.SetPassThroughCellIds (int )` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for picking. The default is off to conserve memory.

- `int = obj.GetPassThroughCellIds ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for picking. The default is off to conserve memory.
- `obj.PassThroughCellIdsOn ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for picking. The default is off to conserve memory.
- `obj.PassThroughCellIdsOff ()` - If on, the output polygonal dataset will have a celldata array that holds the cell index of the original 3D cell that produced each output cell. This is useful for picking. The default is off to conserve memory.
- `obj.SetPassThroughPointIds (int )` - If on, the output polygonal dataset will have a pointdata array that holds the point index of the original vertex that produced each output vertex. This is useful for picking. The default is off to conserve memory.
- `int = obj.GetPassThroughPointIds ()` - If on, the output polygonal dataset will have a pointdata array that holds the point index of the original vertex that produced each output vertex. This is useful for picking. The default is off to conserve memory.
- `obj.PassThroughPointIdsOn ()` - If on, the output polygonal dataset will have a pointdata array that holds the point index of the original vertex that produced each output vertex. This is useful for picking. The default is off to conserve memory.
- `obj.PassThroughPointIdsOff ()` - If on, the output polygonal dataset will have a pointdata array that holds the point index of the original vertex that produced each output vertex. This is useful for picking. The default is off to conserve memory.

33.216 vtkStructuredGridClip

33.216.1 Usage

vtkStructuredGridClip will make an image smaller. The output must have an image extent which is the subset of the input. The filter has two modes of operation: 1: By default, the data is not copied in this filter. Only the whole extent is modified. 2: If ClipDataOn is set, then you will get no more that the clipped extent.

To create an instance of class vtkStructuredGridClip, simply invoke its constructor as follows

```
obj = vtkStructuredGridClip
```

33.216.2 Methods

The class vtkStructuredGridClip has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGridClip class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredGridClip = obj.NewInstance ()`
- `vtkStructuredGridClip = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutputWholeExtent (int extent[6], vtkInformation outInfo)` - The whole extent of the output has to be set explicitly.

- `obj.SetOutputWholeExtent (int minX, int maxX, int minY, int maxY, int minZ, int maxZ)` - The whole extent of the output has to be set explicitly.
- `obj.GetOutputWholeExtent (int extent[6])` - The whole extent of the output has to be set explicitly.
- `obj.ResetOutputWholeExtent ()`
- `obj.SetClipData (int )` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `int = obj.GetClipData ()` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `obj.ClipDataOn ()` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `obj.ClipDataOff ()` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `obj.SetOutputWholeExtent (int piece, int numPieces)` - Hack set output by piece

33.217 vtkStructuredGridGeometryFilter

33.217.1 Usage

vtkStructuredGridGeometryFilter is a filter that extracts geometry from a structured grid. By specifying appropriate i-j-k indices, it is possible to extract a point, a curve, a surface, or a "volume". Depending upon the type of data, the curve and surface may be curved or planar. (The volume is actually a (n x m x o) region of points.)

The extent specification is zero-offset. That is, the first k-plane in a 50x50x50 structured grid is given by (0,49, 0,49, 0,0).

The output of this filter is affected by the structured grid blanking. If blanking is on, and a blanking array defined, then those cells attached to blanked points are not output. (Blanking is a property of the input vtkStructuredGrid.)

To create an instance of class vtkStructuredGridGeometryFilter, simply invoke its constructor as follows

```
obj = vtkStructuredGridGeometryFilter
```

33.217.2 Methods

The class vtkStructuredGridGeometryFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGridGeometryFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredGridGeometryFilter = obj.NewInstance ()`
- `vtkStructuredGridGeometryFilter = obj.SafeDownCast (vtkObject o)`

- `int = obj. GetExtent ()` - Get the extent in topological coordinate range (imin,imax, jmin,jmax, kmin,kmax).
- `obj.SetExtent (int iMin, int iMax, int jMin, int jMax, int kMin, int kMax)` - Specify (imin,imax, jmin,jmax, kmin,kmax) indices.
- `obj.SetExtent (int extent[6])` - Specify (imin,imax, jmin,jmax, kmin,kmax) indices in array form.

33.218 vtkStructuredGridOutlineFilter

33.218.1 Usage

vtkStructuredGridOutlineFilter is a filter that generates a wireframe outline of a structured grid (vtkStructuredGrid). Structured data is topologically a cube, so the outline will have 12 "edges".

To create an instance of class vtkStructuredGridOutlineFilter, simply invoke its constructor as follows

```
obj = vtkStructuredGridOutlineFilter
```

33.218.2 Methods

The class vtkStructuredGridOutlineFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGridOutlineFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredGridOutlineFilter = obj.NewInstance ()`
- `vtkStructuredGridOutlineFilter = obj.SafeDownCast (vtkObject o)`

33.219 vtkStructuredPointsGeometryFilter

33.219.1 Usage

vtkStructuredPointsGeometryFilter has been renamed to vtkImageDataGeometryFilter

To create an instance of class vtkStructuredPointsGeometryFilter, simply invoke its constructor as follows

```
obj = vtkStructuredPointsGeometryFilter
```

33.219.2 Methods

The class vtkStructuredPointsGeometryFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredPointsGeometryFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredPointsGeometryFilter = obj.NewInstance ()`
- `vtkStructuredPointsGeometryFilter = obj.SafeDownCast (vtkObject o)`

33.220 vtkSubdivideTetra

33.220.1 Usage

This filter subdivides tetrahedra in an unstructured grid into twelve tetrahedra.

To create an instance of class vtkSubdivideTetra, simply invoke its constructor as follows

```
obj = vtkSubdivideTetra
```

33.220.2 Methods

The class vtkSubdivideTetra has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSubdivideTetra class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSubdivideTetra = obj.NewInstance ()`
- `vtkSubdivideTetra = obj.SafeDownCast (vtkObject o)`

33.221 vtkSubPixelPositionEdgels

33.221.1 Usage

vtkSubPixelPositionEdgels is a filter that takes a series of linked edgels (digital curves) and gradient maps as input. It then adjusts the edgel locations based on the gradient data. Specifically, the algorithm first determines the neighboring gradient magnitudes of an edgel using simple interpolation of its neighbors. It then fits the following three data points: negative gradient direction gradient magnitude, edgel gradient magnitude and positive gradient direction gradient magnitude to a quadratic function. It then solves this quadratic to find the maximum gradient location along the gradient orientation. It then modifies the edgels location along the gradient orientation to the calculated maximum location. This algorithm does not adjust an edgel in the direction orthogonal to its gradient vector.

To create an instance of class vtkSubPixelPositionEdgels, simply invoke its constructor as follows

```
obj = vtkSubPixelPositionEdgels
```

33.221.2 Methods

The class vtkSubPixelPositionEdgels has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSubPixelPositionEdgels class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSubPixelPositionEdgels = obj.NewInstance ()`
- `vtkSubPixelPositionEdgels = obj.SafeDownCast (vtkObject o)`
- `obj.SetGradMaps (vtkStructuredPoints gm)` - Set/Get the gradient data for doing the position adjustments.

- `vtkStructuredPoints = obj.GetGradMaps ()` - Set/Get the gradient data for doing the position adjustments.
- `obj.SetTargetFlag (int )` - These methods can make the positioning look for a target scalar value instead of looking for a maximum.
- `int = obj.GetTargetFlag ()` - These methods can make the positioning look for a target scalar value instead of looking for a maximum.
- `obj.TargetFlagOn ()` - These methods can make the positioning look for a target scalar value instead of looking for a maximum.
- `obj.TargetFlagOff ()` - These methods can make the positioning look for a target scalar value instead of looking for a maximum.
- `obj.SetTargetValue (double )` - These methods can make the positioning look for a target scalar value instead of looking for a maximum.
- `double = obj.GetTargetValue ()` - These methods can make the positioning look for a target scalar value instead of looking for a maximum.

33.222 vtkSuperquadricSource

33.222.1 Usage

vtkSuperquadricSource creates a superquadric (represented by polygons) of specified size centered at the origin. The resolution (polygonal discretization) in both the latitude (phi) and longitude (theta) directions can be specified. Roundness parameters (PhiRoundness and ThetaRoundness) control the shape of the superquadric. The Toroidal boolean controls whether a toroidal superquadric is produced. If so, the Thickness parameter controls the thickness of the toroid: 0 is the thinnest allowable toroid, and 1 has a minimum sized hole. The Scale parameters allow the superquadric to be scaled in x, y, and z (normal vectors are correctly generated in any case). The Size parameter controls size of the superquadric.

This code is based on "Rigid physically based superquadrics", A. H. Barr, in "Graphics Gems III", David Kirk, ed., Academic Press, 1992.

To create an instance of class vtkSuperquadricSource, simply invoke its constructor as follows

```
obj = vtkSuperquadricSource
```

33.222.2 Methods

The class vtkSuperquadricSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSuperquadricSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSuperquadricSource = obj.NewInstance ()`
- `vtkSuperquadricSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetCenter (double , double , double )` - Set the center of the superquadric. Default is 0,0,0.
- `obj.SetCenter (double  a[3])` - Set the center of the superquadric. Default is 0,0,0.
- `double = obj. GetCenter ()` - Set the center of the superquadric. Default is 0,0,0.

- `obj.SetScale (double , double , double )` - Set the scale factors of the superquadric. Default is 1,1,1.
- `obj.SetScale (double a[3])` - Set the scale factors of the superquadric. Default is 1,1,1.
- `double = obj. GetScale ()` - Set the scale factors of the superquadric. Default is 1,1,1.
- `int = obj.GetThetaResolution ()` - Set the number of points in the longitude direction. Initial value is 16.
- `obj.SetThetaResolution (int i)` - Set the number of points in the longitude direction. Initial value is 16.
- `int = obj.GetPhiResolution ()` - Set the number of points in the latitude direction. Initial value is 16.
- `obj.SetPhiResolution (int i)` - Set the number of points in the latitude direction. Initial value is 16.
- `double = obj.GetThickness ()` - Set/Get Superquadric ring thickness (toroids only). Changing thickness maintains the outside diameter of the toroid. Initial value is 0.3333.
- `obj.SetThickness (double )` - Set/Get Superquadric ring thickness (toroids only). Changing thickness maintains the outside diameter of the toroid. Initial value is 0.3333.
- `double = obj.GetThicknessMinValue ()` - Set/Get Superquadric ring thickness (toroids only). Changing thickness maintains the outside diameter of the toroid. Initial value is 0.3333.
- `double = obj.GetThicknessMaxValue ()` - Set/Get Superquadric ring thickness (toroids only). Changing thickness maintains the outside diameter of the toroid. Initial value is 0.3333.
- `double = obj.GetPhiRoundness ()` - Set/Get Superquadric north/south roundness. Values range from 0 (rectangular) to 1 (circular) to higher orders. Initial value is 1.0.
- `obj.SetPhiRoundness (double e)` - Set/Get Superquadric north/south roundness. Values range from 0 (rectangular) to 1 (circular) to higher orders. Initial value is 1.0.
- `double = obj.GetThetaRoundness ()` - Set/Get Superquadric east/west roundness. Values range from 0 (rectangular) to 1 (circular) to higher orders. Initial value is 1.0.
- `obj.SetThetaRoundness (double e)` - Set/Get Superquadric east/west roundness. Values range from 0 (rectangular) to 1 (circular) to higher orders. Initial value is 1.0.
- `obj.SetSize (double )` - Set/Get Superquadric isotropic size. Initial value is 0.5;
- `double = obj.GetSize ()` - Set/Get Superquadric isotropic size. Initial value is 0.5;
- `obj.ToroidalOn ()` - Set/Get whether or not the superquadric is toroidal (1) or ellipsoidal (0). Initial value is 0.
- `obj.ToroidalOff ()` - Set/Get whether or not the superquadric is toroidal (1) or ellipsoidal (0). Initial value is 0.
- `int = obj.GetToroidal ()` - Set/Get whether or not the superquadric is toroidal (1) or ellipsoidal (0). Initial value is 0.
- `obj.SetToroidal (int )` - Set/Get whether or not the superquadric is toroidal (1) or ellipsoidal (0). Initial value is 0.

33.223 vtkSynchronizedTemplates2D

33.223.1 Usage

vtkSynchronizedTemplates2D is a 2D implementation of the synchronized template algorithm. Note that vtkContourFilter will automatically use this class when appropriate.

To create an instance of class vtkSynchronizedTemplates2D, simply invoke its constructor as follows

```
obj = vtkSynchronizedTemplates2D
```

33.223.2 Methods

The class vtkSynchronizedTemplates2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSynchronizedTemplates2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSynchronizedTemplates2D = obj.NewInstance ()`
- `vtkSynchronizedTemplates2D = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Because we delegate to vtkContourValues
- `obj.SetValue (int i, double value)` - Get the ith contour value.
- `double = obj.GetValue (int i)` - Get a pointer to an array of contour values. There will be GetNumberOfContours() values in the list.
- `obj.GetValues (double contourValues)` - Set the number of contours to place into the list. You only really need to use this method to reduce list size. The method SetValue() will automatically increase list size as needed.
- `obj.SetNumberOfContours (int number)` - Get the number of contours in the list of contour values.
- `int = obj.GetNumberOfContours ()` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double range[2])` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Option to set the point scalars of the output. The scalars will be the iso value of course. By default this flag is on.
- `obj.SetComputeScalars (int )` - Option to set the point scalars of the output. The scalars will be the iso value of course. By default this flag is on.
- `int = obj.GetComputeScalars ()` - Option to set the point scalars of the output. The scalars will be the iso value of course. By default this flag is on.
- `obj.ComputeScalarsOn ()` - Option to set the point scalars of the output. The scalars will be the iso value of course. By default this flag is on.
- `obj.ComputeScalarsOff ()` - Option to set the point scalars of the output. The scalars will be the iso value of course. By default this flag is on.
- `obj.SetArrayComponent (int )` - Set/get which component of the scalar array to contour on; defaults to 0.
- `int = obj.GetArrayComponent ()` - Set/get which component of the scalar array to contour on; defaults to 0.

33.224 vtkSynchronizedTemplates3D

33.224.1 Usage

vtkSynchronizedTemplates3D is a 3D implementation of the synchronized template algorithm. Note that vtkContourFilter will automatically use this class when appropriate.

To create an instance of class vtkSynchronizedTemplates3D, simply invoke its constructor as follows

```
obj = vtkSynchronizedTemplates3D
```

33.224.2 Methods

The class vtkSynchronizedTemplates3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSynchronizedTemplates3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSynchronizedTemplates3D = obj.NewInstance ()`
- `vtkSynchronizedTemplates3D = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Because we delegate to vtkContourValues
- `obj.SetComputeNormals (int )` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeNormals ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOn ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeNormalsOff ()` - Set/Get the computation of normals. Normal computation is fairly expensive in both time and storage. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeGradients (int )` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `int = obj.GetComputeGradients ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.ComputeGradientsOn ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.

- `obj.ComputeGradientsOff ()` - Set/Get the computation of gradients. Gradient computation is fairly expensive in both time and storage. Note that if ComputeNormals is on, gradients will have to be calculated, but will not be stored in the output dataset. If the output data will be processed by filters that modify topology or geometry, it may be wise to turn Normals and Gradients off.
- `obj.SetComputeScalars (int )` - Set/Get the computation of scalars.
- `int = obj.GetComputeScalars ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOn ()` - Set/Get the computation of scalars.
- `obj.ComputeScalarsOff ()` - Set/Get the computation of scalars.
- `obj.SetValue (int i, double value)` - Get the ith contour value.
- `double = obj.GetValue (int i)` - Get a pointer to an array of contour values. There will be GetNumberOfContours() values in the list.
- `obj.GetValues (double contourValues)` - Set the number of contours to place into the list. You only really need to use this method to reduce list size. The method SetValue() will automatically increase list size as needed.
- `obj.SetNumberOfContours (int number)` - Get the number of contours in the list of contour values.
- `int = obj.GetNumberOfContours ()` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double range[2])` - Generate numContours equally spaced contour values between specified range. Contour values will include min/max range values.
- `obj.GenerateValues (int numContours, double rangeStart, double rangeEnd)` - Needed by templated functions.
- `int = obj.GetExecuteExtent ()` - Needed by templated functions.
- `obj.ThreadedExecute (vtkImageData data, vtkInformation inInfo, vtkInformation outInfo, int exExt, v` - Needed by templated functions.
- `obj.SetInputMemoryLimit (long limit)` - Determines the chunk size fro streaming. This filter will act like a collector: ask for many input pieces, but generate one output. Limit is in KBytes
- `long = obj.GetInputMemoryLimit ()` - Determines the chunk size fro streaming. This filter will act like a collector: ask for many input pieces, but generate one output. Limit is in KBytes
- `obj.SetArrayComponent (int )` - Set/get which component of the scalar array to contour on; defaults to 0.
- `int = obj.GetArrayComponent ()` - Set/get which component of the scalar array to contour on; defaults to 0.

33.225 vtkSynchronizedTemplatesCutter3D

33.225.1 Usage

vtkSynchronizedTemplatesCutter3D is an implementation of the synchronized template algorithm. Note that vtkCutFilter will automatically use this class when appropriate.

To create an instance of class vtkSynchronizedTemplatesCutter3D, simply invoke its constructor as follows

```
obj = vtkSynchronizedTemplatesCutter3D
```

33.225.2 Methods

The class vtkSynchronizedTemplatesCutter3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSynchronizedTemplatesCutter3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSynchronizedTemplatesCutter3D = obj.NewInstance ()`
- `vtkSynchronizedTemplatesCutter3D = obj.SafeDownCast (vtkObject o)`
- `obj.ThreadedExecute (vtkImageData data, vtkInformation outInfo, int exExt, int )` - Needed by templated functions.
- `obj.SetCutFunction (vtkImplicitFunction )`
- `vtkImplicitFunction = obj.GetCutFunction ()`

33.226 vtkTableBasedClipDataSet

33.226.1 Usage

vtkTableBasedClipDataSet is a filter that clips any type of dataset using either any subclass of vtkImplicitFunction or an input scalar point data array. Clipping means that it actually "cuts" through the cells of the dataset, returning everything outside the specified implicit function (or greater than the scalar value) including "pieces" of a cell (Note to compare this with vtkExtractGeometry, which pulls out entire, uncut cells). The output of this filter is a vtkUnstructuredGrid data.

To use this filter, you need to decide whether an implicit function or an input scalar point data array is used for clipping. For the former case, 1) define an implicit function 2) provide it to this filter via SetClipFunction() If a clipping function is not specified, or GenerateClipScalars is off(the default), the input scalar point data array is then employed for clipping.

You can also specify a scalar (iso-)value, which is used to decide what is inside and outside the implicit function. You can also reverse the sense of what inside/outside is by setting IVAR InsideOut. The clipping algorithm proceeds by computing an implicit function value or using the input scalar point data value for each point in the dataset. This is compared against the scalar (iso-)value to determine the inside/outside status.

Although this filter sometimes (but rarely) may resort to the sibling class vtkClipDataSet for handling some special grids (such as cylinders or cones with capping faces in the form of a vtkPolyData), it itself is able to deal with most grids. It is worth mentioning that vtkTableBasedClipDataSet is capable of addressing the artifacts that may occur with vtkClipDataSet due to the possibly inconsistent triagulation modes between neighboring cells. In addition, the former is much faster than the latter. Furthermore, the former produces less cells (with ratio usually being 5 6) than by the latter in the output. In other words, this filter retains the original cells (i.e., without triangulation / tetrahedralization) wherever possible. All these advantages are gained by adopting the unique clipping and triangulation tables proposed by VisIt.

To create an instance of class vtkTableBasedClipDataSet, simply invoke its constructor as follows

```
obj = vtkTableBasedClipDataSet
```

33.226.2 Methods

The class vtkTableBasedClipDataSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableBasedClipDataSet class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableBasedClipDataSet = obj.NewInstance ()`
- `vtkTableBasedClipDataSet = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Get the MTime for which the point locator and clip function are consdiered.
- `obj.SetInsideOut (int )` - Set/Get the InsideOut flag. With this flag off, a vertex is considered inside (the implicit function or the isosurface) if the (function or scalar) value is greater than IVAR Value. With this flag on, a vertex is considered inside (the implicit function or the isosurface) if the (function or scalar) value is less than or equal to IVAR Value. This flag is off by default.
- `int = obj.GetInsideOut ()` - Set/Get the InsideOut flag. With this flag off, a vertex is considered inside (the implicit function or the isosurface) if the (function or scalar) value is greater than IVAR Value. With this flag on, a vertex is considered inside (the implicit function or the isosurface) if the (function or scalar) value is less than or equal to IVAR Value. This flag is off by default.
- `obj.InsideOutOn ()` - Set/Get the InsideOut flag. With this flag off, a vertex is considered inside (the implicit function or the isosurface) if the (function or scalar) value is greater than IVAR Value. With this flag on, a vertex is considered inside (the implicit function or the isosurface) if the (function or scalar) value is less than or equal to IVAR Value. This flag is off by default.
- `obj.InsideOutOff ()` - Set/Get the InsideOut flag. With this flag off, a vertex is considered inside (the implicit function or the isosurface) if the (function or scalar) value is greater than IVAR Value. With this flag on, a vertex is considered inside (the implicit function or the isosurface) if the (function or scalar) value is less than or equal to IVAR Value. This flag is off by default.
- `obj.SetValue (double )` - Set/Get the clipping value of the implicit function (if an implicit function is applied) or scalar data array (if a scalar data array is used), with 0.0 as the default value. This value is ignored if flag UseValueAsOffset is true AND a clip function is defined.
- `double = obj.GetValue ()` - Set/Get the clipping value of the implicit function (if an implicit function is applied) or scalar data array (if a scalar data array is used), with 0.0 as the default value. This value is ignored if flag UseValueAsOffset is true AND a clip function is defined.
- `obj.SetUseValueAsOffset (bool )` - Set/Get flag UseValueAsOffset, with true as the default value. With this flag on, IVAR Value is used as an offset parameter to the implicit function. Value is used only when clipping using a scalar array.
- `bool = obj.GetUseValueAsOffset ()` - Set/Get flag UseValueAsOffset, with true as the default value. With this flag on, IVAR Value is used as an offset parameter to the implicit function. Value is used only when clipping using a scalar array.
- `obj.UseValueAsOffsetOn ()` - Set/Get flag UseValueAsOffset, with true as the default value. With this flag on, IVAR Value is used as an offset parameter to the implicit function. Value is used only when clipping using a scalar array.
- `obj.UseValueAsOffsetOff ()` - Set/Get flag UseValueAsOffset, with true as the default value. With this flag on, IVAR Value is used as an offset parameter to the implicit function. Value is used only when clipping using a scalar array.
- `obj.SetClipFunction (vtkImplicitFunction )`
- `vtkImplicitFunction = obj.GetClipFunction ()`

- `obj.SetGenerateClipScalars (int )` - Set/Get flag GenerateClipScalars, with 0 as the default value. With this flag on, the scalar point data values obtained by evaluating the implicit function will be exported to the output. Note that this flag requries that an implicit function be provided.
- `int = obj.GetGenerateClipScalars ()` - Set/Get flag GenerateClipScalars, with 0 as the default value. With this flag on, the scalar point data values obtained by evaluating the implicit function will be exported to the output. Note that this flag requries that an implicit function be provided.
- `obj.GenerateClipScalarsOn ()` - Set/Get flag GenerateClipScalars, with 0 as the default value. With this flag on, the scalar point data values obtained by evaluating the implicit function will be exported to the output. Note that this flag requries that an implicit function be provided.
- `obj.GenerateClipScalarsOff ()` - Set/Get flag GenerateClipScalars, with 0 as the default value. With this flag on, the scalar point data values obtained by evaluating the implicit function will be exported to the output. Note that this flag requries that an implicit function be provided.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set/Get a point locator locator for merging duplicate points. By default, an instance of vtkMergePoints is used. Note that this IVAR is provided in this class only because this filter may resort to its sibling class vtkClipDataSet when processing some special grids (such as cylinders or cones with capping faces in the form of a vtkPolyData) while the latter requires a point locator. This filter itself does not need a locator.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set/Get a point locator locator for merging duplicate points. By default, an instance of vtkMergePoints is used. Note that this IVAR is provided in this class only because this filter may resort to its sibling class vtkClipDataSet when processing some special grids (such as cylinders or cones with capping faces in the form of a vtkPolyData) while the latter requires a point locator. This filter itself does not need a locator.
- `obj.SetMergeTolerance (double )` - Set/Get the tolerance used for merging duplicate points near the clipping intersection cells. This tolerance may prevent the generation of degenerate primitives. Note that only 3D cells actually use this IVAR.
- `double = obj.GetMergeToleranceMinValue ()` - Set/Get the tolerance used for merging duplicate points near the clipping intersection cells. This tolerance may prevent the generation of degenerate primitives. Note that only 3D cells actually use this IVAR.
- `double = obj.GetMergeToleranceMaxValue ()` - Set/Get the tolerance used for merging duplicate points near the clipping intersection cells. This tolerance may prevent the generation of degenerate primitives. Note that only 3D cells actually use this IVAR.
- `double = obj.GetMergeTolerance ()` - Set/Get the tolerance used for merging duplicate points near the clipping intersection cells. This tolerance may prevent the generation of degenerate primitives. Note that only 3D cells actually use this IVAR.
- `obj.CreateDefaultLocator ()` - Create a default point locator when none is specified. The point locator is used to merge coincident points.
- `obj.SetGenerateClippedOutput (int )` - Set/Get whether a second output is generated. The second output contains the polygonal data that is clipped away by the iso-surface.
- `int = obj.GetGenerateClippedOutput ()` - Set/Get whether a second output is generated. The second output contains the polygonal data that is clipped away by the iso-surface.
- `obj.GenerateClippedOutputOn ()` - Set/Get whether a second output is generated. The second output contains the polygonal data that is clipped away by the iso-surface.
- `obj.GenerateClippedOutputOff ()` - Set/Get whether a second output is generated. The second output contains the polygonal data that is clipped away by the iso-surface.
- `vtkUnstructuredGrid = obj.GetClippedOutput ()` - Return the clipped output.

33.227 vtkTableToPolyData

33.227.1 Usage

vtkTableToPolyData is a filter used to convert a vtkTable to a vtkPolyData consisting of vertices.

To create an instance of class vtkTableToPolyData, simply invoke its constructor as follows

```
obj = vtkTableToPolyData
```

33.227.2 Methods

The class vtkTableToPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableToPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableToPolyData = obj.NewInstance ()`
- `vtkTableToPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetXColumn (string )` - Set the name of the column to use as the X coordinate for the points.
- `string = obj.GetXColumn ()` - Set the name of the column to use as the X coordinate for the points.
- `obj.SetXColumnIndex (int )` - Set the index of the column to use as the X coordinate for the points.
- `int = obj.GetXColumnIndexMinValue ()` - Set the index of the column to use as the X coordinate for the points.
- `int = obj.GetXColumnIndexMaxValue ()` - Set the index of the column to use as the X coordinate for the points.
- `int = obj.GetXColumnIndex ()` - Set the index of the column to use as the X coordinate for the points.
- `obj.SetXComponent (int )` - Specify the component for the column specified using SetXColumn() to use as the xcoordinate in case the column is a multi-component array. Default is 0.
- `int = obj.GetXComponentMinValue ()` - Specify the component for the column specified using SetXColumn() to use as the xcoordinate in case the column is a multi-component array. Default is 0.
- `int = obj.GetXComponentMaxValue ()` - Specify the component for the column specified using SetXColumn() to use as the xcoordinate in case the column is a multi-component array. Default is 0.
- `int = obj.GetXComponent ()` - Specify the component for the column specified using SetXColumn() to use as the xcoordinate in case the column is a multi-component array. Default is 0.
- `obj.SetYColumn (string )` - Set the name of the column to use as the Y coordinate for the points. Default is 0.
- `string = obj.GetYColumn ()` - Set the name of the column to use as the Y coordinate for the points. Default is 0.
- `obj.SetYColumnIndex (int )` - Set the index of the column to use as the Y coordinate for the points.
- `int = obj.GetYColumnIndexMinValue ()` - Set the index of the column to use as the Y coordinate for the points.

- `int = obj.GetYColumnIndexMaxValue ()` - Set the index of the column to use as the Y coordinate for the points.
- `int = obj.GetYColumnIndex ()` - Set the index of the column to use as the Y coordinate for the points.
- `obj.SetYComponent (int )` - Specify the component for the column specified using SetYColumn() to use as the Ycoordinate in case the column is a multi-component array.
- `int = obj.GetYComponentMinValue ()` - Specify the component for the column specified using SetYColumn() to use as the Ycoordinate in case the column is a multi-component array.
- `int = obj.GetYComponentMaxValue ()` - Specify the component for the column specified using SetYColumn() to use as the Ycoordinate in case the column is a multi-component array.
- `int = obj.GetYComponent ()` - Specify the component for the column specified using SetYColumn() to use as the Ycoordinate in case the column is a multi-component array.
- `obj.SetZColumn (string )` - Set the name of the column to use as the Z coordinate for the points. Default is 0.
- `string = obj.GetZColumn ()` - Set the name of the column to use as the Z coordinate for the points. Default is 0.
- `obj.SetZColumnIndex (int )` - Set the index of the column to use as the Z coordinate for the points.
- `int = obj.GetZColumnIndexMinValue ()` - Set the index of the column to use as the Z coordinate for the points.
- `int = obj.GetZColumnIndexMaxValue ()` - Set the index of the column to use as the Z coordinate for the points.
- `int = obj.GetZColumnIndex ()` - Set the index of the column to use as the Z coordinate for the points.
- `obj.SetZComponent (int )` - Specify the component for the column specified using SetZColumn() to use as the Zcoordinate in case the column is a multi-component array.
- `int = obj.GetZComponentMinValue ()` - Specify the component for the column specified using SetZColumn() to use as the Zcoordinate in case the column is a multi-component array.
- `int = obj.GetZComponentMaxValue ()` - Specify the component for the column specified using SetZColumn() to use as the Zcoordinate in case the column is a multi-component array.
- `int = obj.GetZComponent ()` - Specify the component for the column specified using SetZColumn() to use as the Zcoordinate in case the column is a multi-component array.
- `obj.SetCreate2DPoints (bool )` - Specify whether the points of the polydata are 3D or 2D. If this is set to true then the Z Column will be ignored and the z value of each point on the polydata will be set to 0. By default this will be off.
- `bool = obj.GetCreate2DPoints ()` - Specify whether the points of the polydata are 3D or 2D. If this is set to true then the Z Column will be ignored and the z value of each point on the polydata will be set to 0. By default this will be off.
- `obj.Create2DPointsOn ()` - Specify whether the points of the polydata are 3D or 2D. If this is set to true then the Z Column will be ignored and the z value of each point on the polydata will be set to 0. By default this will be off.
- `obj.Create2DPointsOff ()` - Specify whether the points of the polydata are 3D or 2D. If this is set to true then the Z Column will be ignored and the z value of each point on the polydata will be set to 0. By default this will be off.

33.228 vtkTableToStructuredGrid

33.228.1 Usage

vtkTableToStructuredGrid is a filter that converts an input vtkTable to a vtkStructuredGrid. It provides API to select columns to use as points in the output structured grid. The specified dimensions of the output (specified using SetWholeExtent()) must match the number of rows in the input table.

To create an instance of class vtkTableToStructuredGrid, simply invoke its constructor as follows

```
obj = vtkTableToStructuredGrid
```

33.228.2 Methods

The class vtkTableToStructuredGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableToStructuredGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableToStructuredGrid = obj.NewInstance ()`
- `vtkTableToStructuredGrid = obj.SafeDownCast (vtkObject o)`
- `obj.SetWholeExtent (int , int , int , int , int , int )` - Get/Set the whole extents for the image to produce. The size of the image must match the number of rows in the input table.
- `obj.SetWholeExtent (int a[6])` - Get/Set the whole extents for the image to produce. The size of the image must match the number of rows in the input table.
- `int = obj. GetWholeExtent ()` - Get/Set the whole extents for the image to produce. The size of the image must match the number of rows in the input table.
- `obj.SetXColumn (string )` - Set the name of the column to use as the X coordinate for the points.
- `string = obj.GetXColumn ()` - Set the name of the column to use as the X coordinate for the points.
- `obj.SetXComponent (int )` - Specify the component for the column specified using SetXColumn() to use as the xcoordinate in case the column is a multi-component array. Default is 0.
- `int = obj.GetXComponentMinValue ()` - Specify the component for the column specified using SetX-Column() to use as the xcoordinate in case the column is a multi-component array. Default is 0.
- `int = obj.GetXComponentMaxValue ()` - Specify the component for the column specified using SetX-Column() to use as the xcoordinate in case the column is a multi-component array. Default is 0.
- `int = obj.GetXComponent ()` - Specify the component for the column specified using SetXColumn() to use as the xcoordinate in case the column is a multi-component array. Default is 0.
- `obj.SetYColumn (string )` - Set the name of the column to use as the Y coordinate for the points. Default is 0.
- `string = obj.GetYColumn ()` - Set the name of the column to use as the Y coordinate for the points. Default is 0.
- `obj.SetYComponent (int )` - Specify the component for the column specified using SetYColumn() to use as the Ycoordinate in case the column is a multi-component array.

- `int = obj.GetYComponentMinValue ()` - Specify the component for the column specified using SetYColumn() to use as the Ycoordinate in case the column is a multi-component array.
- `int = obj.GetYComponentMaxValue ()` - Specify the component for the column specified using SetYColumn() to use as the Ycoordinate in case the column is a multi-component array.
- `int = obj.GetYComponent ()` - Specify the component for the column specified using SetYColumn() to use as the Ycoordinate in case the column is a multi-component array.
- `obj.SetZColumn (string )` - Set the name of the column to use as the Z coordinate for the points. Default is 0.
- `string = obj.GetZColumn ()` - Set the name of the column to use as the Z coordinate for the points. Default is 0.
- `obj.SetZComponent (int )` - Specify the component for the column specified using SetZColumn() to use as the Zcoordinate in case the column is a multi-component array.
- `int = obj.GetZComponentMinValue ()` - Specify the component for the column specified using SetZColumn() to use as the Zcoordinate in case the column is a multi-component array.
- `int = obj.GetZComponentMaxValue ()` - Specify the component for the column specified using SetZColumn() to use as the Zcoordinate in case the column is a multi-component array.
- `int = obj.GetZComponent ()` - Specify the component for the column specified using SetZColumn() to use as the Zcoordinate in case the column is a multi-component array.

33.229 vtkTemporalPathLineFilter

33.229.1 Usage

vtkTemporalPathLineFilter takes any dataset as input, it extracts the point locations of all cells over time to build up a polyline trail. The point number (index) is used as the 'key' if the points are randomly changing their respective order in the points list, then you should specify a scalar that represents the unique ID. This is intended to handle the output of a filter such as the TemporalStreamTracer.

To create an instance of class vtkTemporalPathLineFilter, simply invoke its constructor as follows

```
obj = vtkTemporalPathLineFilter
```

33.229.2 Methods

The class vtkTemporalPathLineFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalPathLineFilter class.

- `string = obj.GetClassName ()` - Standard Type-Macro
- `int = obj.IsA (string name)` - Standard Type-Macro
- `vtkTemporalPathLineFilter = obj.NewInstance ()` - Standard Type-Macro
- `vtkTemporalPathLineFilter = obj.SafeDownCast (vtkObject o)` - Standard Type-Macro
- `obj.SetMaskPoints (int )` - Set the number of particles to track as a ratio of the input example: setting MaskPoints to 10 will track every 10th point
- `int = obj.GetMaskPoints ()` - Set the number of particles to track as a ratio of the input example: setting MaskPoints to 10 will track every 10th point

- `obj.SetMaxTrackLength (int )` - If the Particles being traced animate for a long time, the trails or traces will become long and stringy. Setting the MaxTraceTimeLength will limit how much of the trace is displayed. Tracks longer then the Max will disappear and the trace will apppear like a snake of fixed length which progresses as the particle moves

- `int = obj.GetMaxTrackLength ()` - If the Particles being traced animate for a long time, the trails or traces will become long and stringy. Setting the MaxTraceTimeLength will limit how much of the trace is displayed. Tracks longer then the Max will disappear and the trace will apppear like a snake of fixed length which progresses as the particle moves

- `obj.SetIdChannelArray (string )` - Specify the name of a scalar array which will be used to fetch the index of each point. This is necessary only if the particles change position (Id order) on each time step. The Id can be used to identify particles at each step and hence track them properly. If this array is NULL, the global point ids are used. If an Id array cannot otherwise be found, the point index is used as the ID.

- `string = obj.GetIdChannelArray ()` - Specify the name of a scalar array which will be used to fetch the index of each point. This is necessary only if the particles change position (Id order) on each time step. The Id can be used to identify particles at each step and hence track them properly. If this array is NULL, the global point ids are used. If an Id array cannot otherwise be found, the point index is used as the ID.

- `obj.SetScalarArray (string )`

- `string = obj.GetScalarArray ()`

- `obj.SetMaxStepDistance (double , double , double )` - If a particle disappears from one end of a simulation and reappears on the other side, the track left will be unrepresentative. Set a MaxStepDistancex,y,z which acts as a threshold above which if a step occurs larger than the value (for the dimension), the track will be dropped and restarted after the step. (ie the part before the wrap around will be dropped and the newer part kept).

- `obj.SetMaxStepDistance (double  a[3])` - If a particle disappears from one end of a simulation and reappears on the other side, the track left will be unrepresentative. Set a MaxStepDistancex,y,z which acts as a threshold above which if a step occurs larger than the value (for the dimension), the track will be dropped and restarted after the step. (ie the part before the wrap around will be dropped and the newer part kept).

- `double = obj. GetMaxStepDistance ()` - If a particle disappears from one end of a simulation and reappears on the other side, the track left will be unrepresentative. Set a MaxStepDistancex,y,z which acts as a threshold above which if a step occurs larger than the value (for the dimension), the track will be dropped and restarted after the step. (ie the part before the wrap around will be dropped and the newer part kept).

- `obj.SetKeepDeadTrails (int )` - When a particle 'disappears', the trail belonging to it is removed from the list. When this flag is enabled, dead trails will persist until the next time the list is cleared. Use carefully as it may cause excessive memory consumption if left on by mistake.

- `int = obj.GetKeepDeadTrails ()` - When a particle 'disappears', the trail belonging to it is removed from the list. When this flag is enabled, dead trails will persist until the next time the list is cleared. Use carefully as it may cause excessive memory consumption if left on by mistake.

- `obj.Flush ()` - Flush will wipe any existing data so that traces can be restarted from whatever time step is next supplied.

- `obj.SetSelectionConnection (vtkAlgorithmOutput algOutput)` - Set a second input which is a selection. Particles with the same Id in the selection as the primary input will be chosen for pathlines Note that you must have the same IdChannelArray in the selection as the input

- `obj.SetSelection (vtkDataSet input)` - Set a second input which is a selection. Particles with the same Id in the selection as the primary input will be chosen for pathlines Note that you must have the same IdChannelArray in the selection as the input

33.230 vtkTemporalStatistics

33.230.1 Usage

Given an input that changes over time, vtkTemporalStatistics looks at the data for each time step and computes some statistical information of how a point or cell variable changes over time. For example, vtkTemporalStatistics can compute the average value of "pressure" over time of each point.

Note that this filter will require the upstream filter to be run on every time step that it reports that it can compute. This may be a time consuming operation.

vtkTemporalStatistics ignores the temporal spacing. Each timestep will be weighted the same regardless of how long of an interval it is to the next timestep. Thus, the average statistic may be quite different from an integration of the variable if the time spacing varies.

.SECTION Thanks This class was originally written by Kenneth Moreland (kmorel@sandia.gov) from Sandia National Laboratories.

To create an instance of class vtkTemporalStatistics, simply invoke its constructor as follows

```
obj = vtkTemporalStatistics
```

33.230.2 Methods

The class vtkTemporalStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTemporalStatistics = obj.NewInstance ()`
- `vtkTemporalStatistics = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetComputeAverage ()` - Turn on/off the computation of the average values over time. On by default. The resulting array names have "_average" appended to them.
- `obj.SetComputeAverage (int )` - Turn on/off the computation of the average values over time. On by default. The resulting array names have "_average" appended to them.
- `obj.ComputeAverageOn ()` - Turn on/off the computation of the average values over time. On by default. The resulting array names have "_average" appended to them.
- `obj.ComputeAverageOff ()` - Turn on/off the computation of the average values over time. On by default. The resulting array names have "_average" appended to them.
- `int = obj.GetComputeMinimum ()` - Turn on/off the computation of the minimum values over time. On by default. The resulting array names have "_minimum" appended to them.
- `obj.SetComputeMinimum (int )` - Turn on/off the computation of the minimum values over time. On by default. The resulting array names have "_minimum" appended to them.
- `obj.ComputeMinimumOn ()` - Turn on/off the computation of the minimum values over time. On by default. The resulting array names have "_minimum" appended to them.

- `obj.ComputeMinimumOff ()` - Turn on/off the computation of the minimum values over time. On by default. The resulting array names have "_minimum" appended to them.
- `int = obj.GetComputeMaximum ()` - Turn on/off the computation of the maximum values over time. On by default. The resulting array names have "_maximum" appended to them.
- `obj.SetComputeMaximum (int )` - Turn on/off the computation of the maximum values over time. On by default. The resulting array names have "_maximum" appended to them.
- `obj.ComputeMaximumOn ()` - Turn on/off the computation of the maximum values over time. On by default. The resulting array names have "_maximum" appended to them.
- `obj.ComputeMaximumOff ()` - Turn on/off the computation of the maximum values over time. On by default. The resulting array names have "_maximum" appended to them.
- `int = obj.GetComputeStandardDeviation ()`
- `obj.SetComputeStandardDeviation (int )`
- `obj.ComputeStandardDeviationOn ()`
- `obj.ComputeStandardDeviationOff ()`

33.231 vtkTensorGlyph

33.231.1 Usage

vtkTensorGlyph is a filter that copies a geometric representation (specified as polygonal data) to every input point. The geometric representation, or glyph, can be scaled and/or rotated according to the tensor at the input point. Scaling and rotation is controlled by the eigenvalues/eigenvectors of the tensor as follows. For each tensor, the eigenvalues (and associated eigenvectors) are sorted to determine the major, medium, and minor eigenvalues/eigenvectors.

If the boolean variable ThreeGlyphs is not set the major eigenvalue scales the glyph in the x-direction, the medium in the y-direction, and the minor in the z-direction. Then, the glyph is rotated so that the glyph's local x-axis lies along the major eigenvector, y-axis along the medium eigenvector, and z-axis along the minor.

If the boolean variable ThreeGlyphs is set three glyphs are produced, each of them oriented along an eigenvector and scaled according to the corresponding eigenvector.

If the boolean variable Symmetric is set each glyph is mirrored (2 or 6 glyphs will be produced)

The x-axis of the source glyph will correspond to the eigenvector on output. Point (0,0,0) in the source will be placed in the data point. Variable Length will normally correspond to the distance from the origin to the tip of the source glyph along the x-axis, but can be changed to produce other results when Symmetric is on, e.g. glyphs that do not touch or that overlap.

Please note that when Symmetric is false it will generally be better to place the source glyph from (-0.5,0,0) to (0.5,0,0), i.e. centred at the origin. When symmetric is true the placement from (0,0,0) to (1,0,0) will generally be more convenient.

A scale factor is provided to control the amount of scaling. Also, you can turn off scaling completely if desired. The boolean variable ClampScaling controls the maximum scaling (in conjunction with MaxScaleFactor.) This is useful in certain applications where singularities or large order of magnitude differences exist in the eigenvalues.

If the boolean variable ColorGlyphs is set to true the glyphs are colored. The glyphs can be colored using the input scalars (SetColorModeToScalars), which is the default, or colored using the eigenvalues (SetColorModeToEigenvalues).

Another instance variable, ExtractEigenvalues, has been provided to control extraction of eigenvalues/eigenvectors. If this boolean is false, then eigenvalues/eigenvectors are not extracted, and the columns of the tensor are taken as the eigenvectors (the norm of column, always positive, is the eigenvalue). This allows

additional capability over the vtkGlyph3D object. That is, the glyph can be oriented in three directions instead of one.

To create an instance of class vtkTensorGlyph, simply invoke its constructor as follows

```
obj = vtkTensorGlyph
```

33.231.2 Methods

The class vtkTensorGlyph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTensorGlyph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTensorGlyph = obj.NewInstance ()`
- `vtkTensorGlyph = obj.SafeDownCast (vtkObject o)`
- `obj.SetSource (vtkPolyData source)` - Specify the geometry to copy to each point. Old style. See SetSourceConnection.
- `vtkPolyData = obj.GetSource ()` - Specify the geometry to copy to each point. Old style. See SetSourceConnection.
- `obj.SetSourceConnection (int id, vtkAlgorithmOutput algOutput)` - Specify a source object at a specified table location. New style. Source connection is stored in port 1. This method is equivalent to SetInputConnection(1, id, outputPort).
- `obj.SetSourceConnection (vtkAlgorithmOutput algOutput)` - Turn on/off scaling of glyph with eigenvalues.
- `obj.SetScaling (int )` - Turn on/off scaling of glyph with eigenvalues.
- `int = obj.GetScaling ()` - Turn on/off scaling of glyph with eigenvalues.
- `obj.ScalingOn ()` - Turn on/off scaling of glyph with eigenvalues.
- `obj.ScalingOff ()` - Turn on/off scaling of glyph with eigenvalues.
- `obj.SetScaleFactor (double )` - Specify scale factor to scale object by. (Scale factor always affects output even if scaling is off.)
- `double = obj.GetScaleFactor ()` - Specify scale factor to scale object by. (Scale factor always affects output even if scaling is off.)
- `obj.SetThreeGlyphs (int )` - Turn on/off drawing three glyphs
- `int = obj.GetThreeGlyphs ()` - Turn on/off drawing three glyphs
- `obj.ThreeGlyphsOn ()` - Turn on/off drawing three glyphs
- `obj.ThreeGlyphsOff ()` - Turn on/off drawing three glyphs
- `obj.SetSymmetric (int )` - Turn on/off drawing a mirror of each glyph
- `int = obj.GetSymmetric ()` - Turn on/off drawing a mirror of each glyph
- `obj.SymmetricOn ()` - Turn on/off drawing a mirror of each glyph

- `obj.SymmetricOff ()` - Turn on/off drawing a mirror of each glyph
- `obj.SetLength (double )` - Set/Get the distance, along x, from the origin to the end of the source glyph. It is used to draw the symmetric glyphs.
- `double = obj.GetLength ()` - Set/Get the distance, along x, from the origin to the end of the source glyph. It is used to draw the symmetric glyphs.
- `obj.SetExtractEigenvalues (int )` - Turn on/off extraction of eigenvalues from tensor.
- `obj.ExtractEigenvaluesOn ()` - Turn on/off extraction of eigenvalues from tensor.
- `obj.ExtractEigenvaluesOff ()` - Turn on/off extraction of eigenvalues from tensor.
- `int = obj.GetExtractEigenvalues ()` - Turn on/off extraction of eigenvalues from tensor.
- `obj.SetColorGlyphs (int )` - Turn on/off coloring of glyph with input scalar data or eigenvalues. If false, or input scalar data not present, then the scalars from the source object are passed through the filter.
- `int = obj.GetColorGlyphs ()` - Turn on/off coloring of glyph with input scalar data or eigenvalues. If false, or input scalar data not present, then the scalars from the source object are passed through the filter.
- `obj.ColorGlyphsOn ()` - Turn on/off coloring of glyph with input scalar data or eigenvalues. If false, or input scalar data not present, then the scalars from the source object are passed through the filter.
- `obj.ColorGlyphsOff ()` - Turn on/off coloring of glyph with input scalar data or eigenvalues. If false, or input scalar data not present, then the scalars from the source object are passed through the filter.
- `obj.SetColorMode (int )` - Set the color mode to be used for the glyphs. This can be set to use the input scalars (default) or to use the eigenvalues at the point. If ThreeGlyphs is set and the eigenvalues are chosen for coloring then each glyph is colored by the corresponding eigenvalue and if not set the color corresponding to the largest eigenvalue is chosen. The recognized values are: COLOR_BY_SCALARS = 0 (default) COLOR_BY_EIGENVALUES = 1
- `int = obj.GetColorModeMinValue ()` - Set the color mode to be used for the glyphs. This can be set to use the input scalars (default) or to use the eigenvalues at the point. If ThreeGlyphs is set and the eigenvalues are chosen for coloring then each glyph is colored by the corresponding eigenvalue and if not set the color corresponding to the largest eigenvalue is chosen. The recognized values are: COLOR_BY_SCALARS = 0 (default) COLOR_BY_EIGENVALUES = 1
- `int = obj.GetColorModeMaxValue ()` - Set the color mode to be used for the glyphs. This can be set to use the input scalars (default) or to use the eigenvalues at the point. If ThreeGlyphs is set and the eigenvalues are chosen for coloring then each glyph is colored by the corresponding eigenvalue and if not set the color corresponding to the largest eigenvalue is chosen. The recognized values are: COLOR_BY_SCALARS = 0 (default) COLOR_BY_EIGENVALUES = 1
- `int = obj.GetColorMode ()` - Set the color mode to be used for the glyphs. This can be set to use the input scalars (default) or to use the eigenvalues at the point. If ThreeGlyphs is set and the eigenvalues are chosen for coloring then each glyph is colored by the corresponding eigenvalue and if not set the color corresponding to the largest eigenvalue is chosen. The recognized values are: COLOR_BY_SCALARS = 0 (default) COLOR_BY_EIGENVALUES = 1
- `obj.SetColorModeToScalars ()` - Set the color mode to be used for the glyphs. This can be set to use the input scalars (default) or to use the eigenvalues at the point. If ThreeGlyphs is set and the eigenvalues are chosen for coloring then each glyph is colored by the corresponding eigenvalue and if not set the color corresponding to the largest eigenvalue is chosen. The recognized values are: COLOR_BY_SCALARS = 0 (default) COLOR_BY_EIGENVALUES = 1

- `obj.SetColorModeToEigenvalues ()` - Set the color mode to be used for the glyphs. This can be set to use the input scalars (default) or to use the eigenvalues at the point. If ThreeGlyphs is set and the eigenvalues are chosen for coloring then each glyph is colored by the corresponding eigenvalue and if not set the color corresponding to the largest eigenvalue is chosen. The recognized values are: COLOR_BY_SCALARS = 0 (default) COLOR_BY_EIGENVALUES = 1

- `obj.SetClampScaling (int )` - Turn on/off scalar clamping. If scalar clamping is on, the ivar MaxScaleFactor is used to control the maximum scale factor. (This is useful to prevent uncontrolled scaling near singularities.)

- `int = obj.GetClampScaling ()` - Turn on/off scalar clamping. If scalar clamping is on, the ivar MaxScaleFactor is used to control the maximum scale factor. (This is useful to prevent uncontrolled scaling near singularities.)

- `obj.ClampScalingOn ()` - Turn on/off scalar clamping. If scalar clamping is on, the ivar MaxScaleFactor is used to control the maximum scale factor. (This is useful to prevent uncontrolled scaling near singularities.)

- `obj.ClampScalingOff ()` - Turn on/off scalar clamping. If scalar clamping is on, the ivar MaxScaleFactor is used to control the maximum scale factor. (This is useful to prevent uncontrolled scaling near singularities.)

- `obj.SetMaxScaleFactor (double )` - Set/Get the maximum allowable scale factor. This value is compared to the combination of the scale factor times the eigenvalue. If less, the scale factor is reset to the MaxScaleFactor. The boolean ClampScaling has to be "on" for this to work.

- `double = obj.GetMaxScaleFactor ()` - Set/Get the maximum allowable scale factor. This value is compared to the combination of the scale factor times the eigenvalue. If less, the scale factor is reset to the MaxScaleFactor. The boolean ClampScaling has to be "on" for this to work.

33.232 vtkTessellatedBoxSource

33.232.1 Usage

vtkTessellatedBoxSource creates a axis-aligned box defined by its bounds and a level of subdivision. Connectivity is strong: points of the vertices and inside the edges are shared between faces. In other words, faces are connected. Each face looks like a grid of quads, each quad is composed of 2 triangles. Given a level of subdivision 'l', each edge has 'l'+2 points, 'l' of them are internal edge points, the 2 other ones are the vertices. Each face has a total of ('l'+2)*('l'+2) points, 4 of them are vertices, 4*'l' are internal edge points, it remains 'l'$\hat{}$2 internal face points.

This source only generate geometry, no DataArrays like normals or texture coordinates.

To create an instance of class vtkTessellatedBoxSource, simply invoke its constructor as follows

```
obj = vtkTessellatedBoxSource
```

33.232.2 Methods

The class vtkTessellatedBoxSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTessellatedBoxSource class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkTessellatedBoxSource = obj.NewInstance ()`

- `vtkTessellatedBoxSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetBounds (double , double , double , double , double , double )` - Set the bounds of the box. See GetBounds() for a detail description.
- `obj.SetBounds (double a[6])` - Set the bounds of the box. See GetBounds() for a detail description.
- `double = obj. GetBounds ()` - Bounds of the box in world coordinates. This a 6-uple of xmin,xmax,ymin, ymax,zmin and zmax. Initial value is (-0.5,0.5,-0.5,0.5,-0.5,0.5), bounds of a cube of length 1 centered at (0,0,0). Bounds are defined such that xmin¡=xmax, ymin¡=ymax and zmin¡zmax.
- `obj.SetLevel (int )` - Set the level of subdivision of the faces.
- `int = obj.GetLevel ()` - Level of subdivision of the faces. Initial value is 0.
- `obj.SetDuplicateSharedPoints (int )` - Flag to tell the source to duplicate points shared between faces (vertices of the box and internal edge points). Initial value is false. Implementation note: duplicating points is an easier method to implement than a minimal number of points.
- `int = obj.GetDuplicateSharedPoints ()` - Flag to tell the source to duplicate points shared between faces (vertices of the box and internal edge points). Initial value is false. Implementation note: duplicating points is an easier method to implement than a minimal number of points.
- `obj.DuplicateSharedPointsOn ()` - Flag to tell the source to duplicate points shared between faces (vertices of the box and internal edge points). Initial value is false. Implementation note: duplicating points is an easier method to implement than a minimal number of points.
- `obj.DuplicateSharedPointsOff ()` - Flag to tell the source to duplicate points shared between faces (vertices of the box and internal edge points). Initial value is false. Implementation note: duplicating points is an easier method to implement than a minimal number of points.
- `obj.SetQuads (int )` - Flag to tell the source to generate either a quad or two triangle for a set of four points. Initial value is false (generate triangles).
- `int = obj.GetQuads ()` - Flag to tell the source to generate either a quad or two triangle for a set of four points. Initial value is false (generate triangles).
- `obj.QuadsOn ()` - Flag to tell the source to generate either a quad or two triangle for a set of four points. Initial value is false (generate triangles).
- `obj.QuadsOff ()` - Flag to tell the source to generate either a quad or two triangle for a set of four points. Initial value is false (generate triangles).

33.233 vtkTessellatorFilter

33.233.1 Usage

This class approximates nonlinear FEM elements with linear simplices.

¡b¿Warning¡/b¿: This class is temporary and will go away at some point after ParaView 1.4.0.

This filter rifles through all the cells in an input vtkDataSet. It tesselates each cell and uses the vtkStreamingTessellator and vtkDataSetEdgeSubdivisionCriterion classes to generate simplices that approximate the nonlinear mesh using some approximation metric (encoded in the particular vtkDataSetEdgeSubdivisionCriterion::EvaluateEdge implementation). The simplices are placed into the filter's output vtkDataSet object by the callback routines AddATetrahedron, AddATriangle, and AddALine, which are registered with the triangulator.

The output mesh will have geometry and any fields specified as attributes in the input mesh's point data. The attribute's copy flags are honored, except for normals.

.SECTION Internals

The filter's main member function is RequestData(). This function first calls SetupOutput() which allocates arrays and some temporary variables for the primitive callbacks (OutputTriangle and OutputLine which are called by AddATriangle and AddALine, respectively). Each cell is given an initial tesselation, which results in one or more calls to OutputTetrahedron, OutputTriangle or OutputLine to add elements to the OutputMesh. Finally, Teardown() is called to free the filter's working space.

To create an instance of class vtkTessellatorFilter, simply invoke its constructor as follows

```
obj = vtkTessellatorFilter
```

33.233.2 Methods

The class vtkTessellatorFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTessellatorFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTessellatorFilter = obj.NewInstance ()`
- `vtkTessellatorFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetTessellator (vtkStreamingTessellator )`
- `vtkStreamingTessellator = obj.GetTessellator ()`
- `obj.SetSubdivider (vtkDataSetEdgeSubdivisionCriterion )`
- `vtkDataSetEdgeSubdivisionCriterion = obj.GetSubdivider ()`
- `long = obj.GetMTime ()`
- `obj.SetOutputDimension (int )` - Set the dimension of the output tessellation. Cells in dimensions higher than the given value will have their boundaries of dimension OutputDimension tessellated. For example, if OutputDimension is 2, a hexahedron's quadrilateral faces would be tessellated rather than its interior.
- `int = obj.GetOutputDimensionMinValue ()` - Set the dimension of the output tessellation. Cells in dimensions higher than the given value will have their boundaries of dimension OutputDimension tessellated. For example, if OutputDimension is 2, a hexahedron's quadrilateral faces would be tessellated rather than its interior.
- `int = obj.GetOutputDimensionMaxValue ()` - Set the dimension of the output tessellation. Cells in dimensions higher than the given value will have their boundaries of dimension OutputDimension tessellated. For example, if OutputDimension is 2, a hexahedron's quadrilateral faces would be tessellated rather than its interior.
- `int = obj.GetOutputDimension ()` - Set the dimension of the output tessellation. Cells in dimensions higher than the given value will have their boundaries of dimension OutputDimension tessellated. For example, if OutputDimension is 2, a hexahedron's quadrilateral faces would be tessellated rather than its interior.
- `obj.SetMaximumNumberOfSubdivisions (int num\_subdiv\_in)` - These are convenience routines for setting properties maintained by the tessellator and subdivider. They are implemented here for ParaView's sake.
- `int = obj.GetMaximumNumberOfSubdivisions ()` - These are convenience routines for setting properties maintained by the tessellator and subdivider. They are implemented here for ParaView's sake.

- `obj.SetChordError (double ce)` - These are convenience routines for setting properties maintained by the tessellator and subdivider. They are implemented here for ParaView's sake.
- `double = obj.GetChordError ()` - These are convenience routines for setting properties maintained by the tessellator and subdivider. They are implemented here for ParaView's sake.
- `obj.ResetFieldCriteria ()` - These methods are for the ParaView client.
- `obj.SetFieldCriterion (int field, double chord)` - These methods are for the ParaView client.
- `int = obj.GetMergePoints ()` - The adaptive tessellation will output vertices that are not shared among cells, even where they should be. This can be corrected to some extents with a vtkMergeFilter. By default, the filter is off and vertices will not be shared.
- `obj.SetMergePoints (int )` - The adaptive tessellation will output vertices that are not shared among cells, even where they should be. This can be corrected to some extents with a vtkMergeFilter. By default, the filter is off and vertices will not be shared.
- `obj.MergePointsOn ()` - The adaptive tessellation will output vertices that are not shared among cells, even where they should be. This can be corrected to some extents with a vtkMergeFilter. By default, the filter is off and vertices will not be shared.
- `obj.MergePointsOff ()` - The adaptive tessellation will output vertices that are not shared among cells, even where they should be. This can be corrected to some extents with a vtkMergeFilter. By default, the filter is off and vertices will not be shared.

33.234 vtkTextSource

33.234.1 Usage

vtkTextSource converts a text string into polygons. This way you can insert text into your renderings. It uses the 9x15 font from X Windows. You can specify if you want the background to be drawn or not. The characters are formed by scan converting the raster font into quadrilaterals. Colors are assigned to the letters using scalar data. To set the color of the characters with the source's actor property, set BackingOff on the text source and ScalarVisibilityOff on the associated vtkPolyDataMapper. Then, the color can be set using the associated actor's property.

vtkVectorText generates higher quality polygonal representations of characters.

To create an instance of class vtkTextSource, simply invoke its constructor as follows

```
obj = vtkTextSource
```

33.234.2 Methods

The class vtkTextSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTextSource = obj.NewInstance ()`
- `vtkTextSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetText (string )` - Set/Get the text to be drawn.
- `string = obj.GetText ()` - Set/Get the text to be drawn.

- `obj.SetBacking (int )` - Controls whether or not a background is drawn with the text.
- `int = obj.GetBacking ()` - Controls whether or not a background is drawn with the text.
- `obj.BackingOn ()` - Controls whether or not a background is drawn with the text.
- `obj.BackingOff ()` - Controls whether or not a background is drawn with the text.
- `obj.SetForegroundColor (double , double , double )` - Set/Get the foreground color. Default is white (1,1,1). ALpha is always 1.
- `obj.SetForegroundColor (double a[3])` - Set/Get the foreground color. Default is white (1,1,1). ALpha is always 1.
- `double = obj. GetForegroundColor ()` - Set/Get the foreground color. Default is white (1,1,1). ALpha is always 1.
- `obj.SetBackgroundColor (double , double , double )` - Set/Get the background color. Default is black (0,0,0). Alpha is always 1.
- `obj.SetBackgroundColor (double a[3])` - Set/Get the background color. Default is black (0,0,0). Alpha is always 1.
- `double = obj. GetBackgroundColor ()` - Set/Get the background color. Default is black (0,0,0). Alpha is always 1.

33.235 vtkTexturedSphereSource

33.235.1 Usage

vtkTexturedSphereSource creates a polygonal sphere of specified radius centered at the origin. The resolution (polygonal discretization) in both the latitude (phi) and longitude (theta) directions can be specified. It also is possible to create partial sphere by specifying maximum phi and theta angles.

To create an instance of class vtkTexturedSphereSource, simply invoke its constructor as follows

```
obj = vtkTexturedSphereSource
```

33.235.2 Methods

The class vtkTexturedSphereSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTexturedSphereSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTexturedSphereSource = obj.NewInstance ()`
- `vtkTexturedSphereSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetRadius (double )` - Set radius of sphere.
- `double = obj.GetRadiusMinValue ()` - Set radius of sphere.
- `double = obj.GetRadiusMaxValue ()` - Set radius of sphere.
- `double = obj.GetRadius ()` - Set radius of sphere.
- `obj.SetThetaResolution (int )` - Set the number of points in the longitude direction.

- `int = obj.GetThetaResolutionMinValue ()` - Set the number of points in the longitude direction.
- `int = obj.GetThetaResolutionMaxValue ()` - Set the number of points in the longitude direction.
- `int = obj.GetThetaResolution ()` - Set the number of points in the longitude direction.
- `obj.SetPhiResolution (int )` - Set the number of points in the latitude direction.
- `int = obj.GetPhiResolutionMinValue ()` - Set the number of points in the latitude direction.
- `int = obj.GetPhiResolutionMaxValue ()` - Set the number of points in the latitude direction.
- `int = obj.GetPhiResolution ()` - Set the number of points in the latitude direction.
- `obj.SetTheta (double )` - Set the maximum longitude angle.
- `double = obj.GetThetaMinValue ()` - Set the maximum longitude angle.
- `double = obj.GetThetaMaxValue ()` - Set the maximum longitude angle.
- `double = obj.GetTheta ()` - Set the maximum longitude angle.
- `obj.SetPhi (double )` - Set the maximum latitude angle (0 is at north pole).
- `double = obj.GetPhiMinValue ()` - Set the maximum latitude angle (0 is at north pole).
- `double = obj.GetPhiMaxValue ()` - Set the maximum latitude angle (0 is at north pole).
- `double = obj.GetPhi ()` - Set the maximum latitude angle (0 is at north pole).

33.236 vtkTextureMapToCylinder

33.236.1 Usage

vtkTextureMapToCylinder is a filter that generates 2D texture coordinates by mapping input dataset points onto a cylinder. The cylinder can either be user specified or generated automatically. (The cylinder is generated automatically by computing the axis of the cylinder.) Note that the generated texture coordinates for the s-coordinate ranges from (0-1) (corresponding to angle of 0-¿360 around axis), while the mapping of the t-coordinate is controlled by the projection of points along the axis.

To specify a cylinder manually, you must provide two points that define the axis of the cylinder. The length of the axis will affect the t-coordinates.

A special ivar controls how the s-coordinate is generated. If PreventSeam is set to true, the s-texture varies from 0-¿1 and then 1-¿0 (corresponding to angles of 0-¿180 and 180-¿360).

To create an instance of class vtkTextureMapToCylinder, simply invoke its constructor as follows

```
obj = vtkTextureMapToCylinder
```

33.236.2 Methods

The class vtkTextureMapToCylinder has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextureMapToCylinder class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTextureMapToCylinder = obj.NewInstance ()`
- `vtkTextureMapToCylinder = obj.SafeDownCast (vtkObject o)`

- `obj.SetPoint1 (double , double , double )` - Specify the first point defining the cylinder axis,
- `obj.SetPoint1 (double  a[3])` - Specify the first point defining the cylinder axis,
- `double = obj. GetPoint1 ()` - Specify the first point defining the cylinder axis,
- `obj.SetPoint2 (double , double , double )` - Specify the second point defining the cylinder axis,
- `obj.SetPoint2 (double  a[3])` - Specify the second point defining the cylinder axis,
- `double = obj. GetPoint2 ()` - Specify the second point defining the cylinder axis,
- `obj.SetAutomaticCylinderGeneration (int )` - Turn on/off automatic cylinder generation. This means it automatically finds the cylinder center and axis.
- `int = obj.GetAutomaticCylinderGeneration ()` - Turn on/off automatic cylinder generation. This means it automatically finds the cylinder center and axis.
- `obj.AutomaticCylinderGenerationOn ()` - Turn on/off automatic cylinder generation. This means it automatically finds the cylinder center and axis.
- `obj.AutomaticCylinderGenerationOff ()` - Turn on/off automatic cylinder generation. This means it automatically finds the cylinder center and axis.
- `obj.SetPreventSeam (int )` - Control how the texture coordinates are generated. If PreventSeam is set, the s-coordinate ranges from 0-¿1 and 1-¿0 corresponding to the angle variation from 0-¿180 and 180-¿0. Otherwise, the s-coordinate ranges from 0-¿1 from 0-¿360 degrees.
- `int = obj.GetPreventSeam ()` - Control how the texture coordinates are generated. If PreventSeam is set, the s-coordinate ranges from 0-¿1 and 1-¿0 corresponding to the angle variation from 0-¿180 and 180-¿0. Otherwise, the s-coordinate ranges from 0-¿1 from 0-¿360 degrees.
- `obj.PreventSeamOn ()` - Control how the texture coordinates are generated. If PreventSeam is set, the s-coordinate ranges from 0-¿1 and 1-¿0 corresponding to the angle variation from 0-¿180 and 180-¿0. Otherwise, the s-coordinate ranges from 0-¿1 from 0-¿360 degrees.
- `obj.PreventSeamOff ()` - Control how the texture coordinates are generated. If PreventSeam is set, the s-coordinate ranges from 0-¿1 and 1-¿0 corresponding to the angle variation from 0-¿180 and 180-¿0. Otherwise, the s-coordinate ranges from 0-¿1 from 0-¿360 degrees.

33.237 vtkTextureMapToPlane

33.237.1 Usage

vtkTextureMapToPlane is a filter that generates 2D texture coordinates by mapping input dataset points onto a plane. The plane can either be user specified or generated automatically. (A least squares method is used to generate the plane automatically.)

There are two ways you can specify the plane. The first is to provide a plane normal. In this case the points are projected to a plane, and the points are then mapped into the user specified s-t coordinate range. For more control, you can specify a plane with three points: an origin and two points defining the two axes of the plane. (This is compatible with the vtkPlaneSource.) Using the second method, the SRange and TRange vectors are ignored, since the presumption is that the user does not want to scale the texture coordinates; and you can adjust the origin and axes points to achieve the texture coordinate scaling you need. Note also that using the three point method the axes do not have to be orthogonal.

To create an instance of class vtkTextureMapToPlane, simply invoke its constructor as follows

```
obj = vtkTextureMapToPlane
```

33.237.2 Methods

The class vtkTextureMapToPlane has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextureMapToPlane class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTextureMapToPlane = obj.NewInstance ()`
- `vtkTextureMapToPlane = obj.SafeDownCast (vtkObject o)`
- `obj.SetOrigin (double , double , double )` - Specify a point defining the origin of the plane. Used in conjunction with the Point1 and Point2 ivars to specify a map plane.
- `obj.SetOrigin (double a[3])` - Specify a point defining the origin of the plane. Used in conjunction with the Point1 and Point2 ivars to specify a map plane.
- `double = obj. GetOrigin ()` - Specify a point defining the origin of the plane. Used in conjunction with the Point1 and Point2 ivars to specify a map plane.
- `obj.SetPoint1 (double , double , double )` - Specify a point defining the first axis of the plane.
- `obj.SetPoint1 (double a[3])` - Specify a point defining the first axis of the plane.
- `double = obj. GetPoint1 ()` - Specify a point defining the first axis of the plane.
- `obj.SetPoint2 (double , double , double )` - Specify a point defining the second axis of the plane.
- `obj.SetPoint2 (double a[3])` - Specify a point defining the second axis of the plane.
- `double = obj. GetPoint2 ()` - Specify a point defining the second axis of the plane.
- `obj.SetNormal (double , double , double )` - Specify plane normal. An alternative way to specify a map plane. Using this method, the object will scale the resulting texture coordinate between the SRange and TRange specified.
- `obj.SetNormal (double a[3])` - Specify plane normal. An alternative way to specify a map plane. Using this method, the object will scale the resulting texture coordinate between the SRange and TRange specified.
- `double = obj. GetNormal ()` - Specify plane normal. An alternative way to specify a map plane. Using this method, the object will scale the resulting texture coordinate between the SRange and TRange specified.
- `obj.SetSRange (double , double )` - Specify s-coordinate range for texture s-t coordinate pair.
- `obj.SetSRange (double a[2])` - Specify s-coordinate range for texture s-t coordinate pair.
- `double = obj. GetSRange ()` - Specify s-coordinate range for texture s-t coordinate pair.
- `obj.SetTRange (double , double )` - Specify t-coordinate range for texture s-t coordinate pair.
- `obj.SetTRange (double a[2])` - Specify t-coordinate range for texture s-t coordinate pair.
- `double = obj. GetTRange ()` - Specify t-coordinate range for texture s-t coordinate pair.
- `obj.SetAutomaticPlaneGeneration (int )` - Turn on/off automatic plane generation.
- `int = obj.GetAutomaticPlaneGeneration ()` - Turn on/off automatic plane generation.
- `obj.AutomaticPlaneGenerationOn ()` - Turn on/off automatic plane generation.
- `obj.AutomaticPlaneGenerationOff ()` - Turn on/off automatic plane generation.

33.238 vtkTextureMapToSphere

33.238.1 Usage

vtkTextureMapToSphere is a filter that generates 2D texture coordinates by mapping input dataset points onto a sphere. The sphere can either be user specified or generated automatically. (The sphere is generated automatically by computing the center (i.e., averaged coordinates) of the sphere.) Note that the generated texture coordinates range between (0,1). The s-coordinate lies in the angular direction around the z-axis, measured counter-clockwise from the x-axis. The t-coordinate lies in the angular direction measured down from the north pole towards the south pole.

A special ivar controls how the s-coordinate is generated. If PreventSeam is set to true, the s-texture varies from 0-¿1 and then 1-¿0 (corresponding to angles of 0-¿180 and 180-¿360).

To create an instance of class vtkTextureMapToSphere, simply invoke its constructor as follows

```
obj = vtkTextureMapToSphere
```

33.238.2 Methods

The class vtkTextureMapToSphere has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextureMapToSphere class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTextureMapToSphere = obj.NewInstance ()`
- `vtkTextureMapToSphere = obj.SafeDownCast (vtkObject o)`
- `obj.SetCenter (double , double , double )` - Specify a point defining the center of the sphere.
- `obj.SetCenter (double  a[3])` - Specify a point defining the center of the sphere.
- `double = obj. GetCenter ()` - Specify a point defining the center of the sphere.
- `obj.SetAutomaticSphereGeneration (int )` - Turn on/off automatic sphere generation. This means it automatically finds the sphere center.
- `int = obj.GetAutomaticSphereGeneration ()` - Turn on/off automatic sphere generation. This means it automatically finds the sphere center.
- `obj.AutomaticSphereGenerationOn ()` - Turn on/off automatic sphere generation. This means it automatically finds the sphere center.
- `obj.AutomaticSphereGenerationOff ()` - Turn on/off automatic sphere generation. This means it automatically finds the sphere center.
- `obj.SetPreventSeam (int )` - Control how the texture coordinates are generated. If PreventSeam is set, the s-coordinate ranges from 0-¿1 and 1-¿0 corresponding to the theta angle variation between 0-¿180 and 180-¿0 degrees. Otherwise, the s-coordinate ranges from 0-¿1 between 0-¿360 degrees.
- `int = obj.GetPreventSeam ()` - Control how the texture coordinates are generated. If PreventSeam is set, the s-coordinate ranges from 0-¿1 and 1-¿0 corresponding to the theta angle variation between 0-¿180 and 180-¿0 degrees. Otherwise, the s-coordinate ranges from 0-¿1 between 0-¿360 degrees.
- `obj.PreventSeamOn ()` - Control how the texture coordinates are generated. If PreventSeam is set, the s-coordinate ranges from 0-¿1 and 1-¿0 corresponding to the theta angle variation between 0-¿180 and 180-¿0 degrees. Otherwise, the s-coordinate ranges from 0-¿1 between 0-¿360 degrees.

- `obj.PreventSeamOff ()` - Control how the texture coordinates are generated. If PreventSeam is set, the s-coordinate ranges from 0-¿1 and 1-¿0 corresponding to the theta angle variation between 0-¿180 and 180-¿0 degrees. Otherwise, the s-coordinate ranges from 0-¿1 between 0-¿360 degrees.

33.239 vtkThreshold

33.239.1 Usage

vtkThreshold is a filter that extracts cells from any dataset type that satisfy a threshold criterion. A cell satisfies the criterion if the scalar value of (every or any) point satisfies the criterion. The criterion can take three forms: 1) greater than a particular value; 2) less than a particular value; or 3) between two values. The output of this filter is an unstructured grid.

Note that scalar values are available from the point and cell attribute data. By default, point data is used to obtain scalars, but you can control this behavior. See the AttributeMode ivar below.

By default only the first scalar value is used in the decision. Use the ComponentMode and SelectedComponent ivars to control this behavior.

To create an instance of class vtkThreshold, simply invoke its constructor as follows

```
obj = vtkThreshold
```

33.239.2 Methods

The class vtkThreshold has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkThreshold class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkThreshold = obj.NewInstance ()`
- `vtkThreshold = obj.SafeDownCast (vtkObject o)`
- `obj.ThresholdByLower (double lower)` - Criterion is cells whose scalars are less or equal to lower threshold.
- `obj.ThresholdByUpper (double upper)` - Criterion is cells whose scalars are greater or equal to upper threshold.
- `obj.ThresholdBetween (double lower, double upper)` - Criterion is cells whose scalars are between lower and upper thresholds (inclusive of the end values).
- `double = obj.GetUpperThreshold ()` - Get the Upper and Lower thresholds.
- `double = obj.GetLowerThreshold ()` - Get the Upper and Lower thresholds.
- `obj.SetAttributeMode (int )` - Control how the filter works with scalar point data and cell attribute data. By default (AttributeModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).
- `int = obj.GetAttributeMode ()` - Control how the filter works with scalar point data and cell attribute data. By default (AttributeModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).

- `obj.SetAttributeModeToDefault ()` - Control how the filter works with scalar point data and cell attribute data. By default (AttributeModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).

- `obj.SetAttributeModeToUsePointData ()` - Control how the filter works with scalar point data and cell attribute data. By default (AttributeModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).

- `obj.SetAttributeModeToUseCellData ()` - Control how the filter works with scalar point data and cell attribute data. By default (AttributeModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).

- `string = obj.GetAttributeModeAsString ()` - Control how the filter works with scalar point data and cell attribute data. By default (AttributeModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).

- `obj.SetComponentMode (int )` - Control how the decision of in / out is made with multi-component data. The choices are to use the selected component (specified in the SelectedComponent ivar), or to look at all components. When looking at all components, the evaluation can pass if all the components satisfy the rule (UseAll) or if any satisfy is (UseAny). The default value is UseSelected.

- `int = obj.GetComponentModeMinValue ()` - Control how the decision of in / out is made with multi-component data. The choices are to use the selected component (specified in the SelectedComponent ivar), or to look at all components. When looking at all components, the evaluation can pass if all the components satisfy the rule (UseAll) or if any satisfy is (UseAny). The default value is UseSelected.

- `int = obj.GetComponentModeMaxValue ()` - Control how the decision of in / out is made with multi-component data. The choices are to use the selected component (specified in the SelectedComponent ivar), or to look at all components. When looking at all components, the evaluation can pass if all the components satisfy the rule (UseAll) or if any satisfy is (UseAny). The default value is UseSelected.

- `int = obj.GetComponentMode ()` - Control how the decision of in / out is made with multi-component data. The choices are to use the selected component (specified in the SelectedComponent ivar), or to look at all components. When looking at all components, the evaluation can pass if all the components satisfy the rule (UseAll) or if any satisfy is (UseAny). The default value is UseSelected.

- `obj.SetComponentModeToUseSelected ()` - Control how the decision of in / out is made with multi-component data. The choices are to use the selected component (specified in the SelectedComponent ivar), or to look at all components. When looking at all components, the evaluation can pass if all the components satisfy the rule (UseAll) or if any satisfy is (UseAny). The default value is UseSelected.

- `obj.SetComponentModeToUseAll ()` - Control how the decision of in / out is made with multi-component data. The choices are to use the selected component (specified in the SelectedComponent ivar), or to look at all components. When looking at all components, the evaluation can pass if all the components satisfy the rule (UseAll) or if any satisfy is (UseAny). The default value is UseSelected.

- `obj.SetComponentModeToUseAny ()` - Control how the decision of in / out is made with multi-component data. The choices are to use the selected component (specified in the SelectedComponent ivar), or to look at all components. When looking at all components, the evaluation can pass if all the components satisfy the rule (UseAll) or if any satisfy is (UseAny). The default value is UseSelected.

- `string = obj.GetComponentModeAsString ()` - Control how the decision of in / out is made with multi-component data. The choices are to use the selected component (specified in the SelectedComponent ivar), or to look at all components. When looking at all components, the evaluation can pass

if all the components satisfy the rule (UseAll) or if any satisfy is (UseAny). The default value is UseSelected.

- `obj.SetSelectedComponent (int )` - When the component mode is UseSelected, this ivar indicated the selected component. The default value is 0.
- `int = obj.GetSelectedComponentMinValue ()` - When the component mode is UseSelected, this ivar indicated the selected component. The default value is 0.
- `int = obj.GetSelectedComponentMaxValue ()` - When the component mode is UseSelected, this ivar indicated the selected component. The default value is 0.
- `int = obj.GetSelectedComponent ()` - When the component mode is UseSelected, this ivar indicated the selected component. The default value is 0.
- `obj.SetAllScalars (int )` - If using scalars from point data, all scalars for all points in a cell must satisfy the threshold criterion if AllScalars is set. Otherwise, just a single scalar value satisfying the threshold criterion enables will extract the cell.
- `int = obj.GetAllScalars ()` - If using scalars from point data, all scalars for all points in a cell must satisfy the threshold criterion if AllScalars is set. Otherwise, just a single scalar value satisfying the threshold criterion enables will extract the cell.
- `obj.AllScalarsOn ()` - If using scalars from point data, all scalars for all points in a cell must satisfy the threshold criterion if AllScalars is set. Otherwise, just a single scalar value satisfying the threshold criterion enables will extract the cell.
- `obj.AllScalarsOff ()` - If using scalars from point data, all scalars for all points in a cell must satisfy the threshold criterion if AllScalars is set. Otherwise, just a single scalar value satisfying the threshold criterion enables will extract the cell.
- `obj.SetPointsDataTypeToDouble ()` - Set the data type of the output points (See the data types defined in vtkType.h). The default data type is float.
- `obj.SetPointsDataTypeToFloat ()` - Set the data type of the output points (See the data types defined in vtkType.h). The default data type is float.
- `obj.SetPointsDataType (int )` - Set the data type of the output points (See the data types defined in vtkType.h). The default data type is float.
- `int = obj.GetPointsDataType ()` - Set the data type of the output points (See the data types defined in vtkType.h). The default data type is float.

33.240 vtkThresholdPoints

33.240.1 Usage

vtkThresholdPoints is a filter that extracts points from a dataset that satisfy a threshold criterion. The criterion can take three forms: 1) greater than a particular value; 2) less than a particular value; or 3) between a particular value. The output of the filter is polygonal data.

To create an instance of class vtkThresholdPoints, simply invoke its constructor as follows

```
obj = vtkThresholdPoints
```

33.240.2 Methods

The class vtkThresholdPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkThresholdPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkThresholdPoints = obj.NewInstance ()`
- `vtkThresholdPoints = obj.SafeDownCast (vtkObject o)`
- `obj.ThresholdByLower (double lower)` - Criterion is cells whose scalars are less or equal to lower threshold.
- `obj.ThresholdByUpper (double upper)` - Criterion is cells whose scalars are greater or equal to upper threshold.
- `obj.ThresholdBetween (double lower, double upper)` - Criterion is cells whose scalars are between lower and upper thresholds (inclusive of the end values).
- `obj.SetUpperThreshold (double )` - Set/Get the upper threshold.
- `double = obj.GetUpperThreshold ()` - Set/Get the upper threshold.
- `obj.SetLowerThreshold (double )` - Set/Get the lower threshold.
- `double = obj.GetLowerThreshold ()` - Set/Get the lower threshold.

33.241 vtkThresholdTextureCoords

33.241.1 Usage

vtkThresholdTextureCoords is a filter that generates texture coordinates for any input dataset type given a threshold criterion. The criterion can take three forms: 1) greater than a particular value (ThresholdByUpper()); 2) less than a particular value (ThresholdByLower(); or 3) between two values (ThresholdBetween(). If the threshold criterion is satisfied, the "in" texture coordinate will be set (this can be specified by the user). If the threshold criterion is not satisfied the "out" is set.

To create an instance of class vtkThresholdTextureCoords, simply invoke its constructor as follows

```
obj = vtkThresholdTextureCoords
```

33.241.2 Methods

The class vtkThresholdTextureCoords has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkThresholdTextureCoords class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkThresholdTextureCoords = obj.NewInstance ()`
- `vtkThresholdTextureCoords = obj.SafeDownCast (vtkObject o)`

- `obj.ThresholdByLower (double lower)` - Criterion is cells whose scalars are less than lower threshold.
- `obj.ThresholdByUpper (double upper)` - Criterion is cells whose scalars are less than upper threshold.
- `obj.ThresholdBetween (double lower, double upper)` - Criterion is cells whose scalars are between lower and upper thresholds.
- `double = obj.GetUpperThreshold ()` - Return the upper and lower thresholds.
- `double = obj.GetLowerThreshold ()` - Return the upper and lower thresholds.
- `obj.SetTextureDimension (int )` - Set the desired dimension of the texture map.
- `int = obj.GetTextureDimensionMinValue ()` - Set the desired dimension of the texture map.
- `int = obj.GetTextureDimensionMaxValue ()` - Set the desired dimension of the texture map.
- `int = obj.GetTextureDimension ()` - Set the desired dimension of the texture map.
- `obj.SetInTextureCoord (double , double , double )` - Set the texture coordinate value for point satisfying threshold criterion.
- `obj.SetInTextureCoord (double a[3])` - Set the texture coordinate value for point satisfying threshold criterion.
- `double = obj. GetInTextureCoord ()` - Set the texture coordinate value for point satisfying threshold criterion.
- `obj.SetOutTextureCoord (double , double , double )` - Set the texture coordinate value for point NOT satisfying threshold criterion.
- `obj.SetOutTextureCoord (double a[3])` - Set the texture coordinate value for point NOT satisfying threshold criterion.
- `double = obj. GetOutTextureCoord ()` - Set the texture coordinate value for point NOT satisfying threshold criterion.

33.242 vtkTimeSourceExample

33.242.1 Usage

Creates a small easily understood time varying data set for testing. The output is a vtkUntructuredGrid in which the point and cell values vary over time in a sin wave. The analytic ivar controls whether the output corresponds to a step function over time or is continuous. The X and Y Amplitude ivars make the output move in the X and Y directions over time. The Growing ivar makes the number of cells in the output grow and then shrink over time.

To create an instance of class vtkTimeSourceExample, simply invoke its constructor as follows

```
obj = vtkTimeSourceExample
```

33.242.2 Methods

The class vtkTimeSourceExample has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTimeSourceExample class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTimeSourceExample = obj.NewInstance ()`
- `vtkTimeSourceExample = obj.SafeDownCast (vtkObject o)`
- `obj.SetAnalytic (int )`
- `int = obj.GetAnalyticMinValue ()`
- `int = obj.GetAnalyticMaxValue ()`
- `int = obj.GetAnalytic ()`
- `obj.AnalyticOn ()`
- `obj.AnalyticOff ()`
- `obj.SetXAmplitude (double )`
- `double = obj.GetXAmplitude ()`
- `obj.SetYAmplitude (double )`
- `double = obj.GetYAmplitude ()`
- `obj.SetGrowing (int )`
- `int = obj.GetGrowingMinValue ()`
- `int = obj.GetGrowingMaxValue ()`
- `int = obj.GetGrowing ()`
- `obj.GrowingOn ()`
- `obj.GrowingOff ()`

33.243 vtkTransformCoordinateSystems

33.243.1 Usage

This filter transforms points from one coordinate system to another. The user must specify the coordinate systems in which the input and output are specified. The user must also specify the VTK viewport (i.e., renderer) in which the transformation occurs.

To create an instance of class vtkTransformCoordinateSystems, simply invoke its constructor as follows

```
obj = vtkTransformCoordinateSystems
```

33.243.2 Methods

The class vtkTransformCoordinateSystems has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransformCoordinateSystems class.

- `string = obj.GetClassName ()` - Standard methods for type information and printing.
- `int = obj.IsA (string name)` - Standard methods for type information and printing.
- `vtkTransformCoordinateSystems = obj.NewInstance ()` - Standard methods for type information and printing.
- `vtkTransformCoordinateSystems = obj.SafeDownCast (vtkObject o)` - Standard methods for type information and printing.
- `obj.SetInputCoordinateSystem (int )` - Set/get the coordinate system in which the input is specified. The current options are World, Viewport, and Display. By default the input coordinate system is World.
- `int = obj.GetInputCoordinateSystem ()` - Set/get the coordinate system in which the input is specified. The current options are World, Viewport, and Display. By default the input coordinate system is World.
- `obj.SetInputCoordinateSystemToDisplay ()` - Set/get the coordinate system in which the input is specified. The current options are World, Viewport, and Display. By default the input coordinate system is World.
- `obj.SetInputCoordinateSystemToViewport ()` - Set/get the coordinate system in which the input is specified. The current options are World, Viewport, and Display. By default the input coordinate system is World.
- `obj.SetInputCoordinateSystemToWorld ()` - Set/get the coordinate system to which to transform the output. The current options are World, Viewport, and Display. By default the output coordinate system is Display.
- `obj.SetOutputCoordinateSystem (int )` - Set/get the coordinate system to which to transform the output. The current options are World, Viewport, and Display. By default the output coordinate system is Display.
- `int = obj.GetOutputCoordinateSystem ()` - Set/get the coordinate system to which to transform the output. The current options are World, Viewport, and Display. By default the output coordinate system is Display.
- `obj.SetOutputCoordinateSystemToDisplay ()` - Set/get the coordinate system to which to transform the output. The current options are World, Viewport, and Display. By default the output coordinate system is Display.
- `obj.SetOutputCoordinateSystemToViewport ()` - Set/get the coordinate system to which to transform the output. The current options are World, Viewport, and Display. By default the output coordinate system is Display.
- `obj.SetOutputCoordinateSystemToWorld ()` - Return the MTime also considering the instance of vtkCoordinate.
- `long = obj.GetMTime ()` - Return the MTime also considering the instance of vtkCoordinate.

- `obj.SetViewport (vtkViewport viewport)` - In order for successful coordinate transformation to occur, an instance of vtkViewport (e.g., a VTK renderer) must be specified. NOTE: this is a raw pointer, not a weak pointer not a reference counted object to avoid reference cycle loop between rendering classes and filter classes.
- `vtkViewport = obj.GetViewport ()` - In order for successful coordinate transformation to occur, an instance of vtkViewport (e.g., a VTK renderer) must be specified. NOTE: this is a raw pointer, not a weak pointer not a reference counted object to avoid reference cycle loop between rendering classes and filter classes.

33.244 vtkTransformFilter

33.244.1 Usage

vtkTransformFilter is a filter to transform point coordinates, and associated point normals and vectors. Other point data is passed through the filter.

An alternative method of transformation is to use vtkActor's methods to scale, rotate, and translate objects. The difference between the two methods is that vtkActor's transformation simply effects where objects are rendered (via the graphics pipeline), whereas vtkTransformFilter actually modifies point coordinates in the visualization pipeline. This is necessary for some objects (e.g., vtkProbeFilter) that require point coordinates as input.

To create an instance of class vtkTransformFilter, simply invoke its constructor as follows

```
obj = vtkTransformFilter
```

33.244.2 Methods

The class vtkTransformFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransformFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransformFilter = obj.NewInstance ()`
- `vtkTransformFilter = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Return the MTime also considering the transform.
- `obj.SetTransform (vtkAbstractTransform )` - Specify the transform object used to transform points.
- `vtkAbstractTransform = obj.GetTransform ()` - Specify the transform object used to transform points.

33.245 vtkTransformPolyDataFilter

33.245.1 Usage

vtkTransformPolyDataFilter is a filter to transform point coordinates and associated point and cell normals and vectors. Other point and cell data is passed through the filter unchanged. This filter is specialized for polygonal data. See vtkTransformFilter for more general data.

An alternative method of transformation is to use vtkActor's methods to scale, rotate, and translate objects. The difference between the two methods is that vtkActor's transformation simply effects where objects are rendered (via the graphics pipeline), whereas vtkTransformPolyDataFilter actually modifies

point coordinates in the visualization pipeline. This is necessary for some objects (e.g., vtkProbeFilter) that require point coordinates as input.

To create an instance of class vtkTransformPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkTransformPolyDataFilter
```

33.245.2 Methods

The class vtkTransformPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransformPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransformPolyDataFilter = obj.NewInstance ()`
- `vtkTransformPolyDataFilter = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Return the MTime also considering the transform.
- `obj.SetTransform (vtkAbstractTransform )` - Specify the transform object used to transform points.
- `vtkAbstractTransform = obj.GetTransform ()` - Specify the transform object used to transform points.

33.246 vtkTransformTextureCoords

33.246.1 Usage

vtkTransformTextureCoords is a filter that operates on texture coordinates. It ingests any type of dataset, and outputs a dataset of the same type. The filter lets you scale, translate, and rotate texture coordinates. For example, by using the the Scale ivar, you can shift texture coordinates that range from (0-¿1) to range from (0-¿10) (useful for repeated patterns).

The filter operates on texture coordinates of dimension 1-¿3. The texture coordinates are referred to as r-s-t. If the texture map is two dimensional, the t-coordinate (and operations on the t-coordinate) are ignored.

To create an instance of class vtkTransformTextureCoords, simply invoke its constructor as follows

```
obj = vtkTransformTextureCoords
```

33.246.2 Methods

The class vtkTransformTextureCoords has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransformTextureCoords class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransformTextureCoords = obj.NewInstance ()`
- `vtkTransformTextureCoords = obj.SafeDownCast (vtkObject o)`

- `obj.SetPosition (double , double , double )` - Set/Get the position of the texture map. Setting the position translates the texture map by the amount specified.
- `obj.SetPosition (double a[3])` - Set/Get the position of the texture map. Setting the position translates the texture map by the amount specified.
- `double = obj. GetPosition ()` - Set/Get the position of the texture map. Setting the position translates the texture map by the amount specified.
- `obj.AddPosition (double deltaR, double deltaS, double deltaT)` - Incrementally change the position of the texture map (i.e., does a translate or shift of the texture coordinates).
- `obj.AddPosition (double deltaPosition[3])` - Incrementally change the position of the texture map (i.e., does a translate or shift of the texture coordinates).
- `obj.SetScale (double , double , double )` - Set/Get the scale of the texture map. Scaling in performed independently on the r, s and t axes.
- `obj.SetScale (double a[3])` - Set/Get the scale of the texture map. Scaling in performed independently on the r, s and t axes.
- `double = obj. GetScale ()` - Set/Get the scale of the texture map. Scaling in performed independently on the r, s and t axes.
- `obj.SetOrigin (double , double , double )` - Set/Get the origin of the texture map. This is the point about which the texture map is flipped (e.g., rotated). Since a typical texture map ranges from (0,1) in the r-s-t coordinates, the default origin is set at (0.5,0.5,0.5).
- `obj.SetOrigin (double a[3])` - Set/Get the origin of the texture map. This is the point about which the texture map is flipped (e.g., rotated). Since a typical texture map ranges from (0,1) in the r-s-t coordinates, the default origin is set at (0.5,0.5,0.5).
- `double = obj. GetOrigin ()` - Set/Get the origin of the texture map. This is the point about which the texture map is flipped (e.g., rotated). Since a typical texture map ranges from (0,1) in the r-s-t coordinates, the default origin is set at (0.5,0.5,0.5).
- `obj.SetFlipR (int )` - Boolean indicates whether the texture map should be flipped around the s-axis. Note that the flips occur around the texture origin.
- `int = obj.GetFlipR ()` - Boolean indicates whether the texture map should be flipped around the s-axis. Note that the flips occur around the texture origin.
- `obj.FlipROn ()` - Boolean indicates whether the texture map should be flipped around the s-axis. Note that the flips occur around the texture origin.
- `obj.FlipROff ()` - Boolean indicates whether the texture map should be flipped around the s-axis. Note that the flips occur around the texture origin.
- `obj.SetFlipS (int )` - Boolean indicates whether the texture map should be flipped around the s-axis. Note that the flips occur around the texture origin.
- `int = obj.GetFlipS ()` - Boolean indicates whether the texture map should be flipped around the s-axis. Note that the flips occur around the texture origin.
- `obj.FlipSOn ()` - Boolean indicates whether the texture map should be flipped around the s-axis. Note that the flips occur around the texture origin.
- `obj.FlipSOff ()` - Boolean indicates whether the texture map should be flipped around the s-axis. Note that the flips occur around the texture origin.

- `obj.SetFlipT (int )` - Boolean indicates whether the texture map should be flipped around the t-axis. Note that the flips occur around the texture origin.

- `int = obj.GetFlipT ()` - Boolean indicates whether the texture map should be flipped around the t-axis. Note that the flips occur around the texture origin.

- `obj.FlipTOn ()` - Boolean indicates whether the texture map should be flipped around the t-axis. Note that the flips occur around the texture origin.

- `obj.FlipTOff ()` - Boolean indicates whether the texture map should be flipped around the t-axis. Note that the flips occur around the texture origin.

33.247 vtkTriangleFilter

33.247.1 Usage

vtkTriangleFilter generates triangles from input polygons and triangle strips. The filter also will pass through vertices and lines, if requested.

To create an instance of class vtkTriangleFilter, simply invoke its constructor as follows

```
obj = vtkTriangleFilter
```

33.247.2 Methods

The class vtkTriangleFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTriangleFilter class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkTriangleFilter = obj.NewInstance ()`

- `vtkTriangleFilter = obj.SafeDownCast (vtkObject o)`

- `obj.PassVertsOn ()` - Turn on/off passing vertices through filter.

- `obj.PassVertsOff ()` - Turn on/off passing vertices through filter.

- `obj.SetPassVerts (int )` - Turn on/off passing vertices through filter.

- `int = obj.GetPassVerts ()` - Turn on/off passing vertices through filter.

- `obj.PassLinesOn ()` - Turn on/off passing lines through filter.

- `obj.PassLinesOff ()` - Turn on/off passing lines through filter.

- `obj.SetPassLines (int )` - Turn on/off passing lines through filter.

- `int = obj.GetPassLines ()` - Turn on/off passing lines through filter.

33.248 vtkTriangularTCoords

33.248.1 Usage

vtkTriangularTCoords is a filter that generates texture coordinates for triangles. Texture coordinates for each triangle are: (0,0), (1,0) and (.5,sqrt(3)/2). This filter assumes that the triangle texture map is symmetric about the center of the triangle. Thus the order Of the texture coordinates is not important. The procedural texture in vtkTriangularTexture is designed with this symmetry. For more information see the paper "Opacity-modulating Triangular Textures for Irregular Surfaces," by Penny Rheingans, IEEE Visualization '96, pp. 219-225.

To create an instance of class vtkTriangularTCoords, simply invoke its constructor as follows

```
obj = vtkTriangularTCoords
```

33.248.2 Methods

The class vtkTriangularTCoords has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTriangularTCoords class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTriangularTCoords = obj.NewInstance ()`
- `vtkTriangularTCoords = obj.SafeDownCast (vtkObject o)`

33.249 vtkTubeFilter

33.249.1 Usage

vtkTubeFilter is a filter that generates a tube around each input line. The tubes are made up of triangle strips and rotate around the tube with the rotation of the line normals. (If no normals are present, they are computed automatically.) The radius of the tube can be set to vary with scalar or vector value. If the radius varies with scalar value the radius is linearly adjusted. If the radius varies with vector value, a mass flux preserving variation is used. The number of sides for the tube also can be specified. You can also specify which of the sides are visible. This is useful for generating interesting striping effects. Other options include the ability to cap the tube and generate texture coordinates. Texture coordinates can be used with an associated texture map to create interesting effects such as marking the tube with stripes corresponding to length or time.

This filter is typically used to create thick or dramatic lines. Another common use is to combine this filter with vtkStreamLine to generate streamtubes.

To create an instance of class vtkTubeFilter, simply invoke its constructor as follows

```
obj = vtkTubeFilter
```

33.249.2 Methods

The class vtkTubeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTubeFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkTubeFilter = obj.NewInstance ()`
- `vtkTubeFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetRadius (double )` - Set the minimum tube radius (minimum because the tube radius may vary).
- `double = obj.GetRadiusMinValue ()` - Set the minimum tube radius (minimum because the tube radius may vary).
- `double = obj.GetRadiusMaxValue ()` - Set the minimum tube radius (minimum because the tube radius may vary).
- `double = obj.GetRadius ()` - Set the minimum tube radius (minimum because the tube radius may vary).
- `obj.SetVaryRadius (int )` - Turn on/off the variation of tube radius with scalar value.
- `int = obj.GetVaryRadiusMinValue ()` - Turn on/off the variation of tube radius with scalar value.
- `int = obj.GetVaryRadiusMaxValue ()` - Turn on/off the variation of tube radius with scalar value.
- `int = obj.GetVaryRadius ()` - Turn on/off the variation of tube radius with scalar value.
- `obj.SetVaryRadiusToVaryRadiusOff ()` - Turn on/off the variation of tube radius with scalar value.
- `obj.SetVaryRadiusToVaryRadiusByScalar ()` - Turn on/off the variation of tube radius with scalar value.
- `obj.SetVaryRadiusToVaryRadiusByVector ()` - Turn on/off the variation of tube radius with scalar value.
- `obj.SetVaryRadiusToVaryRadiusByAbsoluteScalar ()` - Turn on/off the variation of tube radius with scalar value.
- `string = obj.GetVaryRadiusAsString ()` - Turn on/off the variation of tube radius with scalar value.
- `obj.SetNumberOfSides (int )` - Set the number of sides for the tube. At a minimum, number of sides is 3.
- `int = obj.GetNumberOfSidesMinValue ()` - Set the number of sides for the tube. At a minimum, number of sides is 3.
- `int = obj.GetNumberOfSidesMaxValue ()` - Set the number of sides for the tube. At a minimum, number of sides is 3.
- `int = obj.GetNumberOfSides ()` - Set the number of sides for the tube. At a minimum, number of sides is 3.
- `obj.SetRadiusFactor (double )` - Set the maximum tube radius in terms of a multiple of the minimum radius.
- `double = obj.GetRadiusFactor ()` - Set the maximum tube radius in terms of a multiple of the minimum radius.
- `obj.SetDefaultNormal (double , double , double )` - Set the default normal to use if no normals are supplied, and the DefaultNormalOn is set.
- `obj.SetDefaultNormal (double  a[3])` - Set the default normal to use if no normals are supplied, and the DefaultNormalOn is set.

- `double = obj. GetDefaultNormal ()` - Set the default normal to use if no normals are supplied, and the DefaultNormalOn is set.
- `obj.SetUseDefaultNormal (int )` - Set a boolean to control whether to use default normals. DefaultNormalOn is set.
- `int = obj.GetUseDefaultNormal ()` - Set a boolean to control whether to use default normals. DefaultNormalOn is set.
- `obj.UseDefaultNormalOn ()` - Set a boolean to control whether to use default normals. DefaultNormalOn is set.
- `obj.UseDefaultNormalOff ()` - Set a boolean to control whether to use default normals. DefaultNormalOn is set.
- `obj.SetSidesShareVertices (int )` - Set a boolean to control whether tube sides should share vertices. This creates independent strips, with constant normals so the tube is always faceted in appearance.
- `int = obj.GetSidesShareVertices ()` - Set a boolean to control whether tube sides should share vertices. This creates independent strips, with constant normals so the tube is always faceted in appearance.
- `obj.SidesShareVerticesOn ()` - Set a boolean to control whether tube sides should share vertices. This creates independent strips, with constant normals so the tube is always faceted in appearance.
- `obj.SidesShareVerticesOff ()` - Set a boolean to control whether tube sides should share vertices. This creates independent strips, with constant normals so the tube is always faceted in appearance.
- `obj.SetCapping (int )` - Turn on/off whether to cap the ends with polygons.
- `int = obj.GetCapping ()` - Turn on/off whether to cap the ends with polygons.
- `obj.CappingOn ()` - Turn on/off whether to cap the ends with polygons.
- `obj.CappingOff ()` - Turn on/off whether to cap the ends with polygons.
- `obj.SetOnRatio (int )` - Control the striping of the tubes. If OnRatio is greater than 1, then every nth tube side is turned on, beginning with the Offset side.
- `int = obj.GetOnRatioMinValue ()` - Control the striping of the tubes. If OnRatio is greater than 1, then every nth tube side is turned on, beginning with the Offset side.
- `int = obj.GetOnRatioMaxValue ()` - Control the striping of the tubes. If OnRatio is greater than 1, then every nth tube side is turned on, beginning with the Offset side.
- `int = obj.GetOnRatio ()` - Control the striping of the tubes. If OnRatio is greater than 1, then every nth tube side is turned on, beginning with the Offset side.
- `obj.SetOffset (int )` - Control the striping of the tubes. The offset sets the first tube side that is visible. Offset is generally used with OnRatio to create nifty striping effects.
- `int = obj.GetOffsetMinValue ()` - Control the striping of the tubes. The offset sets the first tube side that is visible. Offset is generally used with OnRatio to create nifty striping effects.
- `int = obj.GetOffsetMaxValue ()` - Control the striping of the tubes. The offset sets the first tube side that is visible. Offset is generally used with OnRatio to create nifty striping effects.
- `int = obj.GetOffset ()` - Control the striping of the tubes. The offset sets the first tube side that is visible. Offset is generally used with OnRatio to create nifty striping effects.

- `obj.SetGenerateTCoords (int )` - Control whether and how texture coordinates are produced. This is useful for striping the tube with length textures, etc. If you use scalars to create the texture, the scalars are assumed to be monotonically increasing (or decreasing).
- `int = obj.GetGenerateTCoordsMinValue ()` - Control whether and how texture coordinates are produced. This is useful for striping the tube with length textures, etc. If you use scalars to create the texture, the scalars are assumed to be monotonically increasing (or decreasing).
- `int = obj.GetGenerateTCoordsMaxValue ()` - Control whether and how texture coordinates are produced. This is useful for striping the tube with length textures, etc. If you use scalars to create the texture, the scalars are assumed to be monotonically increasing (or decreasing).
- `int = obj.GetGenerateTCoords ()` - Control whether and how texture coordinates are produced. This is useful for striping the tube with length textures, etc. If you use scalars to create the texture, the scalars are assumed to be monotonically increasing (or decreasing).
- `obj.SetGenerateTCoordsToOff ()` - Control whether and how texture coordinates are produced. This is useful for striping the tube with length textures, etc. If you use scalars to create the texture, the scalars are assumed to be monotonically increasing (or decreasing).
- `obj.SetGenerateTCoordsToNormalizedLength ()` - Control whether and how texture coordinates are produced. This is useful for striping the tube with length textures, etc. If you use scalars to create the texture, the scalars are assumed to be monotonically increasing (or decreasing).
- `obj.SetGenerateTCoordsToUseLength ()` - Control whether and how texture coordinates are produced. This is useful for striping the tube with length textures, etc. If you use scalars to create the texture, the scalars are assumed to be monotonically increasing (or decreasing).
- `obj.SetGenerateTCoordsToUseScalars ()` - Control whether and how texture coordinates are produced. This is useful for striping the tube with length textures, etc. If you use scalars to create the texture, the scalars are assumed to be monotonically increasing (or decreasing).
- `string = obj.GetGenerateTCoordsAsString ()` - Control whether and how texture coordinates are produced. This is useful for striping the tube with length textures, etc. If you use scalars to create the texture, the scalars are assumed to be monotonically increasing (or decreasing).
- `obj.SetTextureLength (double )` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space.
- `double = obj.GetTextureLengthMinValue ()` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space.
- `double = obj.GetTextureLengthMaxValue ()` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space.
- `double = obj.GetTextureLength ()` - Control the conversion of units during the texture coordinates calculation. The TextureLength indicates what length (whether calculated from scalars or length) is mapped to the [0,1) texture space.

33.250 vtkUncertaintyTubeFilter

33.250.1 Usage

vtkUncertaintyTubeFilter is a filter that generates ellipsoidal (in cross section) tubes that follows a polyline. The input is a vtkPolyData with polylines that have associated vector point data. The vector data represents the uncertainty of the polyline in the x-y-z directions.

To create an instance of class vtkUncertaintyTubeFilter, simply invoke its constructor as follows

```
obj = vtkUncertaintyTubeFilter
```

33.250.2 Methods

The class vtkUncertaintyTubeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUncertaintyTubeFilter class.

- `string = obj.GetClassName ()` - Standard methods for printing and obtaining type information for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for printing and obtaining type information for instances of this class.
- `vtkUncertaintyTubeFilter = obj.NewInstance ()` - Standard methods for printing and obtaining type information for instances of this class.
- `vtkUncertaintyTubeFilter = obj.SafeDownCast (vtkObject o)` - Standard methods for printing and obtaining type information for instances of this class.
- `obj.SetNumberOfSides (int )` - Set / get the number of sides for the tube. At a minimum, the number of sides is 3.
- `int = obj.GetNumberOfSidesMinValue ()` - Set / get the number of sides for the tube. At a minimum, the number of sides is 3.
- `int = obj.GetNumberOfSidesMaxValue ()` - Set / get the number of sides for the tube. At a minimum, the number of sides is 3.
- `int = obj.GetNumberOfSides ()` - Set / get the number of sides for the tube. At a minimum, the number of sides is 3.

33.251 vtkUnstructuredGridGeometryFilter

33.251.1 Usage

vtkUnstructuredGridGeometryFilter is a filter that extracts geometry (and associated data) from an unstructured grid. It differs from vtkGeometryFilter by not tessellating higher order faces: 2D faces of quadratic 3D cells will be quadratic. A quadratic edge is extracted as a quadratic edge. For that purpose, the output of this filter is an unstructured grid, not a polydata. Also, the face of a voxel is a pixel, not a quad. Geometry is obtained as follows: all 0D, 1D, and 2D cells are extracted. All 2D faces that are used by only one 3D cell (i.e., boundary faces) are extracted. It also is possible to specify conditions on point ids, cell ids, and on bounding box (referred to as "Extent") to control the extraction process.

To create an instance of class vtkUnstructuredGridGeometryFilter, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridGeometryFilter
```

33.251.2 Methods

The class vtkUnstructuredGridGeometryFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridGeometryFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridGeometryFilter = obj.NewInstance ()`
- `vtkUnstructuredGridGeometryFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetPointClipping (int )` - Turn on/off selection of geometry by point id.
- `int = obj.GetPointClipping ()` - Turn on/off selection of geometry by point id.
- `obj.PointClippingOn ()` - Turn on/off selection of geometry by point id.
- `obj.PointClippingOff ()` - Turn on/off selection of geometry by point id.
- `obj.SetCellClipping (int )` - Turn on/off selection of geometry by cell id.
- `int = obj.GetCellClipping ()` - Turn on/off selection of geometry by cell id.
- `obj.CellClippingOn ()` - Turn on/off selection of geometry by cell id.
- `obj.CellClippingOff ()` - Turn on/off selection of geometry by cell id.
- `obj.SetExtentClipping (int )` - Turn on/off selection of geometry via bounding box.
- `int = obj.GetExtentClipping ()` - Turn on/off selection of geometry via bounding box.
- `obj.ExtentClippingOn ()` - Turn on/off selection of geometry via bounding box.
- `obj.ExtentClippingOff ()` - Turn on/off selection of geometry via bounding box.
- `obj.SetPointMinimum (vtkIdType )` - Specify the minimum point id for point id selection.
- `vtkIdType = obj.GetPointMinimumMinValue ()` - Specify the minimum point id for point id selection.
- `vtkIdType = obj.GetPointMinimumMaxValue ()` - Specify the minimum point id for point id selection.
- `vtkIdType = obj.GetPointMinimum ()` - Specify the minimum point id for point id selection.
- `obj.SetPointMaximum (vtkIdType )` - Specify the maximum point id for point id selection.
- `vtkIdType = obj.GetPointMaximumMinValue ()` - Specify the maximum point id for point id selection.
- `vtkIdType = obj.GetPointMaximumMaxValue ()` - Specify the maximum point id for point id selection.
- `vtkIdType = obj.GetPointMaximum ()` - Specify the maximum point id for point id selection.
- `obj.SetCellMinimum (vtkIdType )` - Specify the minimum cell id for point id selection.
- `vtkIdType = obj.GetCellMinimumMinValue ()` - Specify the minimum cell id for point id selection.
- `vtkIdType = obj.GetCellMinimumMaxValue ()` - Specify the minimum cell id for point id selection.
- `vtkIdType = obj.GetCellMinimum ()` - Specify the minimum cell id for point id selection.
- `obj.SetCellMaximum (vtkIdType )` - Specify the maximum cell id for point id selection.
- `vtkIdType = obj.GetCellMaximumMinValue ()` - Specify the maximum cell id for point id selection.
- `vtkIdType = obj.GetCellMaximumMaxValue ()` - Specify the maximum cell id for point id selection.

- `vtkIdType = obj.GetCellMaximum ()` - Specify the maximum cell id for point id selection.
- `obj.SetExtent (double xMin, double xMax, double yMin, double yMax, double zMin, double zMax)` - Specify a (xmin,xmax, ymin,ymax, zmin,zmax) bounding box to clip data.
- `obj.SetExtent (double extent[6])` - Set / get a (xmin,xmax, ymin,ymax, zmin,zmax) bounding box to clip data.
- `obj.SetMerging (int )` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `int = obj.GetMerging ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.MergingOn ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.MergingOff ()` - Turn on/off merging of coincident points. Note that is merging is on, points with different point attributes (e.g., normals) are merged, which may cause rendering artifacts.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified.
- `long = obj.GetMTime ()` - Return the MTime also considering the locator.

33.252 vtkVectorDot

33.252.1 Usage

vtkVectorDot is a filter to generate scalar values from a dataset. The scalar value at a point is created by computing the dot product between the normal and vector at that point. Combined with the appropriate color map, this can show nodal lines/mode shapes of vibration, or a displacement plot.

To create an instance of class vtkVectorDot, simply invoke its constructor as follows

```
obj = vtkVectorDot
```

33.252.2 Methods

The class vtkVectorDot has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVectorDot class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVectorDot = obj.NewInstance ()`
- `vtkVectorDot = obj.SafeDownCast (vtkObject o)`
- `obj.SetScalarRange (double , double )` - Specify range to map scalars into.
- `obj.SetScalarRange (double  a[2])` - Specify range to map scalars into.
- `double = obj. GetScalarRange ()` - Get the range that scalars map into.

33.253 vtkVectorNorm

33.253.1 Usage

vtkVectorNorm is a filter that generates scalar values by computing Euclidean norm of vector triplets. Scalars can be normalized 0¡=s¡=1 if desired.

Note that this filter operates on point or cell attribute data, or both. By default, the filter operates on both point and cell data if vector point and cell data, respectively, are available from the input. Alternatively, you can choose to generate scalar norm values for just cell or point data.

To create an instance of class vtkVectorNorm, simply invoke its constructor as follows

```
obj = vtkVectorNorm
```

33.253.2 Methods

The class vtkVectorNorm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVectorNorm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVectorNorm = obj.NewInstance ()`
- `vtkVectorNorm = obj.SafeDownCast (vtkObject o)`
- `obj.SetNormalize (int )` - Specify whether to normalize scalar values.
- `int = obj.GetNormalize ()` - Specify whether to normalize scalar values.
- `obj.NormalizeOn ()` - Specify whether to normalize scalar values.
- `obj.NormalizeOff ()` - Specify whether to normalize scalar values.
- `obj.SetAttributeMode (int )` - Control how the filter works to generate scalar data from the input vector data. By default, (AttributeModeToDefault) the filter will generate the scalar norm for point and cell data (if vector data present in the input). Alternatively, you can explicitly set the filter to generate point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).
- `int = obj.GetAttributeMode ()` - Control how the filter works to generate scalar data from the input vector data. By default, (AttributeModeToDefault) the filter will generate the scalar norm for point and cell data (if vector data present in the input). Alternatively, you can explicitly set the filter to generate point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).
- `obj.SetAttributeModeToDefault ()` - Control how the filter works to generate scalar data from the input vector data. By default, (AttributeModeToDefault) the filter will generate the scalar norm for point and cell data (if vector data present in the input). Alternatively, you can explicitly set the filter to generate point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).
- `obj.SetAttributeModeToUsePointData ()` - Control how the filter works to generate scalar data from the input vector data. By default, (AttributeModeToDefault) the filter will generate the scalar norm for point and cell data (if vector data present in the input). Alternatively, you can explicitly set the filter to generate point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).
- `obj.SetAttributeModeToUseCellData ()` - Control how the filter works to generate scalar data from the input vector data. By default, (AttributeModeToDefault) the filter will generate the scalar norm for point and cell data (if vector data present in the input). Alternatively, you can explicitly set the filter to generate point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).

- `string = obj.GetAttributeModeAsString ()` - Control how the filter works to generate scalar data from the input vector data. By default, (AttributeModeToDefault) the filter will generate the scalar norm for point and cell data (if vector data present in the input). Alternatively, you can explicitly set the filter to generate point data (AttributeModeToUsePointData) or cell data (AttributeModeToUseCellData).

33.254 vtkVertexGlyphFilter

33.254.1 Usage

This filter throws away all of the cells in the input and replaces them with a vertex on each point. The intended use of this filter is roughly equivalent to the vtkGlyph3D filter, except this filter is specifically for data that has many vertices, making the rendered result faster and less cluttered than the glyph filter. This filter may take a graph or point set as input.

To create an instance of class vtkVertexGlyphFilter, simply invoke its constructor as follows

```
obj = vtkVertexGlyphFilter
```

33.254.2 Methods

The class vtkVertexGlyphFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVertexGlyphFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVertexGlyphFilter = obj.NewInstance ()`
- `vtkVertexGlyphFilter = obj.SafeDownCast (vtkObject o)`

33.255 vtkVoxelContoursToSurfaceFilter

33.255.1 Usage

vtkVoxelContoursToSurfaceFilter is a filter that takes contours and produces surfaces. There are some restrictions for the contours:

- The contours are input as vtkPolyData, with the contours being polys in the vtkPolyData. - The contours lie on XY planes - each contour has a constant Z - The contours are ordered in the polys of the vtkPolyData such that all contours on the first (lowest) XY plane are first, then continuing in order of increasing Z value. - The X, Y and Z coordinates are all integer values. - The desired sampling of the contour data is 1x1x1 - Aspect can be used to control the aspect ratio in the output polygonal dataset.

This filter takes the contours and produces a structured points dataset of signed floating point number indicating distance from a contour. A contouring filter is then applied to generate 3D surfaces from a stack of 2D contour distance slices. This is done in a streaming fashion so as not to use to much memory.

To create an instance of class vtkVoxelContoursToSurfaceFilter, simply invoke its constructor as follows

```
obj = vtkVoxelContoursToSurfaceFilter
```

33.255.2 Methods

The class vtkVoxelContoursToSurfaceFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVoxelContoursToSurfaceFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVoxelContoursToSurfaceFilter = obj.NewInstance ()`
- `vtkVoxelContoursToSurfaceFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetMemoryLimitInBytes (int )` - Set / Get the memory limit in bytes for this filter. This is the limit of the size of the structured points data set that is created for intermediate processing. The data will be streamed through this volume in as many pieces as necessary.
- `int = obj.GetMemoryLimitInBytes ()` - Set / Get the memory limit in bytes for this filter. This is the limit of the size of the structured points data set that is created for intermediate processing. The data will be streamed through this volume in as many pieces as necessary.
- `obj.SetSpacing (double , double , double )`
- `obj.SetSpacing (double  a[3])`
- `double = obj. GetSpacing ()`

33.256 vtkWarpLens

33.256.1 Usage

vtkWarpLens is a filter that modifies point coordinates by moving in accord with a lens distortion model.

To create an instance of class vtkWarpLens, simply invoke its constructor as follows

```
obj = vtkWarpLens
```

33.256.2 Methods

The class vtkWarpLens has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWarpLens class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWarpLens = obj.NewInstance ()`
- `vtkWarpLens = obj.SafeDownCast (vtkObject o)`
- `obj.SetKappa (double kappa)` - Specify second order symmetric radial lens distortion parameter. This is obsoleted by newer instance variables.
- `double = obj.GetKappa ()` - Specify second order symmetric radial lens distortion parameter. This is obsoleted by newer instance variables.
- `obj.SetCenter (double centerX, double centerY)` - Specify the center of radial distortion in pixels. This is obsoleted by newer instance variables.

- `double = obj.GetCenter ()` - Specify the center of radial distortion in pixels. This is obsoleted by newer instance variables.
- `obj.SetPrincipalPoint (double , double )` - Specify the calibrated principal point of the camera/lens
- `obj.SetPrincipalPoint (double a[2])` - Specify the calibrated principal point of the camera/lens
- `double = obj. GetPrincipalPoint ()` - Specify the calibrated principal point of the camera/lens
- `obj.SetK1 (double )` - Specify the symmetric radial distortion parameters for the lens
- `double = obj.GetK1 ()` - Specify the symmetric radial distortion parameters for the lens
- `obj.SetK2 (double )` - Specify the symmetric radial distortion parameters for the lens
- `double = obj.GetK2 ()` - Specify the symmetric radial distortion parameters for the lens
- `obj.SetP1 (double )` - Specify the decentering distortion parameters for the lens
- `double = obj.GetP1 ()` - Specify the decentering distortion parameters for the lens
- `obj.SetP2 (double )` - Specify the decentering distortion parameters for the lens
- `double = obj.GetP2 ()` - Specify the decentering distortion parameters for the lens
- `obj.SetFormatWidth (double )` - Specify the imager format width / height in mm
- `double = obj.GetFormatWidth ()` - Specify the imager format width / height in mm
- `obj.SetFormatHeight (double )` - Specify the imager format width / height in mm
- `double = obj.GetFormatHeight ()` - Specify the imager format width / height in mm
- `obj.SetImageWidth (int )` - Specify the image width / height in pixels
- `int = obj.GetImageWidth ()` - Specify the image width / height in pixels
- `obj.SetImageHeight (int )` - Specify the image width / height in pixels
- `int = obj.GetImageHeight ()` - Specify the image width / height in pixels

33.257 vtkWarpScalar

33.257.1 Usage

vtkWarpScalar is a filter that modifies point coordinates by moving points along point normals by the scalar amount times the scale factor. Useful for creating carpet or x-y-z plots.

If normals are not present in data, the Normal instance variable will be used as the direction along which to warp the geometry. If normals are present but you would like to use the Normal instance variable, set the UseNormal boolean to true.

If XYPlane boolean is set true, then the z-value is considered to be a scalar value (still scaled by scale factor), and the displacement is along the z-axis. If scalars are also present, these are copied through and can be used to color the surface.

Note that the filter passes both its point data and cell data to its output, except for normals, since these are distorted by the warping.

To create an instance of class vtkWarpScalar, simply invoke its constructor as follows

```
obj = vtkWarpScalar
```

33.257.2 Methods

The class vtkWarpScalar has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWarpScalar class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWarpScalar = obj.NewInstance ()`
- `vtkWarpScalar = obj.SafeDownCast (vtkObject o)`
- `obj.SetScaleFactor (double )` - Specify value to scale displacement.
- `double = obj.GetScaleFactor ()` - Specify value to scale displacement.
- `obj.SetUseNormal (int )` - Turn on/off use of user specified normal. If on, data normals will be ignored and instance variable Normal will be used instead.
- `int = obj.GetUseNormal ()` - Turn on/off use of user specified normal. If on, data normals will be ignored and instance variable Normal will be used instead.
- `obj.UseNormalOn ()` - Turn on/off use of user specified normal. If on, data normals will be ignored and instance variable Normal will be used instead.
- `obj.UseNormalOff ()` - Turn on/off use of user specified normal. If on, data normals will be ignored and instance variable Normal will be used instead.
- `obj.SetNormal (double , double , double )` - Normal (i.e., direction) along which to warp geometry. Only used if UseNormal boolean set to true or no normals available in data.
- `obj.SetNormal (double  a[3])` - Normal (i.e., direction) along which to warp geometry. Only used if UseNormal boolean set to true or no normals available in data.
- `double = obj. GetNormal ()` - Normal (i.e., direction) along which to warp geometry. Only used if UseNormal boolean set to true or no normals available in data.
- `obj.SetXYPlane (int )` - Turn on/off flag specifying that input data is x-y plane. If x-y plane, then the z value is used to warp the surface in the z-axis direction (times the scale factor) and scalars are used to color the surface.
- `int = obj.GetXYPlane ()` - Turn on/off flag specifying that input data is x-y plane. If x-y plane, then the z value is used to warp the surface in the z-axis direction (times the scale factor) and scalars are used to color the surface.
- `obj.XYPlaneOn ()` - Turn on/off flag specifying that input data is x-y plane. If x-y plane, then the z value is used to warp the surface in the z-axis direction (times the scale factor) and scalars are used to color the surface.
- `obj.XYPlaneOff ()` - Turn on/off flag specifying that input data is x-y plane. If x-y plane, then the z value is used to warp the surface in the z-axis direction (times the scale factor) and scalars are used to color the surface.

33.258 vtkWarpTo

33.258.1 Usage

vtkWarpTo is a filter that modifies point coordinates by moving the points towards a user specified position.

To create an instance of class vtkWarpTo, simply invoke its constructor as follows

```
obj = vtkWarpTo
```

33.258.2 Methods

The class vtkWarpTo has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWarpTo class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWarpTo = obj.NewInstance ()`
- `vtkWarpTo = obj.SafeDownCast (vtkObject o)`
- `obj.SetScaleFactor (double )` - Set/Get the value to scale displacement.
- `double = obj.GetScaleFactor ()` - Set/Get the value to scale displacement.
- `double = obj. GetPosition ()` - Set/Get the position to warp towards.
- `obj.SetPosition (double , double , double )` - Set/Get the position to warp towards.
- `obj.SetPosition (double  a[3])` - Set/Get the position to warp towards.
- `obj.SetAbsolute (int )` - Set/Get the Absolute ivar. Turning Absolute on causes scale factor of the new position to be one unit away from Position.
- `int = obj.GetAbsolute ()` - Set/Get the Absolute ivar. Turning Absolute on causes scale factor of the new position to be one unit away from Position.
- `obj.AbsoluteOn ()` - Set/Get the Absolute ivar. Turning Absolute on causes scale factor of the new position to be one unit away from Position.
- `obj.AbsoluteOff ()` - Set/Get the Absolute ivar. Turning Absolute on causes scale factor of the new position to be one unit away from Position.

33.259 vtkWarpVector

33.259.1 Usage

vtkWarpVector is a filter that modifies point coordinates by moving points along vector times the scale factor. Useful for showing flow profiles or mechanical deformation.

The filter passes both its point data and cell data to its output.

To create an instance of class vtkWarpVector, simply invoke its constructor as follows

```
obj = vtkWarpVector
```

33.259.2 Methods

The class vtkWarpVector has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWarpVector class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWarpVector = obj.NewInstance ()`
- `vtkWarpVector = obj.SafeDownCast (vtkObject o)`
- `obj.SetScaleFactor (double )` - Specify value to scale displacement.
- `double = obj.GetScaleFactor ()` - Specify value to scale displacement.

33.260 vtkWindowedSincPolyDataFilter

33.260.1 Usage

vtkWindowedSincPolyDataFiler adjust point coordinate using a windowed sinc function interpolation kernel. The effect is to "relax" the mesh, making the cells better shaped and the vertices more evenly distributed. Note that this filter operates the lines, polygons, and triangle strips composing an instance of vtkPolyData. Vertex or poly-vertex cells are never modified.

The algorithm proceeds as follows. For each vertex v, a topological and geometric analysis is performed to determine which vertices are connected to v, and which cells are connected to v. Then, a connectivity array is constructed for each vertex. (The connectivity array is a list of lists of vertices that directly attach to each vertex.) Next, an iteration phase begins over all vertices. For each vertex v, the coordinates of v are modified using a windowed sinc function interpolation kernel. Taubin describes this methodology is the IBM tech report RC-20404 (#90237, dated 3/12/96) "Optimal Surface Smoothing as Filter Design" G. Taubin, T. Zhang and G. Golub. (Zhang and Golub are at Stanford University).

This report discusses using standard signal processing low-pass filters (in particular windowed sinc functions) to smooth polyhedra. The transfer functions of the low-pass filters are approximated by Chebyshev polynomials. This facilitates applying the filters in an iterative diffusion process (as opposed to a kernel convolution). The more smoothing iterations applied, the higher the degree of polynomial approximating the low-pass filter transfer function. Each smoothing iteration, therefore, applies the next higher term of the Chebyshev filter approximation to the polyhedron. This decoupling of the filter into an iteratively applied polynomial is possible since the Chebyshev polynomials are orthogonal, i.e. increasing the order of the approximation to the filter transfer function does not alter the previously calculated coefficients for the low order terms.

Note: Care must be taken to avoid smoothing with too few iterations. A Chebyshev approximation with too few terms is an poor approximation. The first few smoothing iterations represent a severe scaling and translation of the data. Subsequent iterations cause the smoothed polyhedron to converge to the true location and scale of the object. We have attempted to protect against this by automatically adjusting the filter, effectively widening the pass band. This adjustment is only possible if the number of iterations is greater than 1. Note that this sacrifices some degree of smoothing for model integrity. For those interested, the filter is adjusted by searching for a value sigma such that the actual pass band is k_pb + sigma and such that the filter transfer function evaluates to unity at k_pb, i.e. f(k_pb) = 1

To improve the numerical stability of the solution and minimize the scaling the translation effects, the algorithm can translate and scale the position coordinates to within the unit cube [-1, 1], perform the smoothing, and translate and scale the position coordinates back to the original coordinate frame. This mode is controlled with the NormalizeCoordinatesOn() / NormalizeCoordinatesOff() methods. For legacy reasons, the default is NormalizeCoordinatesOff.

This implementation is currently limited to using an interpolation kernel based on Hamming windows. Other windows (such as Hann, Blackman, Kaiser, Lanczos, Gaussian, and exponential windows) could be used instead.

There are some special instance variables used to control the execution of this filter. (These ivars basically control what vertices can be smoothed, and the creation of the connectivity array.) The BoundarySmoothing ivar enables/disables the smoothing operation on vertices that are on the "boundary" of the mesh. A boundary vertex is one that is surrounded by a semi-cycle of polygons (or used by a single line).

Another important ivar is FeatureEdgeSmoothing. If this ivar is enabled, then interior vertices are classified as either "simple", "interior edge", or "fixed", and smoothed differently. (Interior vertices are manifold vertices surrounded by a cycle of polygons; or used by two line cells.) The classification is based on the number of feature edges attached to v. A feature edge occurs when the angle between the two surface normals of a polygon sharing an edge is greater than the FeatureAngle ivar. Then, vertices used by no feature edges are classified "simple", vertices used by exactly two feature edges are classified "interior edge", and all others are "fixed" vertices.

Once the classification is known, the vertices are smoothed differently. Corner (i.e., fixed) vertices are not smoothed at all. Simple vertices are smoothed as before . Interior edge vertices are smoothed only along their two connected edges, and only if the angle between the edges is less than the EdgeAngle ivar.

The total smoothing can be controlled by using two ivars. The NumberOfIterations determines the maximum number of smoothing passes. The NumberOfIterations corresponds to the degree of the polynomial that is used to approximate the windowed sinc function. Ten or twenty iterations is all the is usually necessary. Contrast this with vtkSmoothPolyDataFilter which usually requires 100 to 200 smoothing iterations. vtkSmoothPolyDataFilter is also not an approximation to an ideal low-pass filter, which can cause the geometry to shrink as the amount of smoothing increases.

The second ivar is the specification of the PassBand for the windowed sinc filter. By design, the PassBand is specified as a doubleing point number between 0 and 2. Lower PassBand values produce more smoothing. A good default value for the PassBand is 0.1 (for those interested, the PassBand (and frequencies) for PolyData are based on the valence of the vertices, this limits all the frequency modes in a polyhedral mesh to between 0 and 2.)

There are two instance variables that control the generation of error data. If the ivar GenerateErrorScalars is on, then a scalar value indicating the distance of each vertex from its original position is computed. If the ivar GenerateErrorVectors is on, then a vector representing change in position is computed.

To create an instance of class vtkWindowedSincPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkWindowedSincPolyDataFilter
```

33.260.2 Methods

The class vtkWindowedSincPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWindowedSincPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWindowedSincPolyDataFilter = obj.NewInstance ()`
- `vtkWindowedSincPolyDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfIterations (int )` - Specify the number of iterations (or degree of the polynomial approximating the windowed sinc function).
- `int = obj.GetNumberOfIterationsMinValue ()` - Specify the number of iterations (or degree of the polynomial approximating the windowed sinc function).
- `int = obj.GetNumberOfIterationsMaxValue ()` - Specify the number of iterations (or degree of the polynomial approximating the windowed sinc function).

- `int = obj.GetNumberOfIterations ()` - Specify the number of iterations (or degree of the polynomial approximating the windowed sinc function).
- `obj.SetPassBand (double )` - Set the passband value for the windowed sinc filter
- `double = obj.GetPassBandMinValue ()` - Set the passband value for the windowed sinc filter
- `double = obj.GetPassBandMaxValue ()` - Set the passband value for the windowed sinc filter
- `double = obj.GetPassBand ()` - Set the passband value for the windowed sinc filter
- `obj.SetNormalizeCoordinates (int )` - Turn on/off coordinate normalization. The positions can be translated and scaled such that they fit within a [-1, 1] prior to the smoothing computation. The default is off. The numerical stability of the solution can be improved by turning normalization on. If normalization is on, the coordinates will be rescaled to the original coordinate system after smoothing has completed.
- `int = obj.GetNormalizeCoordinates ()` - Turn on/off coordinate normalization. The positions can be translated and scaled such that they fit within a [-1, 1] prior to the smoothing computation. The default is off. The numerical stability of the solution can be improved by turning normalization on. If normalization is on, the coordinates will be rescaled to the original coordinate system after smoothing has completed.
- `obj.NormalizeCoordinatesOn ()` - Turn on/off coordinate normalization. The positions can be translated and scaled such that they fit within a [-1, 1] prior to the smoothing computation. The default is off. The numerical stability of the solution can be improved by turning normalization on. If normalization is on, the coordinates will be rescaled to the original coordinate system after smoothing has completed.
- `obj.NormalizeCoordinatesOff ()` - Turn on/off coordinate normalization. The positions can be translated and scaled such that they fit within a [-1, 1] prior to the smoothing computation. The default is off. The numerical stability of the solution can be improved by turning normalization on. If normalization is on, the coordinates will be rescaled to the original coordinate system after smoothing has completed.
- `obj.SetFeatureEdgeSmoothing (int )` - Turn on/off smoothing along sharp interior edges.
- `int = obj.GetFeatureEdgeSmoothing ()` - Turn on/off smoothing along sharp interior edges.
- `obj.FeatureEdgeSmoothingOn ()` - Turn on/off smoothing along sharp interior edges.
- `obj.FeatureEdgeSmoothingOff ()` - Turn on/off smoothing along sharp interior edges.
- `obj.SetFeatureAngle (double )` - Specify the feature angle for sharp edge identification.
- `double = obj.GetFeatureAngleMinValue ()` - Specify the feature angle for sharp edge identification.
- `double = obj.GetFeatureAngleMaxValue ()` - Specify the feature angle for sharp edge identification.
- `double = obj.GetFeatureAngle ()` - Specify the feature angle for sharp edge identification.
- `obj.SetEdgeAngle (double )` - Specify the edge angle to control smoothing along edges (either interior or boundary).
- `double = obj.GetEdgeAngleMinValue ()` - Specify the edge angle to control smoothing along edges (either interior or boundary).
- `double = obj.GetEdgeAngleMaxValue ()` - Specify the edge angle to control smoothing along edges (either interior or boundary).
- `double = obj.GetEdgeAngle ()` - Specify the edge angle to control smoothing along edges (either interior or boundary).

- `obj.SetBoundarySmoothing (int )` - Turn on/off the smoothing of vertices on the boundary of the mesh.
- `int = obj.GetBoundarySmoothing ()` - Turn on/off the smoothing of vertices on the boundary of the mesh.
- `obj.BoundarySmoothingOn ()` - Turn on/off the smoothing of vertices on the boundary of the mesh.
- `obj.BoundarySmoothingOff ()` - Turn on/off the smoothing of vertices on the boundary of the mesh.
- `obj.SetNonManifoldSmoothing (int )` - Smooth non-manifold vertices.
- `int = obj.GetNonManifoldSmoothing ()` - Smooth non-manifold vertices.
- `obj.NonManifoldSmoothingOn ()` - Smooth non-manifold vertices.
- `obj.NonManifoldSmoothingOff ()` - Smooth non-manifold vertices.
- `obj.SetGenerateErrorScalars (int )` - Turn on/off the generation of scalar distance values.
- `int = obj.GetGenerateErrorScalars ()` - Turn on/off the generation of scalar distance values.
- `obj.GenerateErrorScalarsOn ()` - Turn on/off the generation of scalar distance values.
- `obj.GenerateErrorScalarsOff ()` - Turn on/off the generation of scalar distance values.
- `obj.SetGenerateErrorVectors (int )` - Turn on/off the generation of error vectors.
- `int = obj.GetGenerateErrorVectors ()` - Turn on/off the generation of error vectors.
- `obj.GenerateErrorVectorsOn ()` - Turn on/off the generation of error vectors.
- `obj.GenerateErrorVectorsOff ()` - Turn on/off the generation of error vectors.

33.261 vtkYoungsMaterialInterface

33.261.1 Usage

Reconstructs material interfaces from a mesh containing mixed cells (where several materials are mixed) this implementation is based on the youngs algorithm, generalized to arbitrary cell types and works on both 2D and 3D meshes. the main advantage of the youngs algorithm is it guarantees the material volume correctness. for 2D meshes, the AxisSymetric flag allows to switch between a pure 2D (plannar) algorithm and an axis symetric 2D algorithm handling volumes of revolution.

.SECTION Thanks This file is part of the generalized Youngs material interface reconstruction algorithm contributed by ¡br¿ CEA/DIF - Commissariat a l'Energie Atomique, Centre DAM Ile-De-France ¡br¿ BP12, F-91297 Arpajon, France. ¡br¿ Implementation by Thierry Carrard (thierry.carrard@cea.fr)

To create an instance of class vtkYoungsMaterialInterface, simply invoke its constructor as follows

```
obj = vtkYoungsMaterialInterface
```

33.261.2 Methods

The class vtkYoungsMaterialInterface has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkYoungsMaterialInterface class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkYoungsMaterialInterface = obj.NewInstance ()`
- `vtkYoungsMaterialInterface = obj.SafeDownCast (vtkObject o)`
- `obj.SetInverseNormal (int )` - Set/Get wether the normal vector has to be flipped.
- `int = obj.GetInverseNormal ()` - Set/Get wether the normal vector has to be flipped.
- `obj.InverseNormalOn ()` - Set/Get wether the normal vector has to be flipped.
- `obj.InverseNormalOff ()` - Set/Get wether the normal vector has to be flipped.
- `obj.SetReverseMaterialOrder (int )` - If this flag is on, material order in reversed. Otherwise, materials are sorted in ascending order depending on the given ordering array.
- `int = obj.GetReverseMaterialOrder ()` - If this flag is on, material order in reversed. Otherwise, materials are sorted in ascending order depending on the given ordering array.
- `obj.ReverseMaterialOrderOn ()` - If this flag is on, material order in reversed. Otherwise, materials are sorted in ascending order depending on the given ordering array.
- `obj.ReverseMaterialOrderOff ()` - If this flag is on, material order in reversed. Otherwise, materials are sorted in ascending order depending on the given ordering array.
- `obj.SetOnionPeel (int )` - Set/Get OnionPeel flag. if this flag is on, the normal vector of the first material (which depends on material ordering) is used for all materials.
- `int = obj.GetOnionPeel ()` - Set/Get OnionPeel flag. if this flag is on, the normal vector of the first material (which depends on material ordering) is used for all materials.
- `obj.OnionPeelOn ()` - Set/Get OnionPeel flag. if this flag is on, the normal vector of the first material (which depends on material ordering) is used for all materials.
- `obj.OnionPeelOff ()` - Set/Get OnionPeel flag. if this flag is on, the normal vector of the first material (which depends on material ordering) is used for all materials.
- `obj.SetAxisSymetric (int )` - Turns on/off AxisSymetric computation of 2D interfaces. in axis symetric mode, 2D meshes are understood as volumes of revolution.
- `int = obj.GetAxisSymetric ()` - Turns on/off AxisSymetric computation of 2D interfaces. in axis symetric mode, 2D meshes are understood as volumes of revolution.
- `obj.AxisSymetricOn ()` - Turns on/off AxisSymetric computation of 2D interfaces. in axis symetric mode, 2D meshes are understood as volumes of revolution.
- `obj.AxisSymetricOff ()` - Turns on/off AxisSymetric computation of 2D interfaces. in axis symetric mode, 2D meshes are understood as volumes of revolution.
- `obj.SetUseFractionAsDistance (int )` - when UseFractionAsDistance is true, the volume fraction is interpreted as the distance of the cutting plane from the origin. in axis symetric mode, 2D meshes are understood as volumes of revolution.
- `int = obj.GetUseFractionAsDistance ()` - when UseFractionAsDistance is true, the volume fraction is interpreted as the distance of the cutting plane from the origin. in axis symetric mode, 2D meshes are understood as volumes of revolution.
- `obj.UseFractionAsDistanceOn ()` - when UseFractionAsDistance is true, the volume fraction is interpreted as the distance of the cutting plane from the origin. in axis symetric mode, 2D meshes are understood as volumes of revolution.

- `obj.UseFractionAsDistanceOff ()` - when UseFractionAsDistance is true, the volume fraction is interpreted as the distance of the cutting plane from the origin. in axis symetric mode, 2D meshes are understood as volumes of revolution.
- `obj.SetFillMaterial (int )` - When FillMaterial is set to 1, the volume containing material is output and not only the interface surface.
- `int = obj.GetFillMaterial ()` - When FillMaterial is set to 1, the volume containing material is output and not only the interface surface.
- `obj.FillMaterialOn ()` - When FillMaterial is set to 1, the volume containing material is output and not only the interface surface.
- `obj.FillMaterialOff ()` - When FillMaterial is set to 1, the volume containing material is output and not only the interface surface.
- `obj.SetTwoMaterialsOptimization (int )` - Triggers some additional optimizations for cells containing only two materials. This option might produce different result than expected if the sum of volume fractions is not 1.
- `int = obj.GetTwoMaterialsOptimization ()` - Triggers some additional optimizations for cells containing only two materials. This option might produce different result than expected if the sum of volume fractions is not 1.
- `obj.TwoMaterialsOptimizationOn ()` - Triggers some additional optimizations for cells containing only two materials. This option might produce different result than expected if the sum of volume fractions is not 1.
- `obj.TwoMaterialsOptimizationOff ()` - Triggers some additional optimizations for cells containing only two materials. This option might produce different result than expected if the sum of volume fractions is not 1.
- `obj.SetVolumeFractionRange (double , double )` - Set/Get minimum and maximum volume fraction value. if a material fills a volume above the minimum value, the material is considered to be void. if a material fills a volume fraction beyond the maximum value it is considered as filling the whole volume.
- `obj.SetVolumeFractionRange (double a[2])` - Set/Get minimum and maximum volume fraction value. if a material fills a volume above the minimum value, the material is considered to be void. if a material fills a volume fraction beyond the maximum value it is considered as filling the whole volume.
- `double = obj. GetVolumeFractionRange ()` - Set/Get minimum and maximum volume fraction value. if a material fills a volume above the minimum value, the material is considered to be void. if a material fills a volume fraction beyond the maximum value it is considered as filling the whole volume.
- `obj.SetNumberOfMaterials (int n)` - Sets/Gets the number of materials.
- `int = obj.GetNumberOfMaterials ()` - Sets/Gets the number of materials.
- `obj.SetMaterialArrays (int i, string volumeFraction, string interfaceNormal, string materialOrderir` - Set ith Material arrays to be used as volume fraction, interface normal and material ordering. Each parameter name a cell array.
- `obj.SetMaterialVolumeFractionArray (int i, string volume)` - Set ith Material arrays to be used as volume fraction, interface normal and material ordering. Each parameter name a cell array.
- `obj.SetMaterialNormalArray (int i, string normal)` - Set ith Material arrays to be used as volume fraction, interface normal and material ordering. Each parameter name a cell array.

- `obj.SetMaterialOrderingArray (int i, string ordering)` - Set ith Material arrays to be used as volume fraction, interface normal and material ordering. Each parameter name a cell array.
- `obj.RemoveAllMaterials ()` - Removes all meterials previously added.

Chapter 34

Visualization Toolkit Hybrid Classes

34.1 vtk3DSImporter

34.1.1 Usage

vtk3DSImporter imports 3D Studio files into vtk.

To create an instance of class vtk3DSImporter, simply invoke its constructor as follows

```
obj = vtk3DSImporter
```

34.1.2 Methods

The class vtk3DSImporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtk3DSImporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtk3DSImporter = obj.NewInstance ()`
- `vtk3DSImporter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify the name of the file to read.
- `string = obj.GetFileName ()` - Specify the name of the file to read.
- `obj.SetComputeNormals (int )` - Set/Get the computation of normals. If on, imported geometry will be run through vtkPolyDataNormals.
- `int = obj.GetComputeNormals ()` - Set/Get the computation of normals. If on, imported geometry will be run through vtkPolyDataNormals.
- `obj.ComputeNormalsOn ()` - Set/Get the computation of normals. If on, imported geometry will be run through vtkPolyDataNormals.
- `obj.ComputeNormalsOff ()` - Set/Get the computation of normals. If on, imported geometry will be run through vtkPolyDataNormals.

34.2 vtkAnnotatedCubeActor

34.2.1 Usage

vtkAnnotatedCubeActor is a hybrid 3D actor used to represent an anatomical orientation marker in a scene. The class consists of a 3D unit cube centered on the origin with each face labelled in correspondance to a particular coordinate direction. For example, with Cartesian directions, the user defined text labels could be: +X, -X, +Y, -Y, +Z, -Z, while for anatomical directions: A, P, L, R, S, I. Text is automatically centered on each cube face and is not restriceted to single characters. In addition to or in replace of a solid text label representation, the outline edges of the labels can be displayed. The individual properties of the cube, face labels and text outlines can be manipulated as can their visibility.

To create an instance of class vtkAnnotatedCubeActor, simply invoke its constructor as follows

```
obj = vtkAnnotatedCubeActor
```

34.2.2 Methods

The class vtkAnnotatedCubeActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAnnotatedCubeActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAnnotatedCubeActor = obj.NewInstance ()`
- `vtkAnnotatedCubeActor = obj.SafeDownCast (vtkObject o)`
- `obj.GetActors (vtkPropCollection )` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of an axes actor. Overloads the virtual vtkProp method.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.GetBounds (double bounds[6])` - Get the bounds for this Actor as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax). (The method GetBounds(double bounds[6]) is available from the superclass.)
- `double = obj.GetBounds ()` - Get the bounds for this Actor as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax). (The method GetBounds(double bounds[6]) is available from the superclass.)
- `long = obj.GetMTime ()` - Get the actors mtime plus consider its properties and texture if set.
- `obj.SetFaceTextScale (double )` - Set/Get the scale factor for the face text
- `double = obj.GetFaceTextScale ()` - Set/Get the scale factor for the face text

- `vtkProperty = obj.GetXPlusFaceProperty ()` - Get the individual face text properties.
- `vtkProperty = obj.GetXMinusFaceProperty ()` - Get the individual face text properties.
- `vtkProperty = obj.GetYPlusFaceProperty ()` - Get the individual face text properties.
- `vtkProperty = obj.GetYMinusFaceProperty ()` - Get the individual face text properties.
- `vtkProperty = obj.GetZPlusFaceProperty ()` - Get the individual face text properties.
- `vtkProperty = obj.GetZMinusFaceProperty ()` - Get the individual face text properties.
- `vtkProperty = obj.GetCubeProperty ()` - Get the cube properties.
- `vtkProperty = obj.GetTextEdgesProperty ()` - Get the text edges properties.
- `obj.SetXPlusFaceText (string )` - Set/get the face text.
- `string = obj.GetXPlusFaceText ()` - Set/get the face text.
- `obj.SetXMinusFaceText (string )` - Set/get the face text.
- `string = obj.GetXMinusFaceText ()` - Set/get the face text.
- `obj.SetYPlusFaceText (string )` - Set/get the face text.
- `string = obj.GetYPlusFaceText ()` - Set/get the face text.
- `obj.SetYMinusFaceText (string )` - Set/get the face text.
- `string = obj.GetYMinusFaceText ()` - Set/get the face text.
- `obj.SetZPlusFaceText (string )` - Set/get the face text.
- `string = obj.GetZPlusFaceText ()` - Set/get the face text.
- `obj.SetZMinusFaceText (string )` - Set/get the face text.
- `string = obj.GetZMinusFaceText ()` - Set/get the face text.
- `obj.SetTextEdgesVisibility (int )` - Enable/disable drawing the vector text edges.
- `int = obj.GetTextEdgesVisibility ()` - Enable/disable drawing the vector text edges.
- `obj.SetCubeVisibility (int )` - Enable/disable drawing the cube.
- `int = obj.GetCubeVisibility ()` - Enable/disable drawing the cube.
- `obj.SetFaceTextVisibility (int )` - Enable/disable drawing the vector text.
- `int = obj.GetFaceTextVisibility ()` - Enable/disable drawing the vector text.
- `obj.SetXFaceTextRotation (double )` - Augment individual face text orientations.
- `double = obj.GetXFaceTextRotation ()` - Augment individual face text orientations.
- `obj.SetYFaceTextRotation (double )` - Augment individual face text orientations.
- `double = obj.GetYFaceTextRotation ()` - Augment individual face text orientations.
- `obj.SetZFaceTextRotation (double )` - Augment individual face text orientations.
- `double = obj.GetZFaceTextRotation ()` - Augment individual face text orientations.
- `vtkAssembly = obj.GetAssembly ()`

34.3 vtkArcPlotter

34.3.1 Usage

vtkArcPlotter performs plotting of attribute data along polylines defined with an input vtkPolyData data object. Any type of attribute data can be plotted including scalars, vectors, tensors, normals, texture coordinates, and field data. Either one or multiple data components can be plotted.

To use this class you must specify an input data set that contains one or more polylines, and some attribute data including which component of the attribute data. (By default, this class processes the first component of scalar data.) You will also need to set an offset radius (the distance of the polyline to the median line of the plot), a width for the plot (the distance that the minimum and maximum plot values are mapped into), an possibly an offset (used to offset attribute data with multiple components).

Normally the filter automatically computes normals for generating the offset arc plot. However, you can specify a default normal and use that instead.

To create an instance of class vtkArcPlotter, simply invoke its constructor as follows

```
obj = vtkArcPlotter
```

34.3.2 Methods

The class vtkArcPlotter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArcPlotter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArcPlotter = obj.NewInstance ()`
- `vtkArcPlotter = obj.SafeDownCast (vtkObject o)`
- `obj.SetCamera (vtkCamera )` - Specify a camera used to orient the plot along the arc. If no camera is specified, then the orientation of the plot is arbitrary.
- `vtkCamera = obj.GetCamera ()` - Specify a camera used to orient the plot along the arc. If no camera is specified, then the orientation of the plot is arbitrary.
- `obj.SetPlotMode (int )` - Specify which data to plot: scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetPlotComponent to control which component to plot.
- `int = obj.GetPlotMode ()` - Specify which data to plot: scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetPlotComponent to control which component to plot.
- `obj.SetPlotModeToPlotScalars ()` - Specify which data to plot: scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetPlotComponent to control which component to plot.
- `obj.SetPlotModeToPlotVectors ()` - Specify which data to plot: scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetPlotComponent to control which component to plot.
- `obj.SetPlotModeToPlotNormals ()` - Specify which data to plot: scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetPlotComponent to control which component to plot.

- `obj.SetPlotModeToPlotTCoords ()` - Specify which data to plot: scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetPlotComponent to control which component to plot.
- `obj.SetPlotModeToPlotTensors ()` - Specify which data to plot: scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetPlotComponent to control which component to plot.
- `obj.SetPlotModeToPlotFieldData ()` - Specify which data to plot: scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetPlotComponent to control which component to plot.
- `obj.SetPlotComponent (int )` - Set/Get the component number to plot if the data has more than one component. If the value of the plot component is == (-1), then all the components will be plotted.
- `int = obj.GetPlotComponent ()` - Set/Get the component number to plot if the data has more than one component. If the value of the plot component is == (-1), then all the components will be plotted.
- `obj.SetRadius (double )` - Set the radius of the "median" value of the first plotted component.
- `double = obj.GetRadiusMinValue ()` - Set the radius of the "median" value of the first plotted component.
- `double = obj.GetRadiusMaxValue ()` - Set the radius of the "median" value of the first plotted component.
- `double = obj.GetRadius ()` - Set the radius of the "median" value of the first plotted component.
- `obj.SetHeight (double )` - Set the height of the plot. (The radius combined with the height define the location of the plot relative to the generating polyline.)
- `double = obj.GetHeightMinValue ()` - Set the height of the plot. (The radius combined with the height define the location of the plot relative to the generating polyline.)
- `double = obj.GetHeightMaxValue ()` - Set the height of the plot. (The radius combined with the height define the location of the plot relative to the generating polyline.)
- `double = obj.GetHeight ()` - Set the height of the plot. (The radius combined with the height define the location of the plot relative to the generating polyline.)
- `obj.SetOffset (double )` - Specify an offset that translates each subsequent plot (if there is more than one component plotted) from the defining arc (i.e., polyline).
- `double = obj.GetOffsetMinValue ()` - Specify an offset that translates each subsequent plot (if there is more than one component plotted) from the defining arc (i.e., polyline).
- `double = obj.GetOffsetMaxValue ()` - Specify an offset that translates each subsequent plot (if there is more than one component plotted) from the defining arc (i.e., polyline).
- `double = obj.GetOffset ()` - Specify an offset that translates each subsequent plot (if there is more than one component plotted) from the defining arc (i.e., polyline).
- `obj.SetUseDefaultNormal (int )` - Set a boolean to control whether to use default normals. By default, normals are automatically computed from the generating polyline and camera.
- `int = obj.GetUseDefaultNormal ()` - Set a boolean to control whether to use default normals. By default, normals are automatically computed from the generating polyline and camera.
- `obj.UseDefaultNormalOn ()` - Set a boolean to control whether to use default normals. By default, normals are automatically computed from the generating polyline and camera.

- `obj.UseDefaultNormalOff ()` - Set a boolean to control whether to use default normals. By default, normals are automatically computed from the generating polyline and camera.
- `obj.SetDefaultNormal (float , float , float )` - Set the default normal to use if you do not wish automatic normal calculation. The arc plot will be generated using this normal.
- `obj.SetDefaultNormal (float a[3])` - Set the default normal to use if you do not wish automatic normal calculation. The arc plot will be generated using this normal.
- `float = obj. GetDefaultNormal ()` - Set the default normal to use if you do not wish automatic normal calculation. The arc plot will be generated using this normal.
- `obj.SetFieldDataArray (int )` - Set/Get the field data array to plot. This instance variable is only applicable if field data is plotted.
- `int = obj.GetFieldDataArrayMinValue ()` - Set/Get the field data array to plot. This instance variable is only applicable if field data is plotted.
- `int = obj.GetFieldDataArrayMaxValue ()` - Set/Get the field data array to plot. This instance variable is only applicable if field data is plotted.
- `int = obj.GetFieldDataArray ()` - Set/Get the field data array to plot. This instance variable is only applicable if field data is plotted.
- `long = obj.GetMTime ()` - New GetMTime because of camera dependency.

34.4 vtkAxesActor

34.4.1 Usage

vtkAxesActor is a hybrid 2D/3D actor used to represent 3D axes in a scene. The user can define the geometry to use for the shaft or the tip, and the user can set the text for the three axes. The text will appear to follow the camera since it is implemented by means of vtkCaptionActor2D. All of the functionality of the underlying vtkCaptionActor2D objects are accessable so that, for instance, the font attributes of the axes text can be manipulated through vtkTextProperty. Since this class inherits from vtkProp3D, one can apply a user transform to the underlying geometry and the positioning of the labels. For example, a rotation transform could be used to generate a left-handed axes representation.

To create an instance of class vtkAxesActor, simply invoke its constructor as follows

```
obj = vtkAxesActor
```

34.4.2 Methods

The class vtkAxesActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAxesActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAxesActor = obj.NewInstance ()`
- `vtkAxesActor = obj.SafeDownCast (vtkObject o)`
- `obj.GetActors (vtkPropCollection )` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.

- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of an axes actor. Overloads the virtual vtkProp method.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.GetBounds (double bounds[6])` - Get the bounds for this Actor as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax). (The method GetBounds(double bounds[6]) is available from the superclass.)
- `double = obj.GetBounds ()` - Get the bounds for this Actor as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax). (The method GetBounds(double bounds[6]) is available from the superclass.)
- `long = obj.GetMTime ()` - Get the actors mtime plus consider its properties and texture if set.
- `long = obj.GetRedrawMTime ()` - Return the mtime of anything that would cause the rendered image to appear differently. Usually this involves checking the mtime of the prop plus anything else it depends on such as properties, textures etc.
- `obj.SetTotalLength (double v[3])` - Set the total length of the axes in 3 dimensions.
- `obj.SetTotalLength (double x, double y, double z)` - Set the total length of the axes in 3 dimensions.
- `double = obj. GetTotalLength ()` - Set the total length of the axes in 3 dimensions.
- `obj.SetNormalizedShaftLength (double v[3])` - Set the normalized (0-1) length of the shaft.
- `obj.SetNormalizedShaftLength (double x, double y, double z)` - Set the normalized (0-1) length of the shaft.
- `double = obj. GetNormalizedShaftLength ()` - Set the normalized (0-1) length of the shaft.
- `obj.SetNormalizedTipLength (double v[3])` - Set the normalized (0-1) length of the tip. Normally, this would be 1 - the normalized length of the shaft.
- `obj.SetNormalizedTipLength (double x, double y, double z)` - Set the normalized (0-1) length of the tip. Normally, this would be 1 - the normalized length of the shaft.
- `double = obj. GetNormalizedTipLength ()` - Set the normalized (0-1) length of the tip. Normally, this would be 1 - the normalized length of the shaft.
- `obj.SetNormalizedLabelPosition (double v[3])` - Set the normalized (0-1) position of the label along the length of the shaft. A value ¿ 1 is permissible.
- `obj.SetNormalizedLabelPosition (double x, double y, double z)` - Set the normalized (0-1) position of the label along the length of the shaft. A value ¿ 1 is permissible.
- `double = obj. GetNormalizedLabelPosition ()` - Set the normalized (0-1) position of the label along the length of the shaft. A value ¿ 1 is permissible.

- `obj.SetConeResolution (int )` - Set/get the resolution of the pieces of the axes actor.
- `int = obj.GetConeResolutionMinValue ()` - Set/get the resolution of the pieces of the axes actor.
- `int = obj.GetConeResolutionMaxValue ()` - Set/get the resolution of the pieces of the axes actor.
- `int = obj.GetConeResolution ()` - Set/get the resolution of the pieces of the axes actor.
- `obj.SetSphereResolution (int )` - Set/get the resolution of the pieces of the axes actor.
- `int = obj.GetSphereResolutionMinValue ()` - Set/get the resolution of the pieces of the axes actor.
- `int = obj.GetSphereResolutionMaxValue ()` - Set/get the resolution of the pieces of the axes actor.
- `int = obj.GetSphereResolution ()` - Set/get the resolution of the pieces of the axes actor.
- `obj.SetCylinderResolution (int )` - Set/get the resolution of the pieces of the axes actor.
- `int = obj.GetCylinderResolutionMinValue ()` - Set/get the resolution of the pieces of the axes actor.
- `int = obj.GetCylinderResolutionMaxValue ()` - Set/get the resolution of the pieces of the axes actor.
- `int = obj.GetCylinderResolution ()` - Set/get the resolution of the pieces of the axes actor.
- `obj.SetConeRadius (double )` - Set/get the radius of the pieces of the axes actor.
- `double = obj.GetConeRadiusMinValue ()` - Set/get the radius of the pieces of the axes actor.
- `double = obj.GetConeRadiusMaxValue ()` - Set/get the radius of the pieces of the axes actor.
- `double = obj.GetConeRadius ()` - Set/get the radius of the pieces of the axes actor.
- `obj.SetSphereRadius (double )` - Set/get the radius of the pieces of the axes actor.
- `double = obj.GetSphereRadiusMinValue ()` - Set/get the radius of the pieces of the axes actor.
- `double = obj.GetSphereRadiusMaxValue ()` - Set/get the radius of the pieces of the axes actor.
- `double = obj.GetSphereRadius ()` - Set/get the radius of the pieces of the axes actor.
- `obj.SetCylinderRadius (double )` - Set/get the radius of the pieces of the axes actor.
- `double = obj.GetCylinderRadiusMinValue ()` - Set/get the radius of the pieces of the axes actor.
- `double = obj.GetCylinderRadiusMaxValue ()` - Set/get the radius of the pieces of the axes actor.
- `double = obj.GetCylinderRadius ()` - Set/get the radius of the pieces of the axes actor.
- `obj.SetShaftType (int type)` - Set the type of the shaft to a cylinder, line, or user defined geometry.
- `obj.SetShaftTypeToCylinder ()` - Set the type of the shaft to a cylinder, line, or user defined geometry.
- `obj.SetShaftTypeToLine ()` - Set the type of the shaft to a cylinder, line, or user defined geometry.
- `obj.SetShaftTypeToUserDefined ()` - Set the type of the shaft to a cylinder, line, or user defined geometry.
- `int = obj.GetShaftType ()` - Set the type of the shaft to a cylinder, line, or user defined geometry.
- `obj.SetTipType (int type)` - Set the type of the tip to a cone, sphere, or user defined geometry.
- `obj.SetTipTypeToCone ()` - Set the type of the tip to a cone, sphere, or user defined geometry.

- `obj.SetTipTypeToSphere ()` - Set the type of the tip to a cone, sphere, or user defined geometry.
- `obj.SetTipTypeToUserDefined ()` - Set the type of the tip to a cone, sphere, or user defined geometry.
- `int = obj.GetTipType ()` - Set the type of the tip to a cone, sphere, or user defined geometry.
- `obj.SetUserDefinedTip (vtkPolyData )` - Set the user defined tip polydata.
- `vtkPolyData = obj.GetUserDefinedTip ()` - Set the user defined tip polydata.
- `obj.SetUserDefinedShaft (vtkPolyData )` - Set the user defined shaft polydata.
- `vtkPolyData = obj.GetUserDefinedShaft ()` - Set the user defined shaft polydata.
- `vtkProperty = obj.GetXAxisTipProperty ()` - Get the tip properties.
- `vtkProperty = obj.GetYAxisTipProperty ()` - Get the tip properties.
- `vtkProperty = obj.GetZAxisTipProperty ()` - Get the tip properties.
- `vtkProperty = obj.GetXAxisShaftProperty ()` - Get the shaft properties.
- `vtkProperty = obj.GetYAxisShaftProperty ()` - Get the shaft properties.
- `vtkProperty = obj.GetZAxisShaftProperty ()` - Get the shaft properties.
- `vtkCaptionActor2D = obj.GetXAxisCaptionActor2D ()` - Retrieve handles to the X, Y and Z axis (so that you can set their text properties for example)
- `vtkCaptionActor2D = obj.GetYAxisCaptionActor2D ()` - Retrieve handles to the X, Y and Z axis (so that you can set their text properties for example)
- `vtkCaptionActor2D = obj.GetZAxisCaptionActor2D ()` - Set/get the label text.
- `obj.SetXAxisLabelText (string )` - Set/get the label text.
- `string = obj.GetXAxisLabelText ()` - Set/get the label text.
- `obj.SetYAxisLabelText (string )` - Set/get the label text.
- `string = obj.GetYAxisLabelText ()` - Set/get the label text.
- `obj.SetZAxisLabelText (string )` - Set/get the label text.
- `string = obj.GetZAxisLabelText ()` - Set/get the label text.
- `obj.SetAxisLabels (int )` - Enable/disable drawing the axis labels.
- `int = obj.GetAxisLabels ()` - Enable/disable drawing the axis labels.
- `obj.AxisLabelsOn ()` - Enable/disable drawing the axis labels.
- `obj.AxisLabelsOff ()` - Enable/disable drawing the axis labels.

34.5 vtkAxisActor

34.5.1 Usage

vtkAxisActor creates an axis with tick marks, labels, and/or a title, depending on the particular instance variable settings. It is assumed that the axes is part of a bounding box and is orthoganal to one of the coordinate axes. To use this class, you typically specify two points defining the start and end points of the line (xyz definition using vtkCoordinate class), the axis type (X, Y or Z), the axis location in relation to the bounding box, the bounding box, the number of labels, and the data range (min,max). You can also control what parts of the axis are visible including the line, the tick marks, the labels, and the title. It is also possible to control gridlines, and specifiy on which 'side' the tickmarks are drawn (again with respect to the underlying assumed bounding box). You can also specify the label format (a printf style format).

This class decides how to locate the labels, and how to create reasonable tick marks and labels.

Labels follow the camera so as to be legible from any viewpoint.

The instance variables Point1 and Point2 are instances of vtkCoordinate. All calculations and references are in World Coordinates.

.SECTION Notes This class was adapted from a 2D version created by Hank Childs called vtkHankAxisActor2D.

To create an instance of class vtkAxisActor, simply invoke its constructor as follows

```
obj = vtkAxisActor
```

34.5.2 Methods

The class vtkAxisActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAxisActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAxisActor = obj.NewInstance ()`
- `vtkAxisActor = obj.SafeDownCast (vtkObject o)`
- `vtkCoordinate = obj.GetPoint1Coordinate ()` - Specify the position of the first point defining the axis.
- `obj.SetPoint1 (double x[3])` - Specify the position of the first point defining the axis.
- `obj.SetPoint1 (double x, double y, double z)` - Specify the position of the first point defining the axis.
- `vtkCoordinate = obj.GetPoint2Coordinate ()` - Specify the position of the second point defining the axis.
- `obj.SetPoint2 (double x[3])` - Specify the position of the second point defining the axis.
- `obj.SetPoint2 (double x, double y, double z)` - Specify the position of the second point defining the axis.
- `obj.SetRange (double , double )` - Specify the (min,max) axis range. This will be used in the generation of labels, if labels are visible.
- `obj.SetRange (double  a[2])` - Specify the (min,max) axis range. This will be used in the generation of labels, if labels are visible.

- `double = obj. GetRange ()` - Specify the (min,max) axis range. This will be used in the generation of labels, if labels are visible.
- `obj.SetBounds (double bounds[6])` - Set or get the bounds for this Actor as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax).
- `double = obj.GetBounds (void )` - Set or get the bounds for this Actor as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax).
- `obj.GetBounds (double bounds[6])` - Set or get the bounds for this Actor as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax).
- `obj.SetLabelFormat (string )` - Set/Get the format with which to print the labels on the axis.
- `string = obj.GetLabelFormat ()` - Set/Get the format with which to print the labels on the axis.
- `obj.SetMinorTicksVisible (int )` - Set/Get the flag that controls whether the minor ticks are visible.
- `int = obj.GetMinorTicksVisible ()` - Set/Get the flag that controls whether the minor ticks are visible.
- `obj.MinorTicksVisibleOn ()` - Set/Get the flag that controls whether the minor ticks are visible.
- `obj.MinorTicksVisibleOff ()` - Set/Get the flag that controls whether the minor ticks are visible.
- `obj.SetTitle (string t)` - Set/Get the title of the axis actor,
- `string = obj.GetTitle ()` - Set/Get the title of the axis actor,
- `obj.SetMajorTickSize (double )` - Set/Get the size of the major tick marks
- `double = obj.GetMajorTickSize ()` - Set/Get the size of the major tick marks
- `obj.SetMinorTickSize (double )` - Set/Get the size of the major tick marks
- `double = obj.GetMinorTickSize ()` - Set/Get the size of the major tick marks
- `obj.SetTickLocation (int )` - Set/Get the location of the ticks.
- `int = obj.GetTickLocationMinValue ()` - Set/Get the location of the ticks.
- `int = obj.GetTickLocationMaxValue ()` - Set/Get the location of the ticks.
- `int = obj.GetTickLocation ()` - Set/Get the location of the ticks.
- `obj.SetTickLocationToInside (void )`
- `obj.SetTickLocationToOutside (void )`
- `obj.SetTickLocationToBoth (void )`
- `obj.SetAxisVisibility (int )` - Set/Get visibility of the axis line.
- `int = obj.GetAxisVisibility ()` - Set/Get visibility of the axis line.
- `obj.AxisVisibilityOn ()` - Set/Get visibility of the axis line.
- `obj.AxisVisibilityOff ()` - Set/Get visibility of the axis line.
- `obj.SetTickVisibility (int )` - Set/Get visibility of the axis tick marks.
- `int = obj.GetTickVisibility ()` - Set/Get visibility of the axis tick marks.
- `obj.TickVisibilityOn ()` - Set/Get visibility of the axis tick marks.
- `obj.TickVisibilityOff ()` - Set/Get visibility of the axis tick marks.

- `obj.SetLabelVisibility (int )` - Set/Get visibility of the axis labels.
- `int = obj.GetLabelVisibility ()` - Set/Get visibility of the axis labels.
- `obj.LabelVisibilityOn ()` - Set/Get visibility of the axis labels.
- `obj.LabelVisibilityOff ()` - Set/Get visibility of the axis labels.
- `obj.SetTitleVisibility (int )` - Set/Get visibility of the axis title.
- `int = obj.GetTitleVisibility ()` - Set/Get visibility of the axis title.
- `obj.TitleVisibilityOn ()` - Set/Get visibility of the axis title.
- `obj.TitleVisibilityOff ()` - Set/Get visibility of the axis title.
- `obj.SetDrawGridlines (int )` - Set/Get whether gridlines should be drawn.
- `int = obj.GetDrawGridlines ()` - Set/Get whether gridlines should be drawn.
- `obj.DrawGridlinesOn ()` - Set/Get whether gridlines should be drawn.
- `obj.DrawGridlinesOff ()` - Set/Get whether gridlines should be drawn.
- `obj.SetGridlineXLength (double )` - Set/Get the length to use when drawing gridlines.
- `double = obj.GetGridlineXLength ()` - Set/Get the length to use when drawing gridlines.
- `obj.SetGridlineYLength (double )` - Set/Get the length to use when drawing gridlines.
- `double = obj.GetGridlineYLength ()` - Set/Get the length to use when drawing gridlines.
- `obj.SetGridlineZLength (double )` - Set/Get the length to use when drawing gridlines.
- `double = obj.GetGridlineZLength ()` - Set/Get the length to use when drawing gridlines.
- `obj.SetAxisType (int )` - Set/Get the type of this axis.
- `int = obj.GetAxisTypeMinValue ()` - Set/Get the type of this axis.
- `int = obj.GetAxisTypeMaxValue ()` - Set/Get the type of this axis.
- `int = obj.GetAxisType ()` - Set/Get the type of this axis.
- `obj.SetAxisTypeToX (void )` - Set/Get the type of this axis.
- `obj.SetAxisTypeToY (void )` - Set/Get the type of this axis.
- `obj.SetAxisTypeToZ (void )` - Set/Get the type of this axis.
- `obj.SetAxisPosition (int )` - Set/Get the position of this axis (in relation to an an assumed bounding box). For an x-type axis, MINMIN corresponds to the x-edge in the bounding box where Y values are minimum and Z values are minimum. For a y-type axis, MAXMIN corresponds to the y-edge where X values are maximum and Z values are minimum.
- `int = obj.GetAxisPositionMinValue ()` - Set/Get the position of this axis (in relation to an an assumed bounding box). For an x-type axis, MINMIN corresponds to the x-edge in the bounding box where Y values are minimum and Z values are minimum. For a y-type axis, MAXMIN corresponds to the y-edge where X values are maximum and Z values are minimum.
- `int = obj.GetAxisPositionMaxValue ()` - Set/Get the position of this axis (in relation to an an assumed bounding box). For an x-type axis, MINMIN corresponds to the x-edge in the bounding box where Y values are minimum and Z values are minimum. For a y-type axis, MAXMIN corresponds to the y-edge where X values are maximum and Z values are minimum.

- `int = obj.GetAxisPosition ()` - Set/Get the position of this axis (in relation to an an assumed bounding box). For an x-type axis, MINMIN corresponds to the x-edge in the bounding box where Y values are minimum and Z values are minimum. For a y-type axis, MAXMIN corresponds to the y-edge where X values are maximum and Z values are minimum.
- `obj.SetAxisPositionToMinMin (void )`
- `obj.SetAxisPositionToMinMax (void )`
- `obj.SetAxisPositionToMaxMax (void )`
- `obj.SetAxisPositionToMaxMin (void )`
- `obj.SetCamera (vtkCamera )` - Set/Get the camera for this axis. The camera is used by the labels to 'follow' the camera and be legible from any viewpoint.
- `vtkCamera = obj.GetCamera ()` - Set/Get the camera for this axis. The camera is used by the labels to 'follow' the camera and be legible from any viewpoint.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Draw the axis.
- `int = obj.RenderTranslucentGeometry (vtkViewport )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of an axis actor. Overloads the virtual vtkProp method.
- `obj.SetLabelScale (double )`
- `obj.SetTitleScale (double )`
- `obj.SetMinorStart (double )` - Set/Get the starting position for minor and major tick points, and the delta values that determine their spacing.
- `double = obj.GetMinorStart ()` - Set/Get the starting position for minor and major tick points, and the delta values that determine their spacing.
- `obj.SetMajorStart (double )` - Set/Get the starting position for minor and major tick points, and the delta values that determine their spacing.
- `double = obj.GetMajorStart ()` - Set/Get the starting position for minor and major tick points, and the delta values that determine their spacing.
- `obj.SetDeltaMinor (double )` - Set/Get the starting position for minor and major tick points, and the delta values that determine their spacing.
- `double = obj.GetDeltaMinor ()` - Set/Get the starting position for minor and major tick points, and the delta values that determine their spacing.
- `obj.SetDeltaMajor (double )` - Set/Get the starting position for minor and major tick points, and the delta values that determine their spacing.
- `double = obj.GetDeltaMajor ()` - Set/Get the starting position for minor and major tick points, and the delta values that determine their spacing.
- `obj.BuildAxis (vtkViewport viewport, bool )`

34.6 vtkBarChartActor

34.6.1 Usage

vtkBarChartActor generates a bar chart from an array of numbers defined in field data (a vtkDataObject). To use this class, you must specify an input data object. You'll probably also want to specify the position of the plot be setting the Position and Position2 instance variables, which define a rectangle in which the plot lies. There are also many other instance variables that control the look of the plot includes its title and legend.

Set the text property/attributes of the title and the labels through the vtkTextProperty objects associated with these components.

To create an instance of class vtkBarChartActor, simply invoke its constructor as follows

```
obj = vtkBarChartActor
```

34.6.2 Methods

The class vtkBarChartActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBarChartActor class.

- `string = obj.GetClassName ()` - Standard methods for type information and printing.
- `int = obj.IsA (string name)` - Standard methods for type information and printing.
- `vtkBarChartActor = obj.NewInstance ()` - Standard methods for type information and printing.
- `vtkBarChartActor = obj.SafeDownCast (vtkObject o)` - Standard methods for type information and printing.
- `obj.SetInput (vtkDataObject )` - Set the input to the bar chart actor.
- `vtkDataObject = obj.GetInput ()` - Get the input data object to this actor.
- `obj.SetTitleVisibility (int )` - Enable/Disable the display of a plot title.
- `int = obj.GetTitleVisibility ()` - Enable/Disable the display of a plot title.
- `obj.TitleVisibilityOn ()` - Enable/Disable the display of a plot title.
- `obj.TitleVisibilityOff ()` - Enable/Disable the display of a plot title.
- `obj.SetTitle (string )` - Set/Get the title of the bar chart.
- `string = obj.GetTitle ()` - Set/Get the title of the bar chart.
- `obj.SetTitleTextProperty (vtkTextProperty p)` - Set/Get the title text property. The property controls the appearance of the plot title.
- `vtkTextProperty = obj.GetTitleTextProperty ()` - Set/Get the title text property. The property controls the appearance of the plot title.
- `obj.SetLabelVisibility (int )` - Enable/Disable the display of bar labels.
- `int = obj.GetLabelVisibility ()` - Enable/Disable the display of bar labels.
- `obj.LabelVisibilityOn ()` - Enable/Disable the display of bar labels.
- `obj.LabelVisibilityOff ()` - Enable/Disable the display of bar labels.

- `obj.SetLabelTextProperty (vtkTextProperty p)` - Set/Get the labels text property. This controls the appearance of all bar bar labels.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - Set/Get the labels text property. This controls the appearance of all bar bar labels.
- `obj.SetBarColor (int i, double r, double g, double b)` - Specify colors for each bar. If not specified, they are automatically generated.
- `obj.SetBarColor (int i, double color[3])` - Specify colors for each bar. If not specified, they are automatically generated.
- `obj.SetBarLabel (int i, string )` - Specify the names of each bar. If not specified, then an integer number is automatically generated.
- `string = obj.GetBarLabel (int i)` - Specify the names of each bar. If not specified, then an integer number is automatically generated.
- `obj.SetYTitle (string )` - Specify the title of the y-axis.
- `string = obj.GetYTitle ()` - Specify the title of the y-axis.
- `obj.SetLegendVisibility (int )` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `int = obj.GetLegendVisibility ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.LegendVisibilityOn ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.LegendVisibilityOff ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `vtkLegendBoxActor = obj.GetLegendActor ()` - Retrieve handles to the legend box. This is useful if you would like to manually control the legend appearance.
- `int = obj.RenderOverlay (vtkViewport )` - Draw the bar plot.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Draw the bar plot.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Does this prop have some translucent polygonal geometry?
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.

34.7 vtkCaptionActor2D

34.7.1 Usage

vtkCaptionActor2D is a hybrid 2D/3D actor that is used to associate text with a point (the AttachmentPoint) in the scene. The caption can be drawn with a rectangular border and a leader connecting the caption to the attachment point. Optionally, the leader can be glyphed at its endpoint to create arrow heads or other indicators.

To use the caption actor, you normally specify the Position and Position2 coordinates (these are inherited from the vtkActor2D superclass). (Note that Position2 can be set using vtkActor2D's SetWidth() and SetHeight() methods.) Position and Position2 define the size of the caption, and a third point, the AttachmentPoint, defines a point that the caption is associated with. You must also define the caption text, whether you want a border around the caption, and whether you want a leader from the caption to the attachment point. The font attributes of the text can be set through the vtkTextProperty associated to this actor. You also indicate whether you want the leader to be 2D or 3D. (2D leaders are always drawn over the underlying geometry. 3D leaders may be occluded by the geometry.) The leader may also be terminated by an optional glyph (e.g., arrow).

The trickiest part about using this class is setting Position, Position2, and AttachmentPoint correctly. These instance variables are vtkCoordinates, and can be set up in various ways. In default usage, the AttachmentPoint is defined in the world coordinate system, Position is the lower-left corner of the caption and relative to AttachmentPoint (defined in display coordaintes, i.e., pixels), and Position2 is relative to Position and is the upper-right corner (also in display coordinates). However, the user has full control over the coordinates, and can do things like place the caption in a fixed position in the renderer, with the leader moving with the AttachmentPoint.

To create an instance of class vtkCaptionActor2D, simply invoke its constructor as follows

```
obj = vtkCaptionActor2D
```

34.7.2 Methods

The class vtkCaptionActor2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCaptionActor2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCaptionActor2D = obj.NewInstance ()`
- `vtkCaptionActor2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetCaption (string )` - Define the text to be placed in the caption. The text can be multiple lines (separated by ”
n”).
- `string = obj.GetCaption ()` - Define the text to be placed in the caption. The text can be multiple lines (separated by ”
n”).
- `vtkCoordinate = obj.GetAttachmentPointCoordinate ()` - Set/Get the attachment point for the caption. By default, the attachment point is defined in world coordinates, but this can be changed using vtkCoordinate methods.
- `obj.SetAttachmentPoint (double, double, double)` - Set/Get the attachment point for the caption. By default, the attachment point is defined in world coordinates, but this can be changed using vtkCoordinate methods.
- `obj.SetAttachmentPoint (double a[3])` - Set/Get the attachment point for the caption. By default, the attachment point is defined in world coordinates, but this can be changed using vtkCoordinate methods.
- `double = obj.GetAttachmentPoint ()` - Set/Get the attachment point for the caption. By default, the attachment point is defined in world coordinates, but this can be changed using vtkCoordinate methods.

- `obj.SetBorder (int )` - Enable/disable the placement of a border around the text.
- `int = obj.GetBorder ()` - Enable/disable the placement of a border around the text.
- `obj.BorderOn ()` - Enable/disable the placement of a border around the text.
- `obj.BorderOff ()` - Enable/disable the placement of a border around the text.
- `obj.SetLeader (int )` - Enable/disable drawing a "line" from the caption to the attachment point.
- `int = obj.GetLeader ()` - Enable/disable drawing a "line" from the caption to the attachment point.
- `obj.LeaderOn ()` - Enable/disable drawing a "line" from the caption to the attachment point.
- `obj.LeaderOff ()` - Enable/disable drawing a "line" from the caption to the attachment point.
- `obj.SetThreeDimensionalLeader (int )` - Indicate whether the leader is 2D (no hidden line) or 3D (z-buffered).
- `int = obj.GetThreeDimensionalLeader ()` - Indicate whether the leader is 2D (no hidden line) or 3D (z-buffered).
- `obj.ThreeDimensionalLeaderOn ()` - Indicate whether the leader is 2D (no hidden line) or 3D (z-buffered).
- `obj.ThreeDimensionalLeaderOff ()` - Indicate whether the leader is 2D (no hidden line) or 3D (z-buffered).
- `obj.SetLeaderGlyph (vtkPolyData )` - Specify a glyph to be used as the leader "head". This could be something like an arrow or sphere. If not specified, no glyph is drawn. Note that the glyph is assumed to be aligned along the x-axis and is rotated about the origin.
- `vtkPolyData = obj.GetLeaderGlyph ()` - Specify a glyph to be used as the leader "head". This could be something like an arrow or sphere. If not specified, no glyph is drawn. Note that the glyph is assumed to be aligned along the x-axis and is rotated about the origin.
- `obj.SetLeaderGlyphSize (double )` - Specify the relative size of the leader head. This is expressed as a fraction of the size (diagonal length) of the renderer. The leader head is automatically scaled so that window resize, zooming or other camera motion results in proportional changes in size to the leader glyph.
- `double = obj.GetLeaderGlyphSizeMinValue ()` - Specify the relative size of the leader head. This is expressed as a fraction of the size (diagonal length) of the renderer. The leader head is automatically scaled so that window resize, zooming or other camera motion results in proportional changes in size to the leader glyph.
- `double = obj.GetLeaderGlyphSizeMaxValue ()` - Specify the relative size of the leader head. This is expressed as a fraction of the size (diagonal length) of the renderer. The leader head is automatically scaled so that window resize, zooming or other camera motion results in proportional changes in size to the leader glyph.
- `double = obj.GetLeaderGlyphSize ()` - Specify the relative size of the leader head. This is expressed as a fraction of the size (diagonal length) of the renderer. The leader head is automatically scaled so that window resize, zooming or other camera motion results in proportional changes in size to the leader glyph.
- `obj.SetMaximumLeaderGlyphSize (int )` - Specify the maximum size of the leader head (if any) in pixels. This is used in conjunction with LeaderGlyphSize to cap the maximum size of the LeaderGlyph.

- `int = obj.GetMaximumLeaderGlyphSizeMinValue ()` - Specify the maximum size of the leader head (if any) in pixels. This is used in conjunction with LeaderGlyphSize to cap the maximum size of the LeaderGlyph.

- `int = obj.GetMaximumLeaderGlyphSizeMaxValue ()` - Specify the maximum size of the leader head (if any) in pixels. This is used in conjunction with LeaderGlyphSize to cap the maximum size of the LeaderGlyph.

- `int = obj.GetMaximumLeaderGlyphSize ()` - Specify the maximum size of the leader head (if any) in pixels. This is used in conjunction with LeaderGlyphSize to cap the maximum size of the LeaderGlyph.

- `obj.SetPadding (int )` - Set/Get the padding between the caption and the border. The value is specified in pixels.

- `int = obj.GetPaddingMinValue ()` - Set/Get the padding between the caption and the border. The value is specified in pixels.

- `int = obj.GetPaddingMaxValue ()` - Set/Get the padding between the caption and the border. The value is specified in pixels.

- `int = obj.GetPadding ()` - Set/Get the padding between the caption and the border. The value is specified in pixels.

- `vtkTextActor = obj.GetTextActor ()` - Get the text actor used by the caption. This is useful if you want to control justification and other characteristics of the text actor.

- `obj.SetCaptionTextProperty (vtkTextProperty p)` - Set/Get the text property.

- `vtkTextProperty = obj.GetCaptionTextProperty ()` - Set/Get the text property.

- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this scaled text actor. Overloads the virtual vtkProp method.

- `obj.SetAttachEdgeOnly (int )` - Enable/disable whether to attach the arrow only to the edge, NOT the vertices of the caption border.

- `int = obj.GetAttachEdgeOnly ()` - Enable/disable whether to attach the arrow only to the edge, NOT the vertices of the caption border.

- `obj.AttachEdgeOnlyOn ()` - Enable/disable whether to attach the arrow only to the edge, NOT the vertices of the caption border.

- `obj.AttachEdgeOnlyOff ()` - Enable/disable whether to attach the arrow only to the edge, NOT the vertices of the caption border.

34.8 vtkCornerAnnotation

34.8.1 Usage

This is an annotation object that manages four text actors / mappers to provide annotation in the four corners of a viewport

To create an instance of class vtkCornerAnnotation, simply invoke its constructor as follows

```
obj = vtkCornerAnnotation
```

34.8.2 Methods

The class vtkCornerAnnotation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCornerAnnotation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCornerAnnotation = obj.NewInstance ()`
- `vtkCornerAnnotation = obj.SafeDownCast (vtkObject o)`
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Draw the scalar bar and annotation text to the screen.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Draw the scalar bar and annotation text to the screen.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Draw the scalar bar and annotation text to the screen.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.SetMaximumLineHeight (double )` - Set/Get the maximum height of a line of text as a percentage of the vertical area allocated to this scaled text actor. Defaults to 1.0
- `double = obj.GetMaximumLineHeight ()` - Set/Get the maximum height of a line of text as a percentage of the vertical area allocated to this scaled text actor. Defaults to 1.0
- `obj.SetMinimumFontSize (int )` - Set/Get the minimum/maximum size font that will be shown. If the font drops below the minimum size it will not be rendered.
- `int = obj.GetMinimumFontSize ()` - Set/Get the minimum/maximum size font that will be shown. If the font drops below the minimum size it will not be rendered.
- `obj.SetMaximumFontSize (int )` - Set/Get the minimum/maximum size font that will be shown. If the font drops below the minimum size it will not be rendered.
- `int = obj.GetMaximumFontSize ()` - Set/Get the minimum/maximum size font that will be shown. If the font drops below the minimum size it will not be rendered.
- `obj.SetLinearFontScaleFactor (double )` - Set/Get font scaling factors The font size, f, is calculated as the largest possible value such that the annotations for the given viewport do not overlap. This font size is scaled non-linearly with the viewport size, to maintain an acceptable readable size at larger viewport sizes, without being too big. f' = linearScale * pow(f,nonlinearScale)
- `double = obj.GetLinearFontScaleFactor ()` - Set/Get font scaling factors The font size, f, is calculated as the largest possible value such that the annotations for the given viewport do not overlap. This font size is scaled non-linearly with the viewport size, to maintain an acceptable readable size at larger viewport sizes, without being too big. f' = linearScale * pow(f,nonlinearScale)
- `obj.SetNonlinearFontScaleFactor (double )` - Set/Get font scaling factors The font size, f, is calculated as the largest possible value such that the annotations for the given viewport do not overlap. This font size is scaled non-linearly with the viewport size, to maintain an acceptable readable size at larger viewport sizes, without being too big. f' = linearScale * pow(f,nonlinearScale)

- `double = obj.GetNonlinearFontScaleFactor ()` - Set/Get font scaling factors The font size, f, is calculated as the largest possible value such that the annotations for the given viewport do not overlap. This font size is scaled non-linearly with the viewport size, to maintain an acceptable readable size at larger viewport sizes, without being too big. f' = linearScale * pow(f,nonlinearScale)
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.SetText (int i, string text)` - Set/Get the text to be displayed for each corner
- `string = obj.GetText (int i)` - Set/Get the text to be displayed for each corner
- `obj.ClearAllTexts ()` - Set/Get the text to be displayed for each corner
- `obj.CopyAllTextsFrom (vtkCornerAnnotation ca)` - Set/Get the text to be displayed for each corner
- `obj.SetImageActor (vtkImageActor )` - Set an image actor to look at for slice information
- `vtkImageActor = obj.GetImageActor ()` - Set an image actor to look at for slice information
- `obj.SetWindowLevel (vtkImageMapToWindowLevelColors )` - Set an instance of vtkImageMapToWindowLevelColors to use for looking at window level changes
- `vtkImageMapToWindowLevelColors = obj.GetWindowLevel ()` - Set an instance of vtkImageMapToWindowLevelColors to use for looking at window level changes
- `obj.SetLevelShift (double )` - Set the value to shift the level by.
- `double = obj.GetLevelShift ()` - Set the value to shift the level by.
- `obj.SetLevelScale (double )` - Set the value to scale the level by.
- `double = obj.GetLevelScale ()` - Set the value to scale the level by.
- `obj.SetTextProperty (vtkTextProperty p)` - Set/Get the text property of all corners.
- `vtkTextProperty = obj.GetTextProperty ()` - Set/Get the text property of all corners.
- `obj.ShowSliceAndImageOn ()` - Even if there is an image actor, should 'slice' and 'image' be displayed?
- `obj.ShowSliceAndImageOff ()` - Even if there is an image actor, should 'slice' and 'image' be displayed?
- `obj.SetShowSliceAndImage (int )` - Even if there is an image actor, should 'slice' and 'image' be displayed?
- `int = obj.GetShowSliceAndImage ()` - Even if there is an image actor, should 'slice' and 'image' be displayed?

34.9 vtkCubeAxesActor

34.9.1 Usage

vtkCubeAxesActor is a composite actor that draws axes of the bounding box of an input dataset. The axes include labels and titles for the x-y-z axes. The algorithm selects which axes to draw based on the user-defined 'fly' mode. (STATIC is default). 'STATIC' constructs axes from all edges of the bounding box. 'CLOSEST_TRIAD' consists of the three axes x-y-z forming a triad that lies closest to the specified camera. 'FURTHEST_TRIAD' consists of the three axes x-y-z forming a triad that lies furthest from the

specified camera. 'OUTER_EDGES' is constructed from edges that are on the "exterior" of the bounding box, exterior as determined from examining outer edges of the bounding box in projection (display) space.

To use this object you must define a bounding box and the camera used to render the vtkCubeAxesActor. You can optionally turn on/off labels, ticks, gridlines, and set tick location, number of labels, and text to use for axis-titles. A 'corner offset' can also be set. This allows the axes to be set partially away from the actual bounding box to perhaps prevent overlap of labels between the various axes.

The Bounds instance variable (an array of six doubles) is used to determine the bounding box.

To create an instance of class vtkCubeAxesActor, simply invoke its constructor as follows

```
obj = vtkCubeAxesActor
```

34.9.2 Methods

The class vtkCubeAxesActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCubeAxesActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCubeAxesActor = obj.NewInstance ()`
- `vtkCubeAxesActor = obj.SafeDownCast (vtkObject o)`
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Draw the axes as per the vtkProp superclass' API.
- `int = obj.RenderTranslucentGeometry (vtkViewport )` - Explicitly specify the region in space around which to draw the bounds. The bounds is used only when no Input or Prop is specified. The bounds are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.SetBounds (double , double , double , double , double , double )` - Explicitly specify the region in space around which to draw the bounds. The bounds is used only when no Input or Prop is specified. The bounds are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.SetBounds (double a[6])` - Explicitly specify the region in space around which to draw the bounds. The bounds is used only when no Input or Prop is specified. The bounds are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `double = obj.GetBounds ()` - Explicitly specify the region in space around which to draw the bounds. The bounds is used only when no Input or Prop is specified. The bounds are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.GetBounds (double bounds[6])` - Explicitly specify the region in space around which to draw the bounds. The bounds is used only when no Input or Prop is specified. The bounds are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.SetCamera (vtkCamera )` - Set/Get the camera to perform scaling and translation of the vtkCubeAxesActor.
- `vtkCamera = obj.GetCamera ()` - Set/Get the camera to perform scaling and translation of the vtkCubeAxesActor.
- `obj.SetFlyMode (int )` - Specify a mode to control how the axes are drawn: either static, closest triad, furthest triad or outer edges in relation to the camera position.

- `int = obj.GetFlyModeMinValue ()` - Specify a mode to control how the axes are drawn: either static, closest triad, furthest triad or outer edges in relation to the camera position.
- `int = obj.GetFlyModeMaxValue ()` - Specify a mode to control how the axes are drawn: either static, closest triad, furthest triad or outer edges in relation to the camera position.
- `int = obj.GetFlyMode ()` - Specify a mode to control how the axes are drawn: either static, closest triad, furthest triad or outer edges in relation to the camera position.
- `obj.SetFlyModeToOuterEdges ()` - Specify a mode to control how the axes are drawn: either static, closest triad, furthest triad or outer edges in relation to the camera position.
- `obj.SetFlyModeToClosestTriad ()` - Specify a mode to control how the axes are drawn: either static, closest triad, furthest triad or outer edges in relation to the camera position.
- `obj.SetFlyModeToFurthestTriad ()` - Specify a mode to control how the axes are drawn: either static, closest triad, furthest triad or outer edges in relation to the camera position.
- `obj.SetFlyModeToStaticTriad ()` - Specify a mode to control how the axes are drawn: either static, closest triad, furthest triad or outer edges in relation to the camera position.
- `obj.SetFlyModeToStaticEdges ()` - Specify a mode to control how the axes are drawn: either static, closest triad, furthest triad or outer edges in relation to the camera position.
- `obj.SetXTitle (string )` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `string = obj.GetXTitle ()` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `obj.SetXUnits (string )` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `string = obj.GetXUnits ()` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `obj.SetYTitle (string )` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `string = obj.GetYTitle ()` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `obj.SetYUnits (string )` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `string = obj.GetYUnits ()` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `obj.SetZTitle (string )` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `string = obj.GetZTitle ()` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `obj.SetZUnits (string )` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `string = obj.GetZUnits ()` - Set/Get the labels for the x, y, and z axes. By default, use "X-Axis", "Y-Axis" and "Z-Axis".
- `obj.SetXLabelFormat (string )` - Set/Get the format with which to print the labels on each of the x-y-z axes.

- `string = obj.GetXLabelFormat ()` - Set/Get the format with which to print the labels on each of the x-y-z axes.
- `obj.SetYLabelFormat (string )` - Set/Get the format with which to print the labels on each of the x-y-z axes.
- `string = obj.GetYLabelFormat ()` - Set/Get the format with which to print the labels on each of the x-y-z axes.
- `obj.SetZLabelFormat (string )` - Set/Get the format with which to print the labels on each of the x-y-z axes.
- `string = obj.GetZLabelFormat ()` - Set/Get the format with which to print the labels on each of the x-y-z axes.
- `obj.SetInertia (int )` - Set/Get the inertial factor that controls how often (i.e, how many renders) the axes can switch position (jump from one axes to another).
- `int = obj.GetInertiaMinValue ()` - Set/Get the inertial factor that controls how often (i.e, how many renders) the axes can switch position (jump from one axes to another).
- `int = obj.GetInertiaMaxValue ()` - Set/Get the inertial factor that controls how often (i.e, how many renders) the axes can switch position (jump from one axes to another).
- `int = obj.GetInertia ()` - Set/Get the inertial factor that controls how often (i.e, how many renders) the axes can switch position (jump from one axes to another).
- `obj.SetCornerOffset (double )` - Specify an offset value to "pull back" the axes from the corner at which they are joined to avoid overlap of axes labels. The "CornerOffset" is the fraction of the axis length to pull back.
- `double = obj.GetCornerOffset ()` - Specify an offset value to "pull back" the axes from the corner at which they are joined to avoid overlap of axes labels. The "CornerOffset" is the fraction of the axis length to pull back.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.SetXAxisVisibility (int )` - Turn on and off the visibility of each axis.
- `int = obj.GetXAxisVisibility ()` - Turn on and off the visibility of each axis.
- `obj.XAxisVisibilityOn ()` - Turn on and off the visibility of each axis.
- `obj.XAxisVisibilityOff ()` - Turn on and off the visibility of each axis.
- `obj.SetYAxisVisibility (int )` - Turn on and off the visibility of each axis.
- `int = obj.GetYAxisVisibility ()` - Turn on and off the visibility of each axis.
- `obj.YAxisVisibilityOn ()` - Turn on and off the visibility of each axis.
- `obj.YAxisVisibilityOff ()` - Turn on and off the visibility of each axis.
- `obj.SetZAxisVisibility (int )` - Turn on and off the visibility of each axis.
- `int = obj.GetZAxisVisibility ()` - Turn on and off the visibility of each axis.
- `obj.ZAxisVisibilityOn ()` - Turn on and off the visibility of each axis.
- `obj.ZAxisVisibilityOff ()` - Turn on and off the visibility of each axis.

- `obj.SetXAxisLabelVisibility (int )` - Turn on and off the visibility of labels for each axis.
- `int = obj.GetXAxisLabelVisibility ()` - Turn on and off the visibility of labels for each axis.
- `obj.XAxisLabelVisibilityOn ()` - Turn on and off the visibility of labels for each axis.
- `obj.XAxisLabelVisibilityOff ()` - Turn on and off the visibility of labels for each axis.
- `obj.SetYAxisLabelVisibility (int )`
- `int = obj.GetYAxisLabelVisibility ()`
- `obj.YAxisLabelVisibilityOn ()`
- `obj.YAxisLabelVisibilityOff ()`
- `obj.SetZAxisLabelVisibility (int )`
- `int = obj.GetZAxisLabelVisibility ()`
- `obj.ZAxisLabelVisibilityOn ()`
- `obj.ZAxisLabelVisibilityOff ()`
- `obj.SetXAxisTickVisibility (int )` - Turn on and off the visibility of ticks for each axis.
- `int = obj.GetXAxisTickVisibility ()` - Turn on and off the visibility of ticks for each axis.
- `obj.XAxisTickVisibilityOn ()` - Turn on and off the visibility of ticks for each axis.
- `obj.XAxisTickVisibilityOff ()` - Turn on and off the visibility of ticks for each axis.
- `obj.SetYAxisTickVisibility (int )`
- `int = obj.GetYAxisTickVisibility ()`
- `obj.YAxisTickVisibilityOn ()`
- `obj.YAxisTickVisibilityOff ()`
- `obj.SetZAxisTickVisibility (int )`
- `int = obj.GetZAxisTickVisibility ()`
- `obj.ZAxisTickVisibilityOn ()`
- `obj.ZAxisTickVisibilityOff ()`
- `obj.SetXAxisMinorTickVisibility (int )` - Turn on and off the visibility of minor ticks for each axis.
- `int = obj.GetXAxisMinorTickVisibility ()` - Turn on and off the visibility of minor ticks for each axis.
- `obj.XAxisMinorTickVisibilityOn ()` - Turn on and off the visibility of minor ticks for each axis.
- `obj.XAxisMinorTickVisibilityOff ()` - Turn on and off the visibility of minor ticks for each axis.
- `obj.SetYAxisMinorTickVisibility (int )`
- `int = obj.GetYAxisMinorTickVisibility ()`
- `obj.YAxisMinorTickVisibilityOn ()`
- `obj.YAxisMinorTickVisibilityOff ()`

- `obj.SetZAxisMinorTickVisibility (int )`
- `int = obj.GetZAxisMinorTickVisibility ()`
- `obj.ZAxisMinorTickVisibilityOn ()`
- `obj.ZAxisMinorTickVisibilityOff ()`
- `obj.SetDrawXGridlines (int )`
- `int = obj.GetDrawXGridlines ()`
- `obj.DrawXGridlinesOn ()`
- `obj.DrawXGridlinesOff ()`
- `obj.SetDrawYGridlines (int )`
- `int = obj.GetDrawYGridlines ()`
- `obj.DrawYGridlinesOn ()`
- `obj.DrawYGridlinesOff ()`
- `obj.SetDrawZGridlines (int )`
- `int = obj.GetDrawZGridlines ()`
- `obj.DrawZGridlinesOn ()`
- `obj.DrawZGridlinesOff ()`
- `obj.SetTickLocation (int )` - Set/Get the location of ticks marks.
- `int = obj.GetTickLocationMinValue ()` - Set/Get the location of ticks marks.
- `int = obj.GetTickLocationMaxValue ()` - Set/Get the location of ticks marks.
- `int = obj.GetTickLocation ()` - Set/Get the location of ticks marks.
- `obj.SetTickLocationToInside (void )`
- `obj.SetTickLocationToOutside (void )`
- `obj.SetTickLocationToBoth (void )`
- `obj.SetLabelScaling (bool , int , int , int )`
- `obj.ShallowCopy (vtkCubeAxesActor actor)` - Shallow copy of a KatCubeAxesActor.

34.10 vtkCubeAxesActor2D

34.10.1 Usage

vtkCubeAxesActor2D is a composite actor that draws three axes of the bounding box of an input dataset. The axes include labels and titles for the x-y-z axes. The algorithm selects the axes that are on the "exterior" of the bounding box, exterior as determined from examining outer edges of the bounding box in projection (display) space. Alternatively, the edges closest to the viewer (i.e., camera position) can be drawn.

To use this object you must define a bounding box and the camera used to render the vtkCubeAxesActor2D. The camera is used to control the scaling and position of the vtkCubeAxesActor2D so that it fits in the viewport and always remains visible.)

The font property of the axes titles and labels can be modified through the AxisTitleTextProperty and AxisLabelTextProperty attributes. You may also use the GetXAxisActor2D, GetYAxisActor2D or GetZAxisActor2D methods to access each individual axis actor to modify their font properties.

The bounding box to use is defined in one of three ways. First, if the Input ivar is defined, then the input dataset's bounds is used. If the Input is not defined, and the Prop (superclass of all actors) is defined, then the Prop's bounds is used. If neither the Input or Prop is defined, then the Bounds instance variable (an array of six doubles) is used.

To create an instance of class vtkCubeAxesActor2D, simply invoke its constructor as follows

```
obj = vtkCubeAxesActor2D
```

34.10.2 Methods

The class vtkCubeAxesActor2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCubeAxesActor2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCubeAxesActor2D = obj.NewInstance ()`
- `vtkCubeAxesActor2D = obj.SafeDownCast (vtkObject o)`
- `int = obj.RenderOverlay (vtkViewport )` - Draw the axes as per the vtkProp superclass' API.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Draw the axes as per the vtkProp superclass' API.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Does this prop have some translucent polygonal geometry?
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.SetInput (vtkDataSet )` - Use the bounding box of this input dataset to draw the cube axes. If this is not specified, then the class will attempt to determine the bounds from the defined Prop or Bounds.
- `vtkDataSet = obj.GetInput ()` - Use the bounding box of this input dataset to draw the cube axes. If this is not specified, then the class will attempt to determine the bounds from the defined Prop or Bounds.
- `obj.SetViewProp (vtkProp prop)` - Use the bounding box of this prop to draw the cube axes. The ViewProp is used to determine the bounds only if the Input is not defined.
- `vtkProp = obj.GetViewProp ()` - Use the bounding box of this prop to draw the cube axes. The ViewProp is used to determine the bounds only if the Input is not defined.
- `obj.SetBounds (double , double , double , double , double , double )` - Explicitly specify the region in space around which to draw the bounds. The bounds is used only when no Input or Prop is specified. The bounds are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.SetBounds (double a[6])` - Explicitly specify the region in space around which to draw the bounds. The bounds is used only when no Input or Prop is specified. The bounds are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.

- `double = obj.GetBounds ()` - Explicitly specify the region in space around which to draw the bounds. The bounds is used only when no Input or Prop is specified. The bounds are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.GetBounds (double bounds[6])` - Explicitly specify the region in space around which to draw the bounds. The bounds is used only when no Input or Prop is specified. The bounds are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.SetRanges (double , double , double , double , double , double )` - Explicitly specify the range of values used on the bounds. The ranges are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.SetRanges (double a[6])` - Explicitly specify the range of values used on the bounds. The ranges are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `double = obj.GetRanges ()` - Explicitly specify the range of values used on the bounds. The ranges are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.GetRanges (double ranges[6])` - Explicitly specify the range of values used on the bounds. The ranges are specified according to (xmin,xmax, ymin,ymax, zmin,zmax), making sure that the min's are less than the max's.
- `obj.SetXOrigin (double )` - Explicitly specify an origin for the axes. These usually intersect at one of the corners of the bounding box, however users have the option to override this if necessary
- `obj.SetYOrigin (double )` - Explicitly specify an origin for the axes. These usually intersect at one of the corners of the bounding box, however users have the option to override this if necessary
- `obj.SetZOrigin (double )` - Explicitly specify an origin for the axes. These usually intersect at one of the corners of the bounding box, however users have the option to override this if necessary
- `obj.SetUseRanges (int )` - Set/Get a flag that controls whether the axes use the data ranges or the ranges set by SetRanges. By default the axes use the data ranges.
- `int = obj.GetUseRanges ()` - Set/Get a flag that controls whether the axes use the data ranges or the ranges set by SetRanges. By default the axes use the data ranges.
- `obj.UseRangesOn ()` - Set/Get a flag that controls whether the axes use the data ranges or the ranges set by SetRanges. By default the axes use the data ranges.
- `obj.UseRangesOff ()` - Set/Get a flag that controls whether the axes use the data ranges or the ranges set by SetRanges. By default the axes use the data ranges.
- `obj.SetCamera (vtkCamera )` - Set/Get the camera to perform scaling and translation of the vtkCubeAxesActor2D.
- `vtkCamera = obj.GetCamera ()` - Set/Get the camera to perform scaling and translation of the vtkCubeAxesActor2D.
- `obj.SetFlyMode (int )` - Specify a mode to control how the axes are drawn: either outer edges or closest triad to the camera position, or you may also disable flying of the axes.
- `int = obj.GetFlyModeMinValue ()` - Specify a mode to control how the axes are drawn: either outer edges or closest triad to the camera position, or you may also disable flying of the axes.
- `int = obj.GetFlyModeMaxValue ()` - Specify a mode to control how the axes are drawn: either outer edges or closest triad to the camera position, or you may also disable flying of the axes.

- `int = obj.GetFlyMode ()` - Specify a mode to control how the axes are drawn: either outer edges or closest triad to the camera position, or you may also disable flying of the axes.
- `obj.SetFlyModeToOuterEdges ()` - Specify a mode to control how the axes are drawn: either outer edges or closest triad to the camera position, or you may also disable flying of the axes.
- `obj.SetFlyModeToClosestTriad ()` - Specify a mode to control how the axes are drawn: either outer edges or closest triad to the camera position, or you may also disable flying of the axes.
- `obj.SetFlyModeToNone ()` - Specify a mode to control how the axes are drawn: either outer edges or closest triad to the camera position, or you may also disable flying of the axes.
- `obj.SetScaling (int )` - Set/Get a flag that controls whether the axes are scaled to fit in the viewport. If off, the axes size remains constant (i.e., stay the size of the bounding box). By default scaling is on so the axes are scaled to fit inside the viewport.
- `int = obj.GetScaling ()` - Set/Get a flag that controls whether the axes are scaled to fit in the viewport. If off, the axes size remains constant (i.e., stay the size of the bounding box). By default scaling is on so the axes are scaled to fit inside the viewport.
- `obj.ScalingOn ()` - Set/Get a flag that controls whether the axes are scaled to fit in the viewport. If off, the axes size remains constant (i.e., stay the size of the bounding box). By default scaling is on so the axes are scaled to fit inside the viewport.
- `obj.ScalingOff ()` - Set/Get a flag that controls whether the axes are scaled to fit in the viewport. If off, the axes size remains constant (i.e., stay the size of the bounding box). By default scaling is on so the axes are scaled to fit inside the viewport.
- `obj.SetNumberOfLabels (int )` - Set/Get the number of annotation labels to show along the x, y, and z axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data.
- `int = obj.GetNumberOfLabelsMinValue ()` - Set/Get the number of annotation labels to show along the x, y, and z axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data.
- `int = obj.GetNumberOfLabelsMaxValue ()` - Set/Get the number of annotation labels to show along the x, y, and z axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data.
- `int = obj.GetNumberOfLabels ()` - Set/Get the number of annotation labels to show along the x, y, and z axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data.
- `obj.SetXLabel (string )` - Set/Get the labels for the x, y, and z axes. By default, use "X", "Y" and "Z".
- `string = obj.GetXLabel ()` - Set/Get the labels for the x, y, and z axes. By default, use "X", "Y" and "Z".
- `obj.SetYLabel (string )` - Set/Get the labels for the x, y, and z axes. By default, use "X", "Y" and "Z".
- `string = obj.GetYLabel ()` - Set/Get the labels for the x, y, and z axes. By default, use "X", "Y" and "Z".
- `obj.SetZLabel (string )` - Set/Get the labels for the x, y, and z axes. By default, use "X", "Y" and "Z".

- `string = obj.GetZLabel ()` - Set/Get the labels for the x, y, and z axes. By default, use "X", "Y" and "Z".
- `vtkAxisActor2D = obj.GetXAxisActor2D ()` - Retrieve handles to the X, Y and Z axis (so that you can set their text properties for example)
- `vtkAxisActor2D = obj.GetYAxisActor2D ()` - Retrieve handles to the X, Y and Z axis (so that you can set their text properties for example)
- `vtkAxisActor2D = obj.GetZAxisActor2D ()` - Set/Get the title text property of all axes. Note that each axis can be controlled individually through the GetX/Y/ZAxisActor2D() methods.
- `obj.SetAxisTitleTextProperty (vtkTextProperty p)` - Set/Get the title text property of all axes. Note that each axis can be controlled individually through the GetX/Y/ZAxisActor2D() methods.
- `vtkTextProperty = obj.GetAxisTitleTextProperty ()` - Set/Get the title text property of all axes. Note that each axis can be controlled individually through the GetX/Y/ZAxisActor2D() methods.
- `obj.SetAxisLabelTextProperty (vtkTextProperty p)` - Set/Get the labels text property of all axes. Note that each axis can be controlled individually through the GetX/Y/ZAxisActor2D() methods.
- `vtkTextProperty = obj.GetAxisLabelTextProperty ()` - Set/Get the labels text property of all axes. Note that each axis can be controlled individually through the GetX/Y/ZAxisActor2D() methods.
- `obj.SetLabelFormat (string )` - Set/Get the format with which to print the labels on each of the x-y-z axes.
- `string = obj.GetLabelFormat ()` - Set/Get the format with which to print the labels on each of the x-y-z axes.
- `obj.SetFontFactor (double )` - Set/Get the factor that controls the overall size of the fonts used to label and title the axes.
- `double = obj.GetFontFactorMinValue ()` - Set/Get the factor that controls the overall size of the fonts used to label and title the axes.
- `double = obj.GetFontFactorMaxValue ()` - Set/Get the factor that controls the overall size of the fonts used to label and title the axes.
- `double = obj.GetFontFactor ()` - Set/Get the factor that controls the overall size of the fonts used to label and title the axes.
- `obj.SetInertia (int )` - Set/Get the inertial factor that controls how often (i.e, how many renders) the axes can switch position (jump from one axes to another).
- `int = obj.GetInertiaMinValue ()` - Set/Get the inertial factor that controls how often (i.e, how many renders) the axes can switch position (jump from one axes to another).
- `int = obj.GetInertiaMaxValue ()` - Set/Get the inertial factor that controls how often (i.e, how many renders) the axes can switch position (jump from one axes to another).
- `int = obj.GetInertia ()` - Set/Get the inertial factor that controls how often (i.e, how many renders) the axes can switch position (jump from one axes to another).
- `obj.SetShowActualBounds (int )` - Set/Get the variable that controls whether the actual bounds of the dataset are always shown. Setting this variable to 1 means that clipping is disabled and that the actual value of the bounds is displayed even with corner offsets Setting this variable to 0 means these axis will clip themselves and show variable bounds (legacy mode)

- `int = obj.GetShowActualBoundsMinValue ()` - Set/Get the variable that controls whether the actual bounds of the dataset are always shown. Setting this variable to 1 means that clipping is disabled and that the actual value of the bounds is displayed even with corner offsets Setting this variable to 0 means these axis will clip themselves and show variable bounds (legacy mode)

- `int = obj.GetShowActualBoundsMaxValue ()` - Set/Get the variable that controls whether the actual bounds of the dataset are always shown. Setting this variable to 1 means that clipping is disabled and that the actual value of the bounds is displayed even with corner offsets Setting this variable to 0 means these axis will clip themselves and show variable bounds (legacy mode)

- `int = obj.GetShowActualBounds ()` - Set/Get the variable that controls whether the actual bounds of the dataset are always shown. Setting this variable to 1 means that clipping is disabled and that the actual value of the bounds is displayed even with corner offsets Setting this variable to 0 means these axis will clip themselves and show variable bounds (legacy mode)

- `obj.SetCornerOffset (double )` - Specify an offset value to "pull back" the axes from the corner at which they are joined to avoid overlap of axes labels. The "CornerOffset" is the fraction of the axis length to pull back.

- `double = obj.GetCornerOffset ()` - Specify an offset value to "pull back" the axes from the corner at which they are joined to avoid overlap of axes labels. The "CornerOffset" is the fraction of the axis length to pull back.

- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.

- `obj.SetXAxisVisibility (int )` - Turn on and off the visibility of each axis.

- `int = obj.GetXAxisVisibility ()` - Turn on and off the visibility of each axis.

- `obj.XAxisVisibilityOn ()` - Turn on and off the visibility of each axis.

- `obj.XAxisVisibilityOff ()` - Turn on and off the visibility of each axis.

- `obj.SetYAxisVisibility (int )` - Turn on and off the visibility of each axis.

- `int = obj.GetYAxisVisibility ()` - Turn on and off the visibility of each axis.

- `obj.YAxisVisibilityOn ()` - Turn on and off the visibility of each axis.

- `obj.YAxisVisibilityOff ()` - Turn on and off the visibility of each axis.

- `obj.SetZAxisVisibility (int )` - Turn on and off the visibility of each axis.

- `int = obj.GetZAxisVisibility ()` - Turn on and off the visibility of each axis.

- `obj.ZAxisVisibilityOn ()` - Turn on and off the visibility of each axis.

- `obj.ZAxisVisibilityOff ()` - Turn on and off the visibility of each axis.

- `obj.ShallowCopy (vtkCubeAxesActor2D actor)` - Shallow copy of a CubeAxesActor2D.

- `obj.SetProp (vtkProp prop)` - @deprecated Replaced by vtkCubeAxesActor2D::SetViewProp() as of VTK 5.0.

- `vtkProp = obj.GetProp ()` - @deprecated Replaced by vtkCubeAxesActor2D::GetViewProp() as of VTK 5.0.

34.11 vtkDepthSortPolyData

34.11.1 Usage

vtkDepthSortPolyData rearranges the order of cells so that certain rendering operations (e.g., transparency or Painter's algorithms) generate correct results. To use this filter you must specify the direction vector along which to sort the cells. You can do this by specifying a camera and/or prop to define a view direction; or explicitly set a view direction.

To create an instance of class vtkDepthSortPolyData, simply invoke its constructor as follows

```
obj = vtkDepthSortPolyData
```

34.11.2 Methods

The class vtkDepthSortPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDepthSortPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDepthSortPolyData = obj.NewInstance ()`
- `vtkDepthSortPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetDirection (int )` - Specify the sort method for the polygonal primitives. By default, the poly data is sorted from back to front.
- `int = obj.GetDirection ()` - Specify the sort method for the polygonal primitives. By default, the poly data is sorted from back to front.
- `obj.SetDirectionToFrontToBack ()` - Specify the sort method for the polygonal primitives. By default, the poly data is sorted from back to front.
- `obj.SetDirectionToBackToFront ()` - Specify the sort method for the polygonal primitives. By default, the poly data is sorted from back to front.
- `obj.SetDirectionToSpecifiedVector ()` - Specify the point to use when sorting. The fastest is to just take the first cell point. Other options are to take the bounding box center or the parametric center of the cell. By default, the first cell point is used.
- `obj.SetDepthSortMode (int )` - Specify the point to use when sorting. The fastest is to just take the first cell point. Other options are to take the bounding box center or the parametric center of the cell. By default, the first cell point is used.
- `int = obj.GetDepthSortMode ()` - Specify the point to use when sorting. The fastest is to just take the first cell point. Other options are to take the bounding box center or the parametric center of the cell. By default, the first cell point is used.
- `obj.SetDepthSortModeToFirstPoint ()` - Specify the point to use when sorting. The fastest is to just take the first cell point. Other options are to take the bounding box center or the parametric center of the cell. By default, the first cell point is used.
- `obj.SetDepthSortModeToBoundsCenter ()` - Specify the point to use when sorting. The fastest is to just take the first cell point. Other options are to take the bounding box center or the parametric center of the cell. By default, the first cell point is used.

- `obj.SetDepthSortModeToParametricCenter ()` - Specify a camera that is used to define a view direction along which the cells are sorted. This ivar only has effect if the direction is set to front-to-back or back-to-front, and a camera is specified.
- `obj.SetCamera (vtkCamera )` - Specify a camera that is used to define a view direction along which the cells are sorted. This ivar only has effect if the direction is set to front-to-back or back-to-front, and a camera is specified.
- `vtkCamera = obj.GetCamera ()` - Specify a camera that is used to define a view direction along which the cells are sorted. This ivar only has effect if the direction is set to front-to-back or back-to-front, and a camera is specified.
- `obj.SetProp3D (vtkProp3D )` - Specify a transformation matrix (via the vtkProp3D::GetMatrix() method) that is used to include the effects of transformation. This ivar only has effect if the direction is set to front-to-back or back-to-front, and a camera is specified. Specifying the vtkProp3D is optional.
- `vtkProp3D = obj.GetProp3D ()` - Specify a transformation matrix (via the vtkProp3D::GetMatrix() method) that is used to include the effects of transformation. This ivar only has effect if the direction is set to front-to-back or back-to-front, and a camera is specified. Specifying the vtkProp3D is optional.
- `obj.SetVector (double , double , double )` - Set/Get the sort direction. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedVector(). The sort occurs in the direction of the vector.
- `obj.SetVector (double a[3])` - Set/Get the sort direction. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedVector(). The sort occurs in the direction of the vector.
- `double = obj. GetVector ()` - Set/Get the sort direction. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedVector(). The sort occurs in the direction of the vector.
- `obj.SetOrigin (double , double , double )` - Set/Get the sort origin. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedVector(). The sort occurs in the direction of the vector, with this point specifying the origin.
- `obj.SetOrigin (double a[3])` - Set/Get the sort origin. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedVector(). The sort occurs in the direction of the vector, with this point specifying the origin.
- `double = obj. GetOrigin ()` - Set/Get the sort origin. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedVector(). The sort occurs in the direction of the vector, with this point specifying the origin.
- `obj.SetSortScalars (int )` - Set/Get a flag that controls the generation of scalar values corresponding to the sort order. If enabled, the output of this filter will include scalar values that range from 0 to (ncells-1), where 0 is closest to the sort direction.
- `int = obj.GetSortScalars ()` - Set/Get a flag that controls the generation of scalar values corresponding to the sort order. If enabled, the output of this filter will include scalar values that range from 0 to (ncells-1), where 0 is closest to the sort direction.
- `obj.SortScalarsOn ()` - Set/Get a flag that controls the generation of scalar values corresponding to the sort order. If enabled, the output of this filter will include scalar values that range from 0 to (ncells-1), where 0 is closest to the sort direction.
- `obj.SortScalarsOff ()` - Set/Get a flag that controls the generation of scalar values corresponding to the sort order. If enabled, the output of this filter will include scalar values that range from 0 to (ncells-1), where 0 is closest to the sort direction.
- `long = obj.GetMTime ()` - Return MTime also considering the dependent objects: the camera and/or the prop3D.

34.12 vtkDSPFilterDefinition

34.12.1 Usage

vtkDSPFilterDefinition is used by vtkExodusReader, vtkExodusIIReader and vtkPExodusReader to do temporal smoothing of data

To create an instance of class vtkDSPFilterDefinition, simply invoke its constructor as follows

```
obj = vtkDSPFilterDefinition
```

34.12.2 Methods

The class vtkDSPFilterDefinition has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDSPFilterDefinition class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDSPFilterDefinition = obj.NewInstance ()`
- `vtkDSPFilterDefinition = obj.SafeDownCast (vtkObject o)`
- `obj.Copy (vtkDSPFilterDefinition other)`
- `obj.Clear ()`
- `bool = obj.IsThisInputVariableInstanceNeeded (int a\_timestep, int a\_outputTimestep)`
- `obj.PushBackNumeratorWeight (double a\_value)`
- `obj.PushBackDenominatorWeight (double a\_value)`
- `obj.PushBackForwardNumeratorWeight (double a\_value)`
- `obj.SetInputVariableName (string a\_value)`
- `obj.SetOutputVariableName (string a\_value)`
- `string = obj.GetInputVariableName ()`
- `string = obj.GetOutputVariableName ()`
- `int = obj.GetNumNumeratorWeights ()`
- `int = obj.GetNumDenominatorWeights ()`
- `int = obj.GetNumForwardNumeratorWeights ()`
- `double = obj.GetNumeratorWeight (int a\_which)`
- `double = obj.GetDenominatorWeight (int a\_which)`
- `double = obj.GetForwardNumeratorWeight (int a\_which)`

34.13 vtkDSPFilterGroup

34.13.1 Usage

vtkDSPFilterGroup is used by vtkExodusReader, vtkExodusIIReader and vtkPExodusReader to do temporal smoothing of data

To create an instance of class vtkDSPFilterGroup, simply invoke its constructor as follows

```
obj = vtkDSPFilterGroup
```

34.13.2 Methods

The class vtkDSPFilterGroup has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDSPFilterGroup class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDSPFilterGroup = obj.NewInstance ()`
- `vtkDSPFilterGroup = obj.SafeDownCast (vtkObject o)`
- `obj.AddFilter (vtkDSPFilterDefinition filter)`
- `obj.RemoveFilter (string a\_outputVariableName)`
- `bool = obj.IsThisInputVariableInstanceNeeded (string a\_name, int a\_timestep, int a\_outputTimeste`
- `bool = obj.IsThisInputVariableInstanceCached (string a\_name, int a\_timestep)`
- `obj.AddInputVariableInstance (string a\_name, int a\_timestep, vtkFloatArray a\_data)`
- `vtkFloatArray = obj.GetCachedInput (int a\_whichFilter, int a\_whichTimestep)`
- `vtkFloatArray = obj.GetCachedOutput (int a\_whichFilter, int a\_whichTimestep)`
- `string = obj.GetInputVariableName (int a\_whichFilter)`
- `int = obj.GetNumFilters ()`
- `obj.Copy (vtkDSPFilterGroup other)`
- `vtkDSPFilterDefinition = obj.GetFilter (int a\_whichFilter)`

34.14 vtkEarthSource

34.14.1 Usage

vtkEarthSource creates a spherical rendering of the geographical shapes of the major continents of the earth. The OnRatio determines how much of the data is actually used. The radius defines the radius of the sphere at which the continents are placed. Obtains data from an imbedded array of coordinates.

To create an instance of class vtkEarthSource, simply invoke its constructor as follows

```
obj = vtkEarthSource
```

34.14.2 Methods

The class vtkEarthSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEarthSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEarthSource = obj.NewInstance ()`
- `vtkEarthSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetRadius (double )` - Set radius of earth.
- `double = obj.GetRadiusMinValue ()` - Set radius of earth.
- `double = obj.GetRadiusMaxValue ()` - Set radius of earth.
- `double = obj.GetRadius ()` - Set radius of earth.
- `obj.SetOnRatio (int )` - Turn on every nth entity. This controls how much detail the model will have. The maximum ratio is sixteen. (The smaller OnRatio, the more detail there is.)
- `int = obj.GetOnRatioMinValue ()` - Turn on every nth entity. This controls how much detail the model will have. The maximum ratio is sixteen. (The smaller OnRatio, the more detail there is.)
- `int = obj.GetOnRatioMaxValue ()` - Turn on every nth entity. This controls how much detail the model will have. The maximum ratio is sixteen. (The smaller OnRatio, the more detail there is.)
- `int = obj.GetOnRatio ()` - Turn on every nth entity. This controls how much detail the model will have. The maximum ratio is sixteen. (The smaller OnRatio, the more detail there is.)
- `obj.SetOutline (int )` - Turn on/off drawing continents as filled polygons or as wireframe outlines. Warning: some graphics systems will have trouble with the very large, concave filled polygons. Recommend you use OutlienOn (i.e., disable filled polygons) for now.
- `int = obj.GetOutline ()` - Turn on/off drawing continents as filled polygons or as wireframe outlines. Warning: some graphics systems will have trouble with the very large, concave filled polygons. Recommend you use OutlienOn (i.e., disable filled polygons) for now.
- `obj.OutlineOn ()` - Turn on/off drawing continents as filled polygons or as wireframe outlines. Warning: some graphics systems will have trouble with the very large, concave filled polygons. Recommend you use OutlienOn (i.e., disable filled polygons) for now.
- `obj.OutlineOff ()` - Turn on/off drawing continents as filled polygons or as wireframe outlines. Warning: some graphics systems will have trouble with the very large, concave filled polygons. Recommend you use OutlienOn (i.e., disable filled polygons) for now.

34.15 vtkExodusIIReader

34.15.1 Usage

vtkExodusIIReader is a unstructured grid source object that reads ExodusII files. Most of the meta data associated with the file is loaded when UpdateInformation is called. This includes information like Title, number of blocks, number and names of arrays. This data can be retrieved from methods in this reader. Separate arrays that are meant to be a single vector, are combined internally for convenience. To be combined, the array names have to be identical except for a trailing X,Y and Z (or x,y,z). By default cell and point

arrays are not loaded. However, the user can flag arrays to load with the methods "SetPointArrayStatus" and "SetCellArrayStatus". The reader DOES NOT respond to piece requests

To create an instance of class vtkExodusIIReader, simply invoke its constructor as follows

```
obj = vtkExodusIIReader
```

34.15.2 Methods

The class vtkExodusIIReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExodusIIReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExodusIIReader = obj.NewInstance ()`
- `vtkExodusIIReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string fname)` - Determine if the file can be readed with this reader.
- `long = obj.GetMTime ()` - Return the object's MTime. This is overridden to include the timestamp of its internal class.
- `long = obj.GetMetadataMTime ()` - Return the MTime of the internal data structure. This is really only intended for use by vtkPExodusIIReader in order to determine if the filename is newer than the metadata.
- `obj.SetFileName (string fname)` - Specify file name of the Exodus file.
- `string = obj.GetFileName ()` - Specify file name of the Exodus file.
- `obj.SetXMLFileName (string fname)` - Specify file name of the xml file.
- `string = obj.GetXMLFileName ()` - Specify file name of the xml file.
- `obj.SetTimeStep (int )` - Which TimeStep to read.
- `int = obj.GetTimeStep ()` - Which TimeStep to read.
- `obj.SetModeShape (int val)` - Returns the available range of valid integer time steps.
- `int = obj. GetTimeStepRange ()` - Returns the available range of valid integer time steps.
- `obj.SetTimeStepRange (int , int )` - Returns the available range of valid integer time steps.
- `obj.SetTimeStepRange (int  a[2])` - Returns the available range of valid integer time steps.
- `obj.SetGenerateObjectIdCellArray (int g)` - Extra cell data array that can be generated. By default, this array is ON. The value of the array is the integer id found in the exodus file. The name of the array is returned by GetBlockIdArrayName(). For cells representing elements from an Exodus element block, this is set to the element block ID. For cells representing edges from an Exodus edge block, this is the edge block ID. Similarly, ths is the face block ID for cells representing faces from an Exodus face block. The same holds for cells representing entries of node, edge, face, side, and element sets.

- `int = obj.GetGenerateObjectIdCellArray ()` - Extra cell data array that can be generated. By default, this array is ON. The value of the array is the integer id found in the exodus file. The name of the array is returned by GetBlockIdArrayName(). For cells representing elements from an Exodus element block, this is set to the element block ID. For cells representing edges from an Exodus edge block, this is the edge block ID. Similarly, ths is the face block ID for cells representing faces from an Exodus face block. The same holds for cells representing entries of node, edge, face, side, and element sets.
- `obj.GenerateObjectIdCellArrayOn ()` - Extra cell data array that can be generated. By default, this array is ON. The value of the array is the integer id found in the exodus file. The name of the array is returned by GetBlockIdArrayName(). For cells representing elements from an Exodus element block, this is set to the element block ID. For cells representing edges from an Exodus edge block, this is the edge block ID. Similarly, ths is the face block ID for cells representing faces from an Exodus face block. The same holds for cells representing entries of node, edge, face, side, and element sets.
- `obj.GenerateObjectIdCellArrayOff ()` - Extra cell data array that can be generated. By default, this array is ON. The value of the array is the integer id found in the exodus file. The name of the array is returned by GetBlockIdArrayName(). For cells representing elements from an Exodus element block, this is set to the element block ID. For cells representing edges from an Exodus edge block, this is the edge block ID. Similarly, ths is the face block ID for cells representing faces from an Exodus face block. The same holds for cells representing entries of node, edge, face, side, and element sets.
- `obj.SetGenerateGlobalElementIdArray (int g)`
- `int = obj.GetGenerateGlobalElementIdArray ()`
- `obj.GenerateGlobalElementIdArrayOn ()`
- `obj.GenerateGlobalElementIdArrayOff ()`
- `obj.SetGenerateGlobalNodeIdArray (int g)`
- `int = obj.GetGenerateGlobalNodeIdArray ()`
- `obj.GenerateGlobalNodeIdArrayOn ()`
- `obj.GenerateGlobalNodeIdArrayOff ()`
- `obj.SetGenerateImplicitElementIdArray (int g)`
- `int = obj.GetGenerateImplicitElementIdArray ()`
- `obj.GenerateImplicitElementIdArrayOn ()`
- `obj.GenerateImplicitElementIdArrayOff ()`
- `obj.SetGenerateImplicitNodeIdArray (int g)`
- `int = obj.GetGenerateImplicitNodeIdArray ()`
- `obj.GenerateImplicitNodeIdArrayOn ()`
- `obj.GenerateImplicitNodeIdArrayOff ()`
- `obj.SetGenerateFileIdArray (int f)`
- `int = obj.GetGenerateFileIdArray ()`
- `obj.GenerateFileIdArrayOn ()`
- `obj.GenerateFileIdArrayOff ()`

- `obj.SetFileId (int f)`
- `int = obj.GetFileId ()`
- `obj.SetApplyDisplacements (int d)` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `int = obj.GetApplyDisplacements ()` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `obj.ApplyDisplacementsOn ()` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `obj.ApplyDisplacementsOff ()` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `obj.SetDisplacementMagnitude (float s)` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `float = obj.GetDisplacementMagnitude ()` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `obj.SetHasModeShapes (int ms)` - Set/Get whether the Exodus sequence number corresponds to time steps or mode shapes. By default, HasModeShapes is false unless two time values in the Exodus file are identical, in which case it is true.
- `int = obj.GetHasModeShapes ()` - Set/Get whether the Exodus sequence number corresponds to time steps or mode shapes. By default, HasModeShapes is false unless two time values in the Exodus file are identical, in which case it is true.
- `obj.HasModeShapesOn ()` - Set/Get whether the Exodus sequence number corresponds to time steps or mode shapes. By default, HasModeShapes is false unless two time values in the Exodus file are identical, in which case it is true.
- `obj.HasModeShapesOff ()` - Set/Get whether the Exodus sequence number corresponds to time steps or mode shapes. By default, HasModeShapes is false unless two time values in the Exodus file are identical, in which case it is true.
- `obj.SetModeShapeTime (double phase)` - Set/Get the time used to animate mode shapes. This is a number between 0 and 1 that is used to scale the DisplacementMagnitude in a sinusoidal pattern. Specifically, the displacement vector for each vertex is scaled by DisplacementMagnitude$cos(2i$ModeShapeTime) before it is added to the vertex coordinates.
- `double = obj.GetModeShapeTime ()` - Set/Get the time used to animate mode shapes. This is a number between 0 and 1 that is used to scale the DisplacementMagnitude in a sinusoidal pattern. Specifically, the displacement vector for each vertex is scaled by DisplacementMagnitude$cos(2i$ModeShapeTime) before it is added to the vertex coordinates.
- `obj.SetAnimateModeShapes (int flag)` - If this flag is on (the default) and HasModeShapes is also on, then this reader will report a continuous time range [0,1] and animate the displacements in a periodic sinusoid. If this flag is off and HasModeShapes is on, this reader ignores time. This flag has no effect if HasModeShapes is off.

- `int = obj.GetAnimateModeShapes ()` - If this flag is on (the default) and HasModeShapes is also on, then this reader will report a continuous time range [0,1] and animate the displacements in a periodic sinusoid. If this flag is off and HasModeShapes is on, this reader ignores time. This flag has no effect if HasModeShapes is off.
- `obj.AnimateModeShapesOn ()` - If this flag is on (the default) and HasModeShapes is also on, then this reader will report a continuous time range [0,1] and animate the displacements in a periodic sinusoid. If this flag is off and HasModeShapes is on, this reader ignores time. This flag has no effect if HasModeShapes is off.
- `obj.AnimateModeShapesOff ()` - If this flag is on (the default) and HasModeShapes is also on, then this reader will report a continuous time range [0,1] and animate the displacements in a periodic sinusoid. If this flag is off and HasModeShapes is on, this reader ignores time. This flag has no effect if HasModeShapes is off.
- `obj.SetEdgeFieldDecorations (int d)` - FIXME
- `int = obj.GetEdgeFieldDecorations ()` - FIXME
- `obj.EdgeFieldDecorationsNone ()` - FIXME
- `obj.EdgeFieldDecorationsGlyphs ()` - FIXME
- `obj.EdgeFieldDecorationsCornerAveraged ()` - FIXME
- `obj.SetFaceFieldDecorations (int d)` - FIXME
- `int = obj.GetFaceFieldDecorations ()` - FIXME
- `obj.FaceFieldDecorationsNone ()` - FIXME
- `obj.FaceFieldDecorationsGlyphs ()` - FIXME
- `obj.FaceFieldDecorationsCornerAveraged ()` - Access to meta data generated by UpdateInformation.
- `string = obj.GetTitle ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetDimensionality ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetNumberOfTimeSteps ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetNumberOfNodesInFile ()`
- `int = obj.GetNumberOfEdgesInFile ()`
- `int = obj.GetNumberOfFacesInFile ()`
- `int = obj.GetNumberOfElementsInFile ()`
- `int = obj.GetObjectTypeFromName (string name)`
- `string = obj.GetObjectTypeName (int )`
- `int = obj.GetNumberOfNodes ()`
- `int = obj.GetNumberOfObjects (int objectType)`
- `int = obj.GetNumberOfEntriesInObject (int objectType, int objectIndex)`
- `int = obj.GetObjectId (int objectType, int objectIndex)`
- `string = obj.GetObjectName (int objectType, int objectIndex)`

- int = obj.GetObjectIndex (int objectType, string objectName)
- int = obj.GetObjectIndex (int objectType, int id)
- int = obj.GetObjectStatus (int objectType, int objectIndex)
- int = obj.GetObjectStatus (int objectType, string objectName)
- obj.SetObjectStatus (int objectType, int objectIndex, int status)
- obj.SetObjectStatus (int objectType, string objectName, int status)
- int = obj.GetNumberOfObjectArrays (int objectType)
- string = obj.GetObjectArrayName (int objectType, int arrayIndex)
- int = obj.GetObjectArrayIndex (int objectType, string arrayName)
- int = obj.GetNumberOfObjectArrayComponents (int objectType, int arrayIndex)
- int = obj.GetObjectArrayStatus (int objectType, int arrayIndex)
- int = obj.GetObjectArrayStatus (int objectType, string arrayName)
- obj.SetObjectArrayStatus (int objectType, int arrayIndex, int status)
- obj.SetObjectArrayStatus (int objectType, string arrayName, int status)
- int = obj.GetNumberOfObjectAttributes (int objectType, int objectIndex)
- string = obj.GetObjectAttributeName (int objectType, int objectIndex, int attribIndex)
- int = obj.GetObjectAttributeIndex (int objectType, int objectIndex, string attribName)
- int = obj.GetObjectAttributeStatus (int objectType, int objectIndex, int attribIndex)
- int = obj.GetObjectAttributeStatus (int objectType, int objectIndex, string attribName)
- obj.SetObjectAttributeStatus (int objectType, int objectIndex, int attribIndex, int status)
- obj.SetObjectAttributeStatus (int objectType, int objectIndex, string attribName, int status)
- vtkIdType = obj.GetTotalNumberOfNodes ()
- vtkIdType = obj.GetTotalNumberOfEdges ()
- vtkIdType = obj.GetTotalNumberOfFaces ()
- vtkIdType = obj.GetTotalNumberOfElements ()
- int = obj.GetNumberOfPartArrays ()
- string = obj.GetPartArrayName (int arrayIdx)
- int = obj.GetPartArrayID (string name)
- string = obj.GetPartBlockInfo (int arrayIdx)
- obj.SetPartArrayStatus (int index, int flag)
- obj.SetPartArrayStatus (string , int flag)
- int = obj.GetPartArrayStatus (int index)
- int = obj.GetPartArrayStatus (string)

- int = obj.GetNumberOfMaterialArrays ()
- string = obj.GetMaterialArrayName (int arrayIdx)
- int = obj.GetMaterialArrayID (string name)
- obj.SetMaterialArrayStatus (int index, int flag)
- obj.SetMaterialArrayStatus (string , int flag)
- int = obj.GetMaterialArrayStatus (int index)
- int = obj.GetMaterialArrayStatus (string)
- int = obj.GetNumberOfAssemblyArrays ()
- string = obj.GetAssemblyArrayName (int arrayIdx)
- int = obj.GetAssemblyArrayID (string name)
- obj.SetAssemblyArrayStatus (int index, int flag)
- obj.SetAssemblyArrayStatus (string , int flag)
- int = obj.GetAssemblyArrayStatus (int index)
- int = obj.GetAssemblyArrayStatus (string)
- int = obj.GetNumberOfHierarchyArrays ()
- string = obj.GetHierarchyArrayName (int arrayIdx)
- obj.SetHierarchyArrayStatus (int index, int flag)
- obj.SetHierarchyArrayStatus (string , int flag)
- int = obj.GetHierarchyArrayStatus (int index)
- int = obj.GetHierarchyArrayStatus (string)
- int = obj.GetDisplayType ()
- obj.SetDisplayType (int type)
- obj.ExodusModelMetadataOn ()
- obj.ExodusModelMetadataOff ()
- obj.SetExodusModelMetadata (int)
- int = obj.GetExodusModelMetadata ()
- vtkExodusModel = obj.GetExodusModel () - Returns the object which encapsulates the model metadata.
- obj.SetPackExodusModelOntoOutput (int)
- int = obj.GetPackExodusModelOntoOutput ()
- obj.PackExodusModelOntoOutputOn ()
- obj.PackExodusModelOntoOutputOff ()
- int = obj.IsValidVariable (string type, string name)

- int = obj.GetVariableID (string type, string name)
- obj.SetAllArrayStatus (int otype, int status)
- int = obj.GetTimeSeriesData (int ID, string vName, string vType, vtkFloatArray result)
- int = obj.GetNumberOfEdgeBlockArrays ()
- string = obj.GetEdgeBlockArrayName (int index)
- int = obj.GetEdgeBlockArrayStatus (string name)
- obj.SetEdgeBlockArrayStatus (string name, int flag)
- int = obj.GetNumberOfFaceBlockArrays ()
- string = obj.GetFaceBlockArrayName (int index)
- int = obj.GetFaceBlockArrayStatus (string name)
- obj.SetFaceBlockArrayStatus (string name, int flag)
- int = obj.GetNumberOfElementBlockArrays ()
- string = obj.GetElementBlockArrayName (int index)
- int = obj.GetElementBlockArrayStatus (string name)
- obj.SetElementBlockArrayStatus (string name, int flag)
- int = obj.GetNumberOfGlobalResultArrays ()
- string = obj.GetGlobalResultArrayName (int index)
- int = obj.GetGlobalResultArrayStatus (string name)
- obj.SetGlobalResultArrayStatus (string name, int flag)
- int = obj.GetNumberOfPointResultArrays ()
- string = obj.GetPointResultArrayName (int index)
- int = obj.GetPointResultArrayStatus (string name)
- obj.SetPointResultArrayStatus (string name, int flag)
- int = obj.GetNumberOfEdgeResultArrays ()
- string = obj.GetEdgeResultArrayName (int index)
- int = obj.GetEdgeResultArrayStatus (string name)
- obj.SetEdgeResultArrayStatus (string name, int flag)
- int = obj.GetNumberOfFaceResultArrays ()
- string = obj.GetFaceResultArrayName (int index)
- int = obj.GetFaceResultArrayStatus (string name)
- obj.SetFaceResultArrayStatus (string name, int flag)
- int = obj.GetNumberOfElementResultArrays ()
- string = obj.GetElementResultArrayName (int index)

- int = obj.GetElementResultArrayStatus (string name)
- obj.SetElementResultArrayStatus (string name, int flag)
- int = obj.GetNumberOfNodeMapArrays ()
- string = obj.GetNodeMapArrayName (int index)
- int = obj.GetNodeMapArrayStatus (string name)
- obj.SetNodeMapArrayStatus (string name, int flag)
- int = obj.GetNumberOfEdgeMapArrays ()
- string = obj.GetEdgeMapArrayName (int index)
- int = obj.GetEdgeMapArrayStatus (string name)
- obj.SetEdgeMapArrayStatus (string name, int flag)
- int = obj.GetNumberOfFaceMapArrays ()
- string = obj.GetFaceMapArrayName (int index)
- int = obj.GetFaceMapArrayStatus (string name)
- obj.SetFaceMapArrayStatus (string name, int flag)
- int = obj.GetNumberOfElementMapArrays ()
- string = obj.GetElementMapArrayName (int index)
- int = obj.GetElementMapArrayStatus (string name)
- obj.SetElementMapArrayStatus (string name, int flag)
- int = obj.GetNumberOfNodeSetArrays ()
- string = obj.GetNodeSetArrayName (int index)
- int = obj.GetNodeSetArrayStatus (string name)
- obj.SetNodeSetArrayStatus (string name, int flag)
- int = obj.GetNumberOfSideSetArrays ()
- string = obj.GetSideSetArrayName (int index)
- int = obj.GetSideSetArrayStatus (string name)
- obj.SetSideSetArrayStatus (string name, int flag)
- int = obj.GetNumberOfEdgeSetArrays ()
- string = obj.GetEdgeSetArrayName (int index)
- int = obj.GetEdgeSetArrayStatus (string name)
- obj.SetEdgeSetArrayStatus (string name, int flag)
- int = obj.GetNumberOfFaceSetArrays ()
- string = obj.GetFaceSetArrayName (int index)
- int = obj.GetFaceSetArrayStatus (string name)

- `obj.SetFaceSetArrayStatus (string name, int flag)`
- `int = obj.GetNumberOfElementSetArrays ()`
- `string = obj.GetElementSetArrayName (int index)`
- `int = obj.GetElementSetArrayStatus (string name)`
- `obj.SetElementSetArrayStatus (string name, int flag)`
- `int = obj.GetNumberOfNodeSetResultArrays ()`
- `string = obj.GetNodeSetResultArrayName (int index)`
- `int = obj.GetNodeSetResultArrayStatus (string name)`
- `obj.SetNodeSetResultArrayStatus (string name, int flag)`
- `int = obj.GetNumberOfSideSetResultArrays ()`
- `string = obj.GetSideSetResultArrayName (int index)`
- `int = obj.GetSideSetResultArrayStatus (string name)`
- `obj.SetSideSetResultArrayStatus (string name, int flag)`
- `int = obj.GetNumberOfEdgeSetResultArrays ()`
- `string = obj.GetEdgeSetResultArrayName (int index)`
- `int = obj.GetEdgeSetResultArrayStatus (string name)`
- `obj.SetEdgeSetResultArrayStatus (string name, int flag)`
- `int = obj.GetNumberOfFaceSetResultArrays ()`
- `string = obj.GetFaceSetResultArrayName (int index)`
- `int = obj.GetFaceSetResultArrayStatus (string name)`
- `obj.SetFaceSetResultArrayStatus (string name, int flag)`
- `int = obj.GetNumberOfElementSetResultArrays ()`
- `string = obj.GetElementSetResultArrayName (int index)`
- `int = obj.GetElementSetResultArrayStatus (string name)`
- `obj.SetElementSetResultArrayStatus (string name, int flag)` - Set the fast-path keys. All three must be set for the fast-path option to work. Possible argument values: "POINT","CELL","EDGE","FACE"
- `obj.SetFastPathObjectType (string type)` - Set the fast-path keys. All three must be set for the fast-path option to work. Possible argument values: "POINT","CELL","EDGE","FACE"
- `obj.SetFastPathIdType (string type)` - Possible argument values: "INDEX","GLOBAL" "GLOBAL" means the id refers to a global id "INDEX" means the id refers to an index into the VTK array
- `obj.SetFastPathObjectId (vtkIdType id)` - Possible argument values: "INDEX","GLOBAL" "GLOBAL" means the id refers to a global id "INDEX" means the id refers to an index into the VTK array
- `obj.Reset ()` - Reset the user-specified parameters and flush internal arrays so that the reader state is just as it was after the reader was instantiated.

  It doesn't make sense to let users reset only the internal state; both the settings and the state are changed by this call.

- `obj.ResetSettings ()` - Reset the user-specified parameters to their default values. The only settings not affected are the filename and/or pattern because these have no default.

  Resetting the settings but not the state allows users to keep the active cache but return to initial array selections, etc.

- `obj.ResetCache ()` - Clears out the cache entries.

- `obj.UpdateTimeInformation ()` - Re-reads time information from the exodus file and updates TimeStepRange accordingly.

- `obj.Dump ()`

- `vtkGraph = obj.GetSIL ()` - SIL describes organization of/relationships between classifications eg. blocks/materials/hierarchies.

- `int = obj.GetSILUpdateStamp ()` - Every time the SIL is updated a this will return a different value.

- `bool = obj.GetProducedFastPathOutput ()` - HACK: Used by vtkPExodusIIReader to tell is the reader produced a valid fast path output.

34.16 vtkExodusModel

34.16.1 Usage

A vtkUnstructuredGrid output by vtkExodusReader or vtkPExodusReader is missing a great deal of initialization and static model data that is in an Exodus II file. (Global variables, properties, node sets, side sets, and so on.) This data can be stored in a vtkModelMetadata object, which can be initialized using this vtkExodusModel class.

This class can be initialized with a file handle for an open Exodus file, and the vtkUnstructuredGrid derived from that file. The methods used would be SetGlobalInformation, SetLocalInformation, AddUGridElementVariable and AddUGridNodeVariable. The vtkExodusReader does this.

It can also be initialized (using UnpackExodusModel) from a vtkUnstructuredGrid that has had metadata packed into it's field arrays with PackExodusModel. The vtkExodusIIWriter does this.

If you plan to write out the Exodus file (with vtkExodusIIWriter), you should direct the Exodus reader to create a vtkExodusModel object. This will be used by the Exodus writer to create a correct Exodus II file on output. In addition, the vtkDistributedDataFilter is cognizant of the ExodusModel object and will unpack, extract, merge, and pack these objects associated with the grids it is partitioning.

.SECTION See also vtkExodusReader vtkPExodusReader vtkExodusIIWriter vtkModelMetadata vtkDistributedDataFilter

To create an instance of class vtkExodusModel, simply invoke its constructor as follows

```
obj = vtkExodusModel
```

34.16.2 Methods

The class vtkExodusModel has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExodusModel class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkExodusModel = obj.NewInstance ()`

- `vtkExodusModel = obj.SafeDownCast (vtkObject o)`

- `int = obj.SetGlobalInformation (int fid, int compute\_word\_size)`
- `int = obj.AddUGridElementVariable (string ugridVarName, string origName, int numComponents)`
- `int = obj.RemoveUGridElementVariable (string ugridVarName)`
- `int = obj.AddUGridNodeVariable (string ugridVarName, string origName, int numComponents)`
- `int = obj.RemoveUGridNodeVariable (string ugridVarName)`
- `int = obj.SetLocalInformation (vtkUnstructuredGrid ugrid, int fid, int timeStep, int newGeometry, i`
- `vtkModelMetadata = obj.GetModelMetadata ()`
- `obj.SetModelMetadata (vtkModelMetadata emData)`
- `int = obj.UnpackExodusModel (vtkUnstructuredGrid grid, int deleteIt)`
- `int = obj.MergeExodusModel (vtkExodusModel em)`
- `vtkExodusModel = obj.ExtractExodusModel (vtkIdTypeArray globalCellIdList, vtkUnstructuredGrid grid)`
- `obj.PackExodusModel (vtkUnstructuredGrid grid)`
- `obj.Reset ()`

34.17 vtkExodusReader

34.17.1 Usage

vtkExodusReader is a unstructured grid source object that reads ExodusII files. Most of the meta data associated with the file is loaded when UpdateInformation is called. This includes information like Title, number of blocks, number and names of arrays. This data can be retrieved from methods in this reader. Separate arrays that are meant to be a single vector, are combined internally for convenience. To be combined, the array names have to be identical except for a trailing X,Y and Z (or x,y,z). By default cell and point arrays are not loaded. However, the user can flag arrays to load with the methods "SetPointArrayStatus" and "SetCellArrayStatus". The reader DOES NOT respond to piece requests

To create an instance of class vtkExodusReader, simply invoke its constructor as follows

```
obj = vtkExodusReader
```

34.17.2 Methods

The class vtkExodusReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExodusReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExodusReader = obj.NewInstance ()`
- `vtkExodusReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string fname)` - Determine if the file can be readed with this reader.
- `obj.SetFileName (string )` - Specify file name of the Exodus file.
- `string = obj.GetFileName ()` - Specify file name of the Exodus file.

- `obj.SetXMLFileName (string )` - Specify file name of the xml file.
- `string = obj.GetXMLFileName ()` - Specify file name of the xml file.
- `obj.SetTimeStep (int )` - Which TimeStep to read.
- `int = obj.GetTimeStep ()` - Which TimeStep to read.
- `obj.SetGenerateBlockIdCellArray (int )` - Extra cell data array that can be generated. By default, this array is ON. The value of the array is the integer id found in the exodus file. The name of the array is returned by GetBlockIdArrayName()
- `int = obj.GetGenerateBlockIdCellArray ()` - Extra cell data array that can be generated. By default, this array is ON. The value of the array is the integer id found in the exodus file. The name of the array is returned by GetBlockIdArrayName()
- `obj.GenerateBlockIdCellArrayOn ()` - Extra cell data array that can be generated. By default, this array is ON. The value of the array is the integer id found in the exodus file. The name of the array is returned by GetBlockIdArrayName()
- `obj.GenerateBlockIdCellArrayOff ()` - Extra cell data array that can be generated. By default, this array is ON. The value of the array is the integer id found in the exodus file. The name of the array is returned by GetBlockIdArrayName()
- `string = obj.GetBlockIdArrayName ()` - Extra cell data array that can be generated. By default, this array is off. The value of the array is the integer global id of the cell. The name of the array is returned by GetGlobalElementIdArrayName()
- `obj.SetGenerateGlobalElementIdArray (int )` - Extra cell data array that can be generated. By default, this array is off. The value of the array is the integer global id of the cell. The name of the array is returned by GetGlobalElementIdArrayName()
- `int = obj.GetGenerateGlobalElementIdArray ()` - Extra cell data array that can be generated. By default, this array is off. The value of the array is the integer global id of the cell. The name of the array is returned by GetGlobalElementIdArrayName()
- `obj.GenerateGlobalElementIdArrayOn ()` - Extra cell data array that can be generated. By default, this array is off. The value of the array is the integer global id of the cell. The name of the array is returned by GetGlobalElementIdArrayName()
- `obj.GenerateGlobalElementIdArrayOff ()` - Extra cell data array that can be generated. By default, this array is off. The value of the array is the integer global id of the cell. The name of the array is returned by GetGlobalElementIdArrayName()
- `obj.SetGenerateGlobalNodeIdArray (int )` - Extra point data array that can be generated. By default, this array is ON. The value of the array is the integer id of the node. The id is relative to the entire data set. The name of the array is returned by GlobalNodeIdArrayName().
- `int = obj.GetGenerateGlobalNodeIdArray ()` - Extra point data array that can be generated. By default, this array is ON. The value of the array is the integer id of the node. The id is relative to the entire data set. The name of the array is returned by GlobalNodeIdArrayName().
- `obj.GenerateGlobalNodeIdArrayOn ()` - Extra point data array that can be generated. By default, this array is ON. The value of the array is the integer id of the node. The id is relative to the entire data set. The name of the array is returned by GlobalNodeIdArrayName().
- `obj.GenerateGlobalNodeIdArrayOff ()` - Extra point data array that can be generated. By default, this array is ON. The value of the array is the integer id of the node. The id is relative to the entire data set. The name of the array is returned by GlobalNodeIdArrayName().

- `obj.SetApplyDisplacements (int )` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `int = obj.GetApplyDisplacements ()` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `obj.ApplyDisplacementsOn ()` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `obj.ApplyDisplacementsOff ()` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `obj.SetDisplacementMagnitude (float )` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `float = obj.GetDisplacementMagnitude ()` - Geometric locations can include displacements. By default, this is ON. The nodal positions are 'displaced' by the standard exodus displacment vector. If displacements are turned 'off', the user can explicitly add them by applying a warp filter.
- `string = obj.GetTitle ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetDimensionality ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetNumberOfTimeSteps ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetNumberOfElements ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetNumberOfNodeSets ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetNumberOfSideSets ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetNumberOfBlocks ()` - Access to meta data generated by UpdateInformation.
- `int = obj. GetTimeStepRange ()` - Access to meta data generated by UpdateInformation.
- `obj.SetTimeStepRange (int , int )` - Access to meta data generated by UpdateInformation.
- `obj.SetTimeStepRange (int  a[2])` - Access to meta data generated by UpdateInformation.
- `int = obj.GetNumberOfNodes ()` - Access to meta data generated by UpdateInformation.
- `int = obj.GetNumberOfElementsInBlock (int block\_idx)` - Access to meta data generated by UpdateInformation.
- `int = obj.GetBlockId (int block\_idx)` - Access to meta data generated by UpdateInformation.
- `int = obj.GetTotalNumberOfNodes ()`
- `int = obj.GetNumberOfPointArrays ()`
- `string = obj.GetPointArrayName (int index)`
- `int = obj.GetPointArrayID (string name)`
- `int = obj.GetPointArrayNumberOfComponents (int index)`
- `obj.SetPointArrayStatus (int index, int flag)`

- `obj.SetPointArrayStatus (string , int flag)`
- `int = obj.GetPointArrayStatus (int index)`
- `int = obj.GetPointArrayStatus (string )`
- `int = obj.GetNumberOfCellArrays ()`
- `string = obj.GetCellArrayName (int index)`
- `int = obj.GetCellArrayID (string name)`
- `int = obj.GetCellArrayNumberOfComponents (int index)`
- `obj.SetCellArrayStatus (int index, int flag)`
- `obj.SetCellArrayStatus (string , int flag)`
- `int = obj.GetCellArrayStatus (int index)`
- `int = obj.GetCellArrayStatus (string )`
- `int = obj.GetTotalNumberOfElements ()`
- `int = obj.GetNumberOfBlockArrays ()`
- `string = obj.GetBlockArrayName (int index)`
- `int = obj.GetBlockArrayID (string name)`
- `obj.SetBlockArrayStatus (int index, int flag)`
- `obj.SetBlockArrayStatus (string , int flag)`
- `int = obj.GetBlockArrayStatus (int index)`
- `int = obj.GetBlockArrayStatus (string )`
- `int = obj.GetNumberOfNodeSetArrays ()` - By default Node/Side sets are not loaded, These methods allow the user to select which Node/Side sets they want to load. NumberOfNodeSets and NumberOfSideSets (set by vtk macros) are stored in vtkExodusReader but other Node/Side set metadata are stored in vtkExodusMetaData Note: GetNumberOfNodeSetArrays and GetNumberOfSideSetArrays are just syntatic sugar for paraview server xml
- `int = obj.GetNodeSetArrayStatus (int index)` - By default Node/Side sets are not loaded, These methods allow the user to select which Node/Side sets they want to load. NumberOfNodeSets and NumberOfSideSets (set by vtk macros) are stored in vtkExodusReader but other Node/Side set metadata are stored in vtkExodusMetaData Note: GetNumberOfNodeSetArrays and GetNumberOfSideSetArrays are just syntatic sugar for paraview server xml
- `int = obj.GetNodeSetArrayStatus (string name)` - By default Node/Side sets are not loaded, These methods allow the user to select which Node/Side sets they want to load. NumberOfNodeSets and NumberOfSideSets (set by vtk macros) are stored in vtkExodusReader but other Node/Side set metadata are stored in vtkExodusMetaData Note: GetNumberOfNodeSetArrays and GetNumberOfSideSetArrays are just syntatic sugar for paraview server xml
- `obj.SetNodeSetArrayStatus (int index, int flag)` - By default Node/Side sets are not loaded, These methods allow the user to select which Node/Side sets they want to load. NumberOfNodeSets and NumberOfSideSets (set by vtk macros) are stored in vtkExodusReader but other Node/Side set metadata are stored in vtkExodusMetaData Note: GetNumberOfNodeSetArrays and GetNumberOfSideSetArrays are just syntatic sugar for paraview server xml

- `obj.SetNodeSetArrayStatus (string name, int flag)` - By default Node/Side sets are not loaded, These methods allow the user to select which Node/Side sets they want to load. NumberOfNodeSets and NumberOfSideSets (set by vtk macros) are stored in vtkExodusReader but other Node/Side set metadata are stored in vtkExodusMetaData Note: GetNumberOfNodeSetArrays and GetNumberOfSideSetArrays are just syntatic sugar for paraview server xml
- `string = obj.GetNodeSetArrayName (int index)` - By default Node/Side sets are not loaded, These methods allow the user to select which Node/Side sets they want to load. NumberOfNodeSets and NumberOfSideSets (set by vtk macros) are stored in vtkExodusReader but other Node/Side set metadata are stored in vtkExodusMetaData Note: GetNumberOfNodeSetArrays and GetNumberOfSideSetArrays are just syntatic sugar for paraview server xml
- `int = obj.GetNumberOfSideSetArrays ()`
- `int = obj.GetSideSetArrayStatus (int index)`
- `int = obj.GetSideSetArrayStatus (string name)`
- `obj.SetSideSetArrayStatus (int index, int flag)`
- `obj.SetSideSetArrayStatus (string name, int flag)`
- `string = obj.GetSideSetArrayName (int index)`
- `int = obj.GetNumberOfPartArrays ()`
- `string = obj.GetPartArrayName (int arrayIdx)`
- `int = obj.GetPartArrayID (string name)`
- `string = obj.GetPartBlockInfo (int arrayIdx)`
- `obj.SetPartArrayStatus (int index, int flag)`
- `obj.SetPartArrayStatus (string , int flag)`
- `int = obj.GetPartArrayStatus (int index)`
- `int = obj.GetPartArrayStatus (string )`
- `int = obj.GetNumberOfMaterialArrays ()`
- `string = obj.GetMaterialArrayName (int arrayIdx)`
- `int = obj.GetMaterialArrayID (string name)`
- `obj.SetMaterialArrayStatus (int index, int flag)`
- `obj.SetMaterialArrayStatus (string , int flag)`
- `int = obj.GetMaterialArrayStatus (int index)`
- `int = obj.GetMaterialArrayStatus (string )`
- `int = obj.GetNumberOfAssemblyArrays ()`
- `string = obj.GetAssemblyArrayName (int arrayIdx)`
- `int = obj.GetAssemblyArrayID (string name)`
- `obj.SetAssemblyArrayStatus (int index, int flag)`
- `obj.SetAssemblyArrayStatus (string , int flag)`

- int = obj.GetAssemblyArrayStatus (int index)
- int = obj.GetAssemblyArrayStatus (string)
- int = obj.GetNumberOfHierarchyArrays ()
- string = obj.GetHierarchyArrayName (int arrayIdx)
- obj.SetHierarchyArrayStatus (int index, int flag)
- obj.SetHierarchyArrayStatus (string , int flag)
- int = obj.GetHierarchyArrayStatus (int index)
- int = obj.GetHierarchyArrayStatus (string)
- int = obj.GetHasModeShapes () - Some simulations overload the Exodus time steps to represent mode shapes. In this case, it does not make sense to iterate over the "time steps", because they are not meant to be played in order. Rather, each represents the vibration at a different "mode." Setting this to 1 changes the semantics of the reader to not report the time steps to downstream filters. By default, this is off, which is the case for most Exodus files.
- obj.SetHasModeShapes (int) - Some simulations overload the Exodus time steps to represent mode shapes. In this case, it does not make sense to iterate over the "time steps", because they are not meant to be played in order. Rather, each represents the vibration at a different "mode." Setting this to 1 changes the semantics of the reader to not report the time steps to downstream filters. By default, this is off, which is the case for most Exodus files.
- obj.HasModeShapesOn () - Some simulations overload the Exodus time steps to represent mode shapes. In this case, it does not make sense to iterate over the "time steps", because they are not meant to be played in order. Rather, each represents the vibration at a different "mode." Setting this to 1 changes the semantics of the reader to not report the time steps to downstream filters. By default, this is off, which is the case for most Exodus files.
- obj.HasModeShapesOff () - Some simulations overload the Exodus time steps to represent mode shapes. In this case, it does not make sense to iterate over the "time steps", because they are not meant to be played in order. Rather, each represents the vibration at a different "mode." Setting this to 1 changes the semantics of the reader to not report the time steps to downstream filters. By default, this is off, which is the case for most Exodus files.
- int = obj.GetDisplayType ()
- obj.SetDisplayType (int type)
- obj.ExodusModelMetadataOn ()
- obj.ExodusModelMetadataOff ()
- obj.SetExodusModelMetadata (int)
- int = obj.GetExodusModelMetadata ()
- vtkExodusModel = obj.GetExodusModel ()
- obj.SetPackExodusModelOntoOutput (int)
- int = obj.GetPackExodusModelOntoOutput ()
- obj.PackExodusModelOntoOutputOn ()
- obj.PackExodusModelOntoOutputOff ()

- int = obj.IsValidVariable (string type, string name)
- int = obj.GetVariableID (string type, string name)
- obj.SetAllAssemblyArrayStatus (int status)
- obj.SetAllBlockArrayStatus (int status)
- obj.SetAllCellArrayStatus (int status)
- obj.SetAllHierarchyArrayStatus (int status)
- obj.SetAllMaterialArrayStatus (int status)
- obj.SetAllPartArrayStatus (int status)
- obj.SetAllPointArrayStatus (int status)
- obj.SetArrayStatus (string type, string name, int flag)
- int = obj.GetArrayStatus (string type, string name)
- int = obj.GetTimeSeriesData (int ID, string vName, string vType, vtkFloatArray result)
- int = obj.GetNumberOfVariableArrays ()
- string = obj.GetVariableArrayName (int a_which)
- obj.EnableDSPFiltering ()
- obj.AddFilter (vtkDSPFilterDefinition a_filter)
- obj.StartAddingFilter ()
- obj.AddFilterInputVar (string name)
- obj.AddFilterOutputVar (string name)
- obj.AddFilterNumeratorWeight (double weight)
- obj.AddFilterForwardNumeratorWeight (double weight)
- obj.AddFilterDenominatorWeight (double weight)
- obj.FinishAddingFilter ()
- obj.RemoveFilter (string a_outputVariableName)
- obj.GetDSPOutputArrays (int exoid, vtkUnstructuredGrid output)

34.18 vtkFacetReader

34.18.1 Usage

vtkFacetReader creates a poly data dataset. It reads ASCII files stored in Facet format

The facet format looks like this: FACET FILE ... nparts Part 1 name 0 npoints 0 0 p1x p1y p1z p2x p2y p2z ... 1 Part 1 name ncells npointspercell p1c1 p2c1 p3c1 ... pnc1 materialnum partnum p1c2 p2c2 p3c2 ... pnc2 materialnum partnum ...

To create an instance of class vtkFacetReader, simply invoke its constructor as follows

```
obj = vtkFacetReader
```

34.18.2 Methods

The class vtkFacetReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFacetReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFacetReader = obj.NewInstance ()`
- `vtkFacetReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name of Facet datafile to read
- `string = obj.GetFileName ()` - Specify file name of Facet datafile to read

34.19 vtkGreedyTerrainDecimation

34.19.1 Usage

vtkGreedyTerrainDecimation approximates a height field with a triangle mesh (triangulated irregular network - TIN) using a greedy insertion algorithm similar to that described by Garland and Heckbert in their paper "Fast Polygonal Approximations of Terrain and Height Fields" (Technical Report CMU-CS-95-181). The input to the filter is a height field (represented by a image whose scalar values are height) and the output of the filter is polygonal data consisting of triangles. The number of triangles in the output is reduced in number as compared to a naive tessellation of the input height field. This filter copies point data from the input to the output for those points present in the output.

An brief description of the algorithm is as follows. The algorithm uses a top-down decimation approach that initially represents the height field with two triangles (whose vertices are at the four corners of the image). These two triangles form a Delaunay triangulation. In an iterative fashion, the point in the image with the greatest error (as compared to the original height field) is injected into the triangulation. (Note that the single point with the greatest error per triangle is identified and placed into a priority queue. As the triangulation is modified, the errors from the deleted triangles are removed from the queue, error values from the new triangles are added.) The point whose error is at the top of the queue is added to the triangulaion modifying it using the standard incremental Delaunay point insertion (see vtkDelaunay2D) algorithm. Points are repeatedly inserted until the appropriate (user-specified) error criterion is met.

To use this filter, set the input and specify the error measure to be used. The error measure options are 1) the absolute number of triangles to be produced; 2) a fractional reduction of the mesh (numTris/maxTris) where maxTris is the largest possible number of triangles 2*(dims[0]-1)*(dims[1]-1); 3) an absolute measure on error (maximum difference in height field to reduced TIN); and 4) relative error (the absolute error is normalized by the diagonal of the bounding box of the height field).

To create an instance of class vtkGreedyTerrainDecimation, simply invoke its constructor as follows

```
obj = vtkGreedyTerrainDecimation
```

34.19.2 Methods

The class vtkGreedyTerrainDecimation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGreedyTerrainDecimation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkGreedyTerrainDecimation = obj.NewInstance ()`
- `vtkGreedyTerrainDecimation = obj.SafeDownCast (vtkObject o)`
- `obj.SetErrorMeasure (int )` - Specify how to terminate the algorithm: either as an absolute number of triangles, a relative number of triangles (normalized by the full resolution mesh), an absolute error (in the height field), or relative error (normalized by the length of the diagonal of the image).
- `int = obj.GetErrorMeasureMinValue ()` - Specify how to terminate the algorithm: either as an absolute number of triangles, a relative number of triangles (normalized by the full resolution mesh), an absolute error (in the height field), or relative error (normalized by the length of the diagonal of the image).
- `int = obj.GetErrorMeasureMaxValue ()` - Specify how to terminate the algorithm: either as an absolute number of triangles, a relative number of triangles (normalized by the full resolution mesh), an absolute error (in the height field), or relative error (normalized by the length of the diagonal of the image).
- `int = obj.GetErrorMeasure ()` - Specify how to terminate the algorithm: either as an absolute number of triangles, a relative number of triangles (normalized by the full resolution mesh), an absolute error (in the height field), or relative error (normalized by the length of the diagonal of the image).
- `obj.SetErrorMeasureToNumberOfTriangles ()` - Specify how to terminate the algorithm: either as an absolute number of triangles, a relative number of triangles (normalized by the full resolution mesh), an absolute error (in the height field), or relative error (normalized by the length of the diagonal of the image).
- `obj.SetErrorMeasureToSpecifiedReduction ()` - Specify how to terminate the algorithm: either as an absolute number of triangles, a relative number of triangles (normalized by the full resolution mesh), an absolute error (in the height field), or relative error (normalized by the length of the diagonal of the image).
- `obj.SetErrorMeasureToAbsoluteError ()` - Specify how to terminate the algorithm: either as an absolute number of triangles, a relative number of triangles (normalized by the full resolution mesh), an absolute error (in the height field), or relative error (normalized by the length of the diagonal of the image).
- `obj.SetErrorMeasureToRelativeError ()` - Specify the number of triangles to produce on output. (It is a good idea to make sure this is less than a tessellated mesh at full resolution.) You need to set this value only when the error measure is set to NumberOfTriangles.
- `obj.SetNumberOfTriangles (vtkIdType )` - Specify the number of triangles to produce on output. (It is a good idea to make sure this is less than a tessellated mesh at full resolution.) You need to set this value only when the error measure is set to NumberOfTriangles.
- `vtkIdType = obj.GetNumberOfTrianglesMinValue ()` - Specify the number of triangles to produce on output. (It is a good idea to make sure this is less than a tessellated mesh at full resolution.) You need to set this value only when the error measure is set to NumberOfTriangles.
- `vtkIdType = obj.GetNumberOfTrianglesMaxValue ()` - Specify the number of triangles to produce on output. (It is a good idea to make sure this is less than a tessellated mesh at full resolution.) You need to set this value only when the error measure is set to NumberOfTriangles.
- `vtkIdType = obj.GetNumberOfTriangles ()` - Specify the number of triangles to produce on output. (It is a good idea to make sure this is less than a tessellated mesh at full resolution.) You need to set this value only when the error measure is set to NumberOfTriangles.
- `obj.SetReduction (double )` - Specify the reduction of the mesh (represented as a fraction). Note that a value of 0.10 means a 10only when the error measure is set to SpecifiedReduction.

- `double = obj.GetReductionMinValue ()` - Specify the reduction of the mesh (represented as a fraction). Note that a value of 0.10 means a 10only when the error measure is set to SpecifiedReduction.
- `double = obj.GetReductionMaxValue ()` - Specify the reduction of the mesh (represented as a fraction). Note that a value of 0.10 means a 10only when the error measure is set to SpecifiedReduction.
- `double = obj.GetReduction ()` - Specify the reduction of the mesh (represented as a fraction). Note that a value of 0.10 means a 10only when the error measure is set to SpecifiedReduction.
- `obj.SetAbsoluteError (double )` - Specify the absolute error of the mesh; that is, the error in height between the decimated mesh and the original height field. You need to set this value only when the error measure is set to AbsoluteError.
- `double = obj.GetAbsoluteErrorMinValue ()` - Specify the absolute error of the mesh; that is, the error in height between the decimated mesh and the original height field. You need to set this value only when the error measure is set to AbsoluteError.
- `double = obj.GetAbsoluteErrorMaxValue ()` - Specify the absolute error of the mesh; that is, the error in height between the decimated mesh and the original height field. You need to set this value only when the error measure is set to AbsoluteError.
- `double = obj.GetAbsoluteError ()` - Specify the absolute error of the mesh; that is, the error in height between the decimated mesh and the original height field. You need to set this value only when the error measure is set to AbsoluteError.
- `obj.SetRelativeError (double )` - Specify the relative error of the mesh; that is, the error in height between the decimated mesh and the original height field normalized by the diagonal of the image. You need to set this value only when the error measure is set to RelativeError.
- `double = obj.GetRelativeErrorMinValue ()` - Specify the relative error of the mesh; that is, the error in height between the decimated mesh and the original height field normalized by the diagonal of the image. You need to set this value only when the error measure is set to RelativeError.
- `double = obj.GetRelativeErrorMaxValue ()` - Specify the relative error of the mesh; that is, the error in height between the decimated mesh and the original height field normalized by the diagonal of the image. You need to set this value only when the error measure is set to RelativeError.
- `double = obj.GetRelativeError ()` - Specify the relative error of the mesh; that is, the error in height between the decimated mesh and the original height field normalized by the diagonal of the image. You need to set this value only when the error measure is set to RelativeError.
- `obj.SetBoundaryVertexDeletion (int )` - Turn on/off the deletion of vertices on the boundary of a mesh. This may limit the maximum reduction that may be achieved.
- `int = obj.GetBoundaryVertexDeletion ()` - Turn on/off the deletion of vertices on the boundary of a mesh. This may limit the maximum reduction that may be achieved.
- `obj.BoundaryVertexDeletionOn ()` - Turn on/off the deletion of vertices on the boundary of a mesh. This may limit the maximum reduction that may be achieved.
- `obj.BoundaryVertexDeletionOff ()` - Turn on/off the deletion of vertices on the boundary of a mesh. This may limit the maximum reduction that may be achieved.
- `obj.SetComputeNormals (int )` - Compute normals based on the input image. Off by default.
- `int = obj.GetComputeNormals ()` - Compute normals based on the input image. Off by default.
- `obj.ComputeNormalsOn ()` - Compute normals based on the input image. Off by default.
- `obj.ComputeNormalsOff ()` - Compute normals based on the input image. Off by default.

34.20 vtkGridTransform

34.20.1 Usage

vtkGridTransform describes a nonlinear warp transformation as a set of displacement vectors sampled along a uniform 3D grid.

To create an instance of class vtkGridTransform, simply invoke its constructor as follows

```
obj = vtkGridTransform
```

34.20.2 Methods

The class vtkGridTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGridTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGridTransform = obj.NewInstance ()`
- `vtkGridTransform = obj.SafeDownCast (vtkObject o)`
- `obj.SetDisplacementGrid (vtkImageData )` - Set/Get the grid transform (the grid transform must have three components for displacement in x, y, and z respectively). The vtkGridTransform class will never modify the data.
- `vtkImageData = obj.GetDisplacementGrid ()` - Set/Get the grid transform (the grid transform must have three components for displacement in x, y, and z respectively). The vtkGridTransform class will never modify the data.
- `obj.SetDisplacementScale (double )` - Set scale factor to be applied to the displacements. This is used primarily for grids which contain integer data types. Default: 1
- `double = obj.GetDisplacementScale ()` - Set scale factor to be applied to the displacements. This is used primarily for grids which contain integer data types. Default: 1
- `obj.SetDisplacementShift (double )` - Set a shift to be applied to the displacements. The shift is applied after the scale, i.e. x = scale*y + shift. Default: 0
- `double = obj.GetDisplacementShift ()` - Set a shift to be applied to the displacements. The shift is applied after the scale, i.e. x = scale*y + shift. Default: 0
- `obj.SetInterpolationMode (int mode)` - Set interpolation mode for sampling the grid. Higher-order interpolation allows you to use a sparser grid. Default: Linear.
- `int = obj.GetInterpolationMode ()` - Set interpolation mode for sampling the grid. Higher-order interpolation allows you to use a sparser grid. Default: Linear.
- `obj.SetInterpolationModeToNearestNeighbor ()` - Set interpolation mode for sampling the grid. Higher-order interpolation allows you to use a sparser grid. Default: Linear.
- `obj.SetInterpolationModeToLinear ()` - Set interpolation mode for sampling the grid. Higher-order interpolation allows you to use a sparser grid. Default: Linear.
- `obj.SetInterpolationModeToCubic ()` - Set interpolation mode for sampling the grid. Higher-order interpolation allows you to use a sparser grid. Default: Linear.

- `string = obj.GetInterpolationModeAsString ()` - Set interpolation mode for sampling the grid. Higher-order interpolation allows you to use a sparser grid. Default: Linear.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.
- `long = obj.GetMTime ()` - Get the MTime.

34.21 vtkImageDataLIC2D

34.21.1 Usage

GPU implementation of a Line Integral Convolution, a technique for imaging vector fields.

The input on port 0 is an vtkImageData with extents of a 2D image. It needs a vector field on point data. Port 1 is a special port for customized noise input. It is an optional port. If not present, noise is generated by the filter. Even if none-power-of-two texture are supported, giving a power-of-two image may result in faster execution on the GPU. If noise input is not specified, then the filter using vtkImageNoiseSource to generate a 128x128 noise texture. This filter only works on point vectors. One can use a vtkCellDataToPointData filter to convert cell vectors to point vectors.

.SECTION Required OpenGL Extensins GL_ARB_texture_non_power_of_two GL_VERSION_2_0 GL_ARB_texture_float GL_ARB_draw_buffers GL_EXT_framebuffer_object GL_ARB_pixel_buffer_object

To create an instance of class vtkImageDataLIC2D, simply invoke its constructor as follows

```
obj = vtkImageDataLIC2D
```

34.21.2 Methods

The class vtkImageDataLIC2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageDataLIC2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageDataLIC2D = obj.NewInstance ()`
- `vtkImageDataLIC2D = obj.SafeDownCast (vtkObject o)`
- `int = obj.SetContext (vtkRenderWindow context)` - Get/Set the context. Context must be a vtkOpenGLRenderWindow. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions. Return 0 upon failure and 1 upon success.
- `vtkRenderWindow = obj.GetContext ()` - Get/Set the context. Context must be a vtkOpenGLRenderWindow. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions. Return 0 upon failure and 1 upon success.
- `obj.SetSteps (int )` - Number of steps. Initial value is 20. class invariant: Steps¿0. In term of visual quality, the greater the better.
- `int = obj.GetSteps ()` - Number of steps. Initial value is 20. class invariant: Steps¿0. In term of visual quality, the greater the better.
- `obj.SetStepSize (double )` - Step size. Specify the step size as a unit of the cell length of the input vector field. Cell lenghth is the length of the diagonal of a cell. Initial value is 1.0. class invariant: StepSize¿0.0. In term of visual quality, the smaller the better. The type for the interface is double as VTK interface is double but GPU only supports float. This value will be converted to float in the execution of the algorithm.

- `double = obj.GetStepSizeMinValue ()` - Step size. Specify the step size as a unit of the cell length of the input vector field. Cell lenghth is the length of the diagonal of a cell. Initial value is 1.0. class invariant: StepSize¿0.0. In term of visual quality, the smaller the better. The type for the interface is double as VTK interface is double but GPU only supports float. This value will be converted to float in the execution of the algorithm.
- `double = obj.GetStepSizeMaxValue ()` - Step size. Specify the step size as a unit of the cell length of the input vector field. Cell lenghth is the length of the diagonal of a cell. Initial value is 1.0. class invariant: StepSize¿0.0. In term of visual quality, the smaller the better. The type for the interface is double as VTK interface is double but GPU only supports float. This value will be converted to float in the execution of the algorithm.
- `double = obj.GetStepSize ()` - Step size. Specify the step size as a unit of the cell length of the input vector field. Cell lenghth is the length of the diagonal of a cell. Initial value is 1.0. class invariant: StepSize¿0.0. In term of visual quality, the smaller the better. The type for the interface is double as VTK interface is double but GPU only supports float. This value will be converted to float in the execution of the algorithm.
- `obj.SetMagnification (int )` - The the magnification factor. Default is 1
- `int = obj.GetMagnificationMinValue ()` - The the magnification factor. Default is 1
- `int = obj.GetMagnificationMaxValue ()` - The the magnification factor. Default is 1
- `int = obj.GetMagnification ()` - The the magnification factor. Default is 1
- `int = obj.GetOpenGLExtensionsSupported ()` - Check if the required OpenGL extensions / GPU are supported.
- `int = obj.GetFBOSuccess ()` - Check if LIC runs properly.
- `int = obj.GetLICSuccess ()`
- `obj.TranslateInputExtent (int inExt, int inWholeExtent, int outExt)`

34.22 vtkImageDataLIC2DExtentTranslator

34.22.1 Usage

To create an instance of class vtkImageDataLIC2DExtentTranslator, simply invoke its constructor as follows

```
obj = vtkImageDataLIC2DExtentTranslator
```

34.22.2 Methods

The class vtkImageDataLIC2DExtentTranslator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageDataLIC2DExtentTranslator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageDataLIC2DExtentTranslator = obj.NewInstance ()`
- `vtkImageDataLIC2DExtentTranslator = obj.SafeDownCast (vtkObject o)`
- `obj.SetAlgorithm (vtkImageDataLIC2D )` - Set the vtkImageDataLIC2D algorithm for which this extent translator is being used.

- `vtkImageDataLIC2D = obj.GetAlgorithm ()` - Set the vtkImageDataLIC2D algorithm for which this extent translator is being used.
- `obj.SetInputExtentTranslator (vtkExtentTranslator )`
- `vtkExtentTranslator = obj.GetInputExtentTranslator ()`
- `obj.SetInputWholeExtent (int , int , int , int , int , int )`
- `obj.SetInputWholeExtent (int  a[6])`
- `int = obj. GetInputWholeExtent ()`
- `int = obj.PieceToExtentThreadSafe (int piece, int numPieces, int ghostLevel, int wholeExtent, int r`

34.23 vtkImageToPolyDataFilter

34.23.1 Usage

vtkImageToPolyDataFilter converts raster data (i.e., an image) into polygonal data (i.e., quads or n-sided polygons), with each polygon assigned a constant color. This is useful for writers that generate vector formats (i.e., CGM or PostScript). To use this filter, you specify how to quantize the color (or whether to use an image with a lookup table), and what style the output should be. The output is always polygons, but the choice is n x m quads (where n and m define the input image dimensions) "Pixelize" option; arbitrary polygons "Polygonalize" option; or variable number of quads of constant color generated along scan lines "RunLength" option.

The algorithm quantizes color in order to create coherent regions that the polygons can represent with good compression. By default, the input image is quantized to 256 colors using a 3-3-2 bits for red-green-blue. However, you can also supply a single component image and a lookup table, with the single component assumed to be an index into the table. (Note: a quantized image can be generated with the filter vtkImageQuantizeRGBToIndex.) The number of colors on output is equal to the number of colors in the input lookup table (or 256 if the built in linear ramp is used).

The output of the filter is polygons with a single color per polygon cell. If the output style is set to "Polygonalize", the polygons may have an large number of points (bounded by something like 2*(n+m)); and the polygon may not be convex which may cause rendering problems on some systems (use vtkTriangleFilter). Otherwise, each polygon will have four vertices. The output also contains scalar data defining RGB color in unsigned char form.

To create an instance of class vtkImageToPolyDataFilter, simply invoke its constructor as follows

```
obj = vtkImageToPolyDataFilter
```

34.23.2 Methods

The class vtkImageToPolyDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageToPolyDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageToPolyDataFilter = obj.NewInstance ()`
- `vtkImageToPolyDataFilter = obj.SafeDownCast (vtkObject o)`

- `obj.SetOutputStyle (int )` - Specify how to create the output. Pixelize means converting the image to quad polygons with a constant color per quad. Polygonalize means merging colors together into polygonal regions, and then smoothing the regions (if smoothing is turned on). RunLength means creating quad polygons that may encompass several pixels on a scan line. The default behavior is Polygonalize.

- `int = obj.GetOutputStyleMinValue ()` - Specify how to create the output. Pixelize means converting the image to quad polygons with a constant color per quad. Polygonalize means merging colors together into polygonal regions, and then smoothing the regions (if smoothing is turned on). RunLength means creating quad polygons that may encompass several pixels on a scan line. The default behavior is Polygonalize.

- `int = obj.GetOutputStyleMaxValue ()` - Specify how to create the output. Pixelize means converting the image to quad polygons with a constant color per quad. Polygonalize means merging colors together into polygonal regions, and then smoothing the regions (if smoothing is turned on). RunLength means creating quad polygons that may encompass several pixels on a scan line. The default behavior is Polygonalize.

- `int = obj.GetOutputStyle ()` - Specify how to create the output. Pixelize means converting the image to quad polygons with a constant color per quad. Polygonalize means merging colors together into polygonal regions, and then smoothing the regions (if smoothing is turned on). RunLength means creating quad polygons that may encompass several pixels on a scan line. The default behavior is Polygonalize.

- `obj.SetOutputStyleToPixelize ()` - Specify how to create the output. Pixelize means converting the image to quad polygons with a constant color per quad. Polygonalize means merging colors together into polygonal regions, and then smoothing the regions (if smoothing is turned on). RunLength means creating quad polygons that may encompass several pixels on a scan line. The default behavior is Polygonalize.

- `obj.SetOutputStyleToPolygonalize ()` - Specify how to create the output. Pixelize means converting the image to quad polygons with a constant color per quad. Polygonalize means merging colors together into polygonal regions, and then smoothing the regions (if smoothing is turned on). RunLength means creating quad polygons that may encompass several pixels on a scan line. The default behavior is Polygonalize.

- `obj.SetOutputStyleToRunLength ()` - Specify how to create the output. Pixelize means converting the image to quad polygons with a constant color per quad. Polygonalize means merging colors together into polygonal regions, and then smoothing the regions (if smoothing is turned on). RunLength means creating quad polygons that may encompass several pixels on a scan line. The default behavior is Polygonalize.

- `obj.SetColorMode (int )` - Specify how to quantize color.

- `int = obj.GetColorModeMinValue ()` - Specify how to quantize color.

- `int = obj.GetColorModeMaxValue ()` - Specify how to quantize color.

- `int = obj.GetColorMode ()` - Specify how to quantize color.

- `obj.SetColorModeToLUT ()` - Specify how to quantize color.

- `obj.SetColorModeToLinear256 ()` - Specify how to quantize color.

- `obj.SetLookupTable (vtkScalarsToColors )` - Set/Get the vtkLookupTable to use. The lookup table is used when the color mode is set to LUT and a single component scalar is input.

- `vtkScalarsToColors = obj.GetLookupTable ()` - Set/Get the vtkLookupTable to use. The lookup table is used when the color mode is set to LUT and a single component scalar is input.

- `obj.SetSmoothing (int )` - If the output style is set to polygonalize, then you can control whether to smooth boundaries.
- `int = obj.GetSmoothing ()` - If the output style is set to polygonalize, then you can control whether to smooth boundaries.
- `obj.SmoothingOn ()` - If the output style is set to polygonalize, then you can control whether to smooth boundaries.
- `obj.SmoothingOff ()` - If the output style is set to polygonalize, then you can control whether to smooth boundaries.
- `obj.SetNumberOfSmoothingIterations (int )` - Specify the number of smoothing iterations to smooth polygons. (Only in effect if output style is Polygonalize and smoothing is on.)
- `int = obj.GetNumberOfSmoothingIterationsMinValue ()` - Specify the number of smoothing iterations to smooth polygons. (Only in effect if output style is Polygonalize and smoothing is on.)
- `int = obj.GetNumberOfSmoothingIterationsMaxValue ()` - Specify the number of smoothing iterations to smooth polygons. (Only in effect if output style is Polygonalize and smoothing is on.)
- `int = obj.GetNumberOfSmoothingIterations ()` - Specify the number of smoothing iterations to smooth polygons. (Only in effect if output style is Polygonalize and smoothing is on.)
- `obj.SetDecimation (int )` - Turn on/off whether the final polygons should be decimated. whether to smooth boundaries.
- `int = obj.GetDecimation ()` - Turn on/off whether the final polygons should be decimated. whether to smooth boundaries.
- `obj.DecimationOn ()` - Turn on/off whether the final polygons should be decimated. whether to smooth boundaries.
- `obj.DecimationOff ()` - Turn on/off whether the final polygons should be decimated. whether to smooth boundaries.
- `obj.SetDecimationError (double )` - Specify the error to use for decimation (if decimation is on). The error is an absolute number–the image spacing and dimensions are used to create points so the error should be consistent with the image size.
- `double = obj.GetDecimationErrorMinValue ()` - Specify the error to use for decimation (if decimation is on). The error is an absolute number–the image spacing and dimensions are used to create points so the error should be consistent with the image size.
- `double = obj.GetDecimationErrorMaxValue ()` - Specify the error to use for decimation (if decimation is on). The error is an absolute number–the image spacing and dimensions are used to create points so the error should be consistent with the image size.
- `double = obj.GetDecimationError ()` - Specify the error to use for decimation (if decimation is on). The error is an absolute number–the image spacing and dimensions are used to create points so the error should be consistent with the image size.
- `obj.SetError (int )` - Specify the error value between two colors where the colors are considered the same. Only use this if the color mode uses the default 256 table.
- `int = obj.GetErrorMinValue ()` - Specify the error value between two colors where the colors are considered the same. Only use this if the color mode uses the default 256 table.
- `int = obj.GetErrorMaxValue ()` - Specify the error value between two colors where the colors are considered the same. Only use this if the color mode uses the default 256 table.

- `int = obj.GetError ()` - Specify the error value between two colors where the colors are considered the same. Only use this if the color mode uses the default 256 table.

- `obj.SetSubImageSize (int )` - Specify the size (n by n pixels) of the largest region to polygonalize. When the OutputStyle is set to VTK_STYLE_POLYGONALIZE, large amounts of memory are used. In order to process large images, the image is broken into pieces that are at most Size pixels in width and height.

- `int = obj.GetSubImageSizeMinValue ()` - Specify the size (n by n pixels) of the largest region to polygonalize. When the OutputStyle is set to VTK_STYLE_POLYGONALIZE, large amounts of memory are used. In order to process large images, the image is broken into pieces that are at most Size pixels in width and height.

- `int = obj.GetSubImageSizeMaxValue ()` - Specify the size (n by n pixels) of the largest region to polygonalize. When the OutputStyle is set to VTK_STYLE_POLYGONALIZE, large amounts of memory are used. In order to process large images, the image is broken into pieces that are at most Size pixels in width and height.

- `int = obj.GetSubImageSize ()` - Specify the size (n by n pixels) of the largest region to polygonalize. When the OutputStyle is set to VTK_STYLE_POLYGONALIZE, large amounts of memory are used. In order to process large images, the image is broken into pieces that are at most Size pixels in width and height.

34.24 vtkImplicitModeller

34.24.1 Usage

vtkImplicitModeller is a filter that computes the distance from the input geometry to the points of an output structured point set. This distance function can then be "contoured" to generate new, offset surfaces from the original geometry. An important feature of this object is "capping". If capping is turned on, after the implicit model is created, the values on the boundary of the structured points dataset are set to the cap value. This is used to force closure of the resulting contoured surface. Note, however, that large cap values can generate weird surface normals in those cells adjacent to the boundary of the dataset. Using smaller cap value will reduce this effect. ¡P¿ Another important ivar is MaximumDistance. This controls how far into the volume the distance function is computed from the input geometry. Small values give significant increases in performance. However, there can strange sampling effects at the extreme range of the MaximumDistance. ¡P¿ In order to properly execute and sample the input data, a rectangular region in space must be defined (this is the ivar ModelBounds). If not explicitly defined, the model bounds will be computed. Note that to avoid boundary effects, it is possible to adjust the model bounds (i.e., using the AdjustBounds and AdjustDistance ivars) to strictly contain the sampled data. ¡P¿ This filter has one other unusual capability: it is possible to append data in a sequence of operations to generate a single output. This is useful when you have multiple datasets and want to create a conglomeration of all the data. However, the user must be careful to either specify the ModelBounds or specify the first item such that its bounds completely contain all other items. This is because the rectangular region of the output can not be changed after the 1st Append. ¡P¿ The ProcessMode ivar controls the method used within the Append function (where the actual work is done regardless if the Append function is explicitly called) to compute the implicit model. If set to work in voxel mode, each voxel is visited once. If set to cell mode, each cell is visited once. Tests have shown once per voxel to be faster when there are a lot of cells (at least a thousand?); relative performance improvement increases with addition cells. Primitives should not be stripped for best performance of the voxel mode. Also, if explicitly using the Append feature many times, the cell mode will probably be better because each voxel will be visited each Append. Append the data before input if possible when using the voxel mode. Do not switch between voxel and cell mode between execution of StartAppend and EndAppend. ¡P¿ Further performance improvement is now possible using the PerVoxel process mode on multi-processor machines (the mode is now multithreaded). Each thread processes a different "slab" of the output. Also, if the input is vtkPolyData, it is appropriately clipped for each thread; that is, each thread

only considers the input which could affect its slab of the output. ¡P¿ This filter can now produce output of any type supported by vtkImageData. However to support this change, additional sqrts must be executed during the Append step. Previously, the output was initialized to the squared CapValue in StartAppend, the output was updated with squared distance values during the Append, and then the sqrt of the distances was computed in EndAppend. To support different scalar types in the output (largely to reduce memory requirements as an vtkImageShiftScale and/or vtkImageCast could have achieved the same result), we can't "afford" to save squared value in the output, because then we could only represent up to the sqrt of the scalar max for an integer type in the output; 1 (instead of 255) for an unsigned char; 11 for a char (instead of 127). Thus this change may result in a minor performance degradation. Non-float output types can be scaled to the CapValue by turning ScaleToMaximumDistance On.

To create an instance of class vtkImplicitModeller, simply invoke its constructor as follows

```
obj = vtkImplicitModeller
```

34.24.2 Methods

The class vtkImplicitModeller has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitModeller class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitModeller = obj.NewInstance ()`
- `vtkImplicitModeller = obj.SafeDownCast (vtkObject o)`
- `double = obj.ComputeModelBounds (vtkDataSet inputNULL)` - Compute ModelBounds from input geometry. If input is not specified, the input of the filter will be used.
- `int = obj. GetSampleDimensions ()` - Set/Get the i-j-k dimensions on which to sample distance function.
- `obj.SetSampleDimensions (int i, int j, int k)` - Set/Get the i-j-k dimensions on which to sample distance function.
- `obj.SetSampleDimensions (int dim[3])` - Set/Get the i-j-k dimensions on which to sample distance function.
- `obj.SetMaximumDistance (double )` - Set / get the distance away from surface of input geometry to sample. Smaller values make large increases in performance.
- `double = obj.GetMaximumDistanceMinValue ()` - Set / get the distance away from surface of input geometry to sample. Smaller values make large increases in performance.
- `double = obj.GetMaximumDistanceMaxValue ()` - Set / get the distance away from surface of input geometry to sample. Smaller values make large increases in performance.
- `double = obj.GetMaximumDistance ()` - Set / get the distance away from surface of input geometry to sample. Smaller values make large increases in performance.
- `obj.SetModelBounds (double , double , double , double , double , double )` - Set / get the region in space in which to perform the sampling. If not specified, it will be computed automatically.
- `obj.SetModelBounds (double a[6])` - Set / get the region in space in which to perform the sampling. If not specified, it will be computed automatically.

- `double = obj. GetModelBounds ()` - Set / get the region in space in which to perform the sampling. If not specified, it will be computed automatically.
- `obj.SetAdjustBounds (int )` - Control how the model bounds are computed. If the ivar AdjustBounds is set, then the bounds specified (or computed automatically) is modified by the fraction given by AdjustDistance. This means that the model bounds is expanded in each of the x-y-z directions.
- `int = obj.GetAdjustBounds ()` - Control how the model bounds are computed. If the ivar AdjustBounds is set, then the bounds specified (or computed automatically) is modified by the fraction given by AdjustDistance. This means that the model bounds is expanded in each of the x-y-z directions.
- `obj.AdjustBoundsOn ()` - Control how the model bounds are computed. If the ivar AdjustBounds is set, then the bounds specified (or computed automatically) is modified by the fraction given by AdjustDistance. This means that the model bounds is expanded in each of the x-y-z directions.
- `obj.AdjustBoundsOff ()` - Control how the model bounds are computed. If the ivar AdjustBounds is set, then the bounds specified (or computed automatically) is modified by the fraction given by AdjustDistance. This means that the model bounds is expanded in each of the x-y-z directions.
- `obj.SetAdjustDistance (double )` - Specify the amount to grow the model bounds (if the ivar AdjustBounds is set). The value is a fraction of the maximum length of the sides of the box specified by the model bounds.
- `double = obj.GetAdjustDistanceMinValue ()` - Specify the amount to grow the model bounds (if the ivar AdjustBounds is set). The value is a fraction of the maximum length of the sides of the box specified by the model bounds.
- `double = obj.GetAdjustDistanceMaxValue ()` - Specify the amount to grow the model bounds (if the ivar AdjustBounds is set). The value is a fraction of the maximum length of the sides of the box specified by the model bounds.
- `double = obj.GetAdjustDistance ()` - Specify the amount to grow the model bounds (if the ivar AdjustBounds is set). The value is a fraction of the maximum length of the sides of the box specified by the model bounds.
- `obj.SetCapping (int )` - The outer boundary of the structured point set can be assigned a particular value. This can be used to close or "cap" all surfaces.
- `int = obj.GetCapping ()` - The outer boundary of the structured point set can be assigned a particular value. This can be used to close or "cap" all surfaces.
- `obj.CappingOn ()` - The outer boundary of the structured point set can be assigned a particular value. This can be used to close or "cap" all surfaces.
- `obj.CappingOff ()` - The outer boundary of the structured point set can be assigned a particular value. This can be used to close or "cap" all surfaces.
- `obj.SetCapValue (double value)` - Specify the capping value to use. The CapValue is also used as an initial distance value at each point in the dataset.
- `double = obj.GetCapValue ()` - Specify the capping value to use. The CapValue is also used as an initial distance value at each point in the dataset.
- `obj.SetScaleToMaximumDistance (int )` - If a non-floating output type is specified, the output distances can be scaled to use the entire positive scalar range of the output type specified (up to the CapValue which is equal to the max for the type unless modified by the user). For example, if ScaleToMaximumDistance is On and the OutputScalarType is UnsignedChar the distances saved in the output would be linearly scaled between 0 (for distances "very close" to the surface) and 255 (at the specifed maximum distance)... assuming the CapValue is not changed from 255.

- `int = obj.GetScaleToMaximumDistance ()` - If a non-floating output type is specified, the output distances can be scaled to use the entire positive scalar range of the output type specified (up to the CapValue which is equal to the max for the type unless modified by the user). For example, if ScaleToMaximumDistance is On and the OutputScalarType is UnsignedChar the distances saved in the output would be linearly scaled between 0 (for distances "very close" to the surface) and 255 (at the specifed maximum distance)... assuming the CapValue is not changed from 255.

- `obj.ScaleToMaximumDistanceOn ()` - If a non-floating output type is specified, the output distances can be scaled to use the entire positive scalar range of the output type specified (up to the CapValue which is equal to the max for the type unless modified by the user). For example, if ScaleToMaximumDistance is On and the OutputScalarType is UnsignedChar the distances saved in the output would be linearly scaled between 0 (for distances "very close" to the surface) and 255 (at the specifed maximum distance)... assuming the CapValue is not changed from 255.

- `obj.ScaleToMaximumDistanceOff ()` - If a non-floating output type is specified, the output distances can be scaled to use the entire positive scalar range of the output type specified (up to the CapValue which is equal to the max for the type unless modified by the user). For example, if ScaleToMaximumDistance is On and the OutputScalarType is UnsignedChar the distances saved in the output would be linearly scaled between 0 (for distances "very close" to the surface) and 255 (at the specifed maximum distance)... assuming the CapValue is not changed from 255.

- `obj.SetProcessMode (int )` - Specify whether to visit each cell once per append or each voxel once per append. Some tests have shown once per voxel to be faster when there are a lot of cells (at least a thousand?); relative performance improvement increases with addition cells. Primitives should not be stripped for best performance of the voxel mode.

- `int = obj.GetProcessModeMinValue ()` - Specify whether to visit each cell once per append or each voxel once per append. Some tests have shown once per voxel to be faster when there are a lot of cells (at least a thousand?); relative performance improvement increases with addition cells. Primitives should not be stripped for best performance of the voxel mode.

- `int = obj.GetProcessModeMaxValue ()` - Specify whether to visit each cell once per append or each voxel once per append. Some tests have shown once per voxel to be faster when there are a lot of cells (at least a thousand?); relative performance improvement increases with addition cells. Primitives should not be stripped for best performance of the voxel mode.

- `int = obj.GetProcessMode ()` - Specify whether to visit each cell once per append or each voxel once per append. Some tests have shown once per voxel to be faster when there are a lot of cells (at least a thousand?); relative performance improvement increases with addition cells. Primitives should not be stripped for best performance of the voxel mode.

- `obj.SetProcessModeToPerVoxel ()` - Specify whether to visit each cell once per append or each voxel once per append. Some tests have shown once per voxel to be faster when there are a lot of cells (at least a thousand?); relative performance improvement increases with addition cells. Primitives should not be stripped for best performance of the voxel mode.

- `obj.SetProcessModeToPerCell ()` - Specify whether to visit each cell once per append or each voxel once per append. Some tests have shown once per voxel to be faster when there are a lot of cells (at least a thousand?); relative performance improvement increases with addition cells. Primitives should not be stripped for best performance of the voxel mode.

- `string = obj.GetProcessModeAsString (void )` - Specify whether to visit each cell once per append or each voxel once per append. Some tests have shown once per voxel to be faster when there are a lot of cells (at least a thousand?); relative performance improvement increases with addition cells. Primitives should not be stripped for best performance of the voxel mode.

- `obj.SetLocatorMaxLevel (int )` - Specify the level of the locator to use when using the per voxel process mode.

- `int = obj.GetLocatorMaxLevel ()` - Specify the level of the locator to use when using the per voxel process mode.
- `obj.SetNumberOfThreads (int )` - Set / Get the number of threads used during Per-Voxel processing mode
- `int = obj.GetNumberOfThreadsMinValue ()` - Set / Get the number of threads used during Per-Voxel processing mode
- `int = obj.GetNumberOfThreadsMaxValue ()` - Set / Get the number of threads used during Per-Voxel processing mode
- `int = obj.GetNumberOfThreads ()` - Set / Get the number of threads used during Per-Voxel processing mode
- `obj.SetOutputScalarType (int type)` - Set the desired output scalar type.
- `int = obj.GetOutputScalarType ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToFloat ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToDouble ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToInt ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToUnsignedInt ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToLong ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToUnsignedLong ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToShort ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToUnsignedShort ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToUnsignedChar ()` - Set the desired output scalar type.
- `obj.SetOutputScalarTypeToChar ()` - Set the desired output scalar type.
- `obj.StartAppend ()` - Initialize the filter for appending data. You must invoke the StartAppend() method before doing successive Appends(). It's also a good idea to manually specify the model bounds; otherwise the input bounds for the data will be used.
- `obj.Append (vtkDataSet input)` - Append a data set to the existing output. To use this function, you'll have to invoke the StartAppend() method before doing successive appends. It's also a good idea to specify the model bounds; otherwise the input model bounds is used. When you've finished appending, use the EndAppend() method.
- `obj.EndAppend ()` - Method completes the append process.

34.25 vtkIterativeClosestPointTransform

34.25.1 Usage

Match two surfaces using the iterative closest point (ICP) algorithm. The core of the algorithm is to match each vertex in one surface with the closest surface point on the other, then apply the transformation that modify one surface to best match the other (in a least square sense). This has to be iterated to get proper convergence of the surfaces. .SECTION Note Use vtkTransformPolyDataFilter to apply the resulting ICP transform to your data. You might also set it to your actor's user transform. .SECTION Note This class makes use of vtkLandmarkTransform internally to compute the best fit. Use the GetLandmarkTransform

member to get a pointer to that transform and set its parameters. You might, for example, constrain the number of degrees of freedom of the solution (i.e. rigid body, similarity, etc.) by checking the vtkLandmarkTransform documentation for its SetMode member.

To create an instance of class vtkIterativeClosestPointTransform, simply invoke its constructor as follows

```
obj = vtkIterativeClosestPointTransform
```

34.25.2 Methods

The class vtkIterativeClosestPointTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIterativeClosestPointTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIterativeClosestPointTransform = obj.NewInstance ()`
- `vtkIterativeClosestPointTransform = obj.SafeDownCast (vtkObject o)`
- `obj.SetSource (vtkDataSet source)` - Specify the source and target data sets.
- `obj.SetTarget (vtkDataSet target)` - Specify the source and target data sets.
- `vtkDataSet = obj.GetSource ()` - Specify the source and target data sets.
- `vtkDataSet = obj.GetTarget ()` - Specify the source and target data sets.
- `obj.SetLocator (vtkCellLocator locator)` - Set/Get a spatial locator for speeding up the search process. An instance of vtkCellLocator is used by default.
- `vtkCellLocator = obj.GetLocator ()` - Set/Get a spatial locator for speeding up the search process. An instance of vtkCellLocator is used by default.
- `obj.SetMaximumNumberOfIterations (int )` - Set/Get the maximum number of iterations. Default is 50.
- `int = obj.GetMaximumNumberOfIterations ()` - Set/Get the maximum number of iterations. Default is 50.
- `int = obj.GetNumberOfIterations ()` - Get the number of iterations since the last update
- `obj.SetCheckMeanDistance (int )` - Force the algorithm to check the mean distance between two iterations. Default is Off.
- `int = obj.GetCheckMeanDistance ()` - Force the algorithm to check the mean distance between two iterations. Default is Off.
- `obj.CheckMeanDistanceOn ()` - Force the algorithm to check the mean distance between two iterations. Default is Off.
- `obj.CheckMeanDistanceOff ()` - Force the algorithm to check the mean distance between two iterations. Default is Off.
- `obj.SetMeanDistanceMode (int )` - Specify the mean distance mode. This mode expresses how the mean distance is computed. The RMS mode is the square root of the average of the sum of squares of the closest point distances. The Absolute Value mode is the mean of the sum of absolute values of the closest point distances. The default is VTK_ICP_MODE_RMS

- `int = obj.GetMeanDistanceModeMinValue ()` - Specify the mean distance mode. This mode expresses how the mean distance is computed. The RMS mode is the square root of the average of the sum of squares of the closest point distances. The Absolute Value mode is the mean of the sum of absolute values of the closest point distances. The default is VTK_ICP_MODE_RMS
- `int = obj.GetMeanDistanceModeMaxValue ()` - Specify the mean distance mode. This mode expresses how the mean distance is computed. The RMS mode is the square root of the average of the sum of squares of the closest point distances. The Absolute Value mode is the mean of the sum of absolute values of the closest point distances. The default is VTK_ICP_MODE_RMS
- `int = obj.GetMeanDistanceMode ()` - Specify the mean distance mode. This mode expresses how the mean distance is computed. The RMS mode is the square root of the average of the sum of squares of the closest point distances. The Absolute Value mode is the mean of the sum of absolute values of the closest point distances. The default is VTK_ICP_MODE_RMS
- `obj.SetMeanDistanceModeToRMS ()` - Specify the mean distance mode. This mode expresses how the mean distance is computed. The RMS mode is the square root of the average of the sum of squares of the closest point distances. The Absolute Value mode is the mean of the sum of absolute values of the closest point distances. The default is VTK_ICP_MODE_RMS
- `obj.SetMeanDistanceModeToAbsoluteValue ()` - Specify the mean distance mode. This mode expresses how the mean distance is computed. The RMS mode is the square root of the average of the sum of squares of the closest point distances. The Absolute Value mode is the mean of the sum of absolute values of the closest point distances. The default is VTK_ICP_MODE_RMS
- `string = obj.GetMeanDistanceModeAsString ()` - Specify the mean distance mode. This mode expresses how the mean distance is computed. The RMS mode is the square root of the average of the sum of squares of the closest point distances. The Absolute Value mode is the mean of the sum of absolute values of the closest point distances. The default is VTK_ICP_MODE_RMS
- `obj.SetMaximumMeanDistance (double )` - Set/Get the maximum mean distance between two iteration. If the mean distance is lower than this, the convergence stops. The default is 0.01.
- `double = obj.GetMaximumMeanDistance ()` - Set/Get the maximum mean distance between two iteration. If the mean distance is lower than this, the convergence stops. The default is 0.01.
- `double = obj.GetMeanDistance ()` - Get the mean distance between the last two iterations.
- `obj.SetMaximumNumberOfLandmarks (int )` - Set/Get the maximum number of landmarks sampled in your dataset. If your dataset is dense, then you will typically not need all the points to compute the ICP transform. The default is 200.
- `int = obj.GetMaximumNumberOfLandmarks ()` - Set/Get the maximum number of landmarks sampled in your dataset. If your dataset is dense, then you will typically not need all the points to compute the ICP transform. The default is 200.
- `obj.SetStartByMatchingCentroids (int )` - Starts the process by translating source centroid to target centroid. The default is Off.
- `int = obj.GetStartByMatchingCentroids ()` - Starts the process by translating source centroid to target centroid. The default is Off.
- `obj.StartByMatchingCentroidsOn ()` - Starts the process by translating source centroid to target centroid. The default is Off.
- `obj.StartByMatchingCentroidsOff ()` - Starts the process by translating source centroid to target centroid. The default is Off.
- `vtkLandmarkTransform = obj.GetLandmarkTransform ()` - Get the internal landmark transform. Use it to constrain the number of degrees of freedom of the solution (i.e. rigid body, similarity, etc.).

- `obj.Inverse ()` - Invert the transformation. This is done by switching the source and target.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.

34.26 vtkLandmarkTransform

34.26.1 Usage

A vtkLandmarkTransform is defined by two sets of landmarks, the transform computed gives the best fit mapping one onto the other, in a least squares sense. The indices are taken to correspond, so point 1 in the first set will get mapped close to point 1 in the second set, etc. Call SetSourceLandmarks and SetTargetLandmarks to specify the two sets of landmarks, ensure they have the same number of points.

To create an instance of class vtkLandmarkTransform, simply invoke its constructor as follows

```
obj = vtkLandmarkTransform
```

34.26.2 Methods

The class vtkLandmarkTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLandmarkTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLandmarkTransform = obj.NewInstance ()`
- `vtkLandmarkTransform = obj.SafeDownCast (vtkObject o)`
- `obj.SetSourceLandmarks (vtkPoints points)` - Specify the source and target landmark sets. The two sets must have the same number of points. If you add or change points in these objects, you must call Modified() on them or the transformation might not update.
- `obj.SetTargetLandmarks (vtkPoints points)` - Specify the source and target landmark sets. The two sets must have the same number of points. If you add or change points in these objects, you must call Modified() on them or the transformation might not update.
- `vtkPoints = obj.GetSourceLandmarks ()` - Specify the source and target landmark sets. The two sets must have the same number of points. If you add or change points in these objects, you must call Modified() on them or the transformation might not update.
- `vtkPoints = obj.GetTargetLandmarks ()` - Specify the source and target landmark sets. The two sets must have the same number of points. If you add or change points in these objects, you must call Modified() on them or the transformation might not update.
- `obj.SetMode (int )` - Set the number of degrees of freedom to constrain the solution to. Rigidbody (VTK_LANDMARK_RIGIDBODY): rotation and translation only. Similarity (VTK_LANDMARK_SIMILARITY): rotation, translation and isotropic scaling. Affine (VTK_LANDMARK_AFFINE): collinearity is preserved. Ratios of distances along a line are preserved. The default is similarity.
- `obj.SetModeToRigidBody ()` - Set the number of degrees of freedom to constrain the solution to. Rigidbody (VTK_LANDMARK_RIGIDBODY): rotation and translation only. Similarity (VTK_LANDMARK_SIMILA rotation, translation and isotropic scaling. Affine (VTK_LANDMARK_AFFINE): collinearity is preserved. Ratios of distances along a line are preserved. The default is similarity.

- `obj.SetModeToSimilarity ()` - Set the number of degrees of freedom to constrain the solution to. Rigidbody (VTK_LANDMARK_RIGIDBODY): rotation and translation only. Similarity (VTK_LANDMARK_SIMILA rotation, translation and isotropic scaling. Affine (VTK_LANDMARK_AFFINE): collinearity is preserved. Ratios of distances along a line are preserved. The default is similarity.

- `obj.SetModeToAffine ()` - Set the number of degrees of freedom to constrain the solution to. Rigidbody (VTK_LANDMARK_RIGIDBODY): rotation and translation only. Similarity (VTK_LANDMARK_SIMILARITY rotation, translation and isotropic scaling. Affine (VTK_LANDMARK_AFFINE): collinearity is preserved. Ratios of distances along a line are preserved. The default is similarity.

- `int = obj.GetMode ()` - Get the current transformation mode.

- `string = obj.GetModeAsString ()` - Get the current transformation mode.

- `obj.Inverse ()` - Invert the transformation. This is done by switching the source and target landmarks.

- `long = obj.GetMTime ()` - Get the MTime.

- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.

34.27 vtkLegendBoxActor

34.27.1 Usage

vtkLegendBoxActor is used to associate a symbol with a text string. The user specifies a vtkPolyData to use as the symbol, and a string associated with the symbol. The actor can then be placed in the scene in the same way that any other vtkActor2D can be used.

To use this class, you must define the position of the legend box by using the superclasses' vtkActor2D::Position coordinate and Position2 coordinate. Then define the set of symbols and text strings that make up the menu box. The font attributes of the entries can be set through the vtkTextProperty associated to this actor. The class will scale the symbols and text to fit in the legend box defined by (Position,Position2). Optional features like turning on a border line and setting the spacing between the border and the symbols/text can also be set.

To create an instance of class vtkLegendBoxActor, simply invoke its constructor as follows

```
obj = vtkLegendBoxActor
```

34.27.2 Methods

The class vtkLegendBoxActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLegendBoxActor class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkLegendBoxActor = obj.NewInstance ()`

- `vtkLegendBoxActor = obj.SafeDownCast (vtkObject o)`

- `obj.SetNumberOfEntries (int num)` - Specify the number of entries in the legend box.

- `int = obj.GetNumberOfEntries ()` - Add an entry to the legend box. You must supply a vtkPolyData to be used as a symbol (it can be NULL) and a text string (which also can be NULL). The vtkPolyData is assumed to be defined in the x-y plane, and the text is assumed to be a single line in height. Note that when this method is invoked previous entries are deleted. Also supply a text string and optionally a color. (If a color is not specified, then the entry color is the same as this actor's color.) (Note: use the set methods when you use SetNumberOfEntries().)

- `obj.SetEntry (int i, vtkPolyData symbol, string string, double color[3])` - Add an entry to the legend box. You must supply a vtkPolyData to be used as a symbol (it can be NULL) and a text string (which also can be NULL). The vtkPolyData is assumed to be defined in the x-y plane, and the text is assumed to be a single line in height. Note that when this method is invoked previous entries are deleted. Also supply a text string and optionally a color. (If a color is not specified, then the entry color is the same as this actor's color.) (Note: use the set methods when you use SetNumberOfEntries().)

- `obj.SetEntrySymbol (int i, vtkPolyData symbol)` - Add an entry to the legend box. You must supply a vtkPolyData to be used as a symbol (it can be NULL) and a text string (which also can be NULL). The vtkPolyData is assumed to be defined in the x-y plane, and the text is assumed to be a single line in height. Note that when this method is invoked previous entries are deleted. Also supply a text string and optionally a color. (If a color is not specified, then the entry color is the same as this actor's color.) (Note: use the set methods when you use SetNumberOfEntries().)

- `obj.SetEntryString (int i, string string)` - Add an entry to the legend box. You must supply a vtkPolyData to be used as a symbol (it can be NULL) and a text string (which also can be NULL). The vtkPolyData is assumed to be defined in the x-y plane, and the text is assumed to be a single line in height. Note that when this method is invoked previous entries are deleted. Also supply a text string and optionally a color. (If a color is not specified, then the entry color is the same as this actor's color.) (Note: use the set methods when you use SetNumberOfEntries().)

- `obj.SetEntryColor (int i, double color[3])` - Add an entry to the legend box. You must supply a vtkPolyData to be used as a symbol (it can be NULL) and a text string (which also can be NULL). The vtkPolyData is assumed to be defined in the x-y plane, and the text is assumed to be a single line in height. Note that when this method is invoked previous entries are deleted. Also supply a text string and optionally a color. (If a color is not specified, then the entry color is the same as this actor's color.) (Note: use the set methods when you use SetNumberOfEntries().)

- `obj.SetEntryColor (int i, double r, double g, double b)` - Add an entry to the legend box. You must supply a vtkPolyData to be used as a symbol (it can be NULL) and a text string (which also can be NULL). The vtkPolyData is assumed to be defined in the x-y plane, and the text is assumed to be a single line in height. Note that when this method is invoked previous entries are deleted. Also supply a text string and optionally a color. (If a color is not specified, then the entry color is the same as this actor's color.) (Note: use the set methods when you use SetNumberOfEntries().)

- `vtkPolyData = obj.GetEntrySymbol (int i)` - Add an entry to the legend box. You must supply a vtkPolyData to be used as a symbol (it can be NULL) and a text string (which also can be NULL). The vtkPolyData is assumed to be defined in the x-y plane, and the text is assumed to be a single line in height. Note that when this method is invoked previous entries are deleted. Also supply a text string and optionally a color. (If a color is not specified, then the entry color is the same as this actor's color.) (Note: use the set methods when you use SetNumberOfEntries().)

- `string = obj.GetEntryString (int i)` - Add an entry to the legend box. You must supply a vtkPolyData to be used as a symbol (it can be NULL) and a text string (which also can be NULL). The vtkPolyData is assumed to be defined in the x-y plane, and the text is assumed to be a single line in height. Note that when this method is invoked previous entries are deleted. Also supply a text string and optionally a color. (If a color is not specified, then the entry color is the same as this actor's color.) (Note: use the set methods when you use SetNumberOfEntries().)

- `double = obj.GetEntryColor (int i)` - Add an entry to the legend box. You must supply a vtkPolyData to be used as a symbol (it can be NULL) and a text string (which also can be NULL). The vtkPolyData is assumed to be defined in the x-y plane, and the text is assumed to be a single line in height. Note that when this method is invoked previous entries are deleted. Also supply a text string and optionally a color. (If a color is not specified, then the entry color is the same as this actor's color.) (Note: use the set methods when you use SetNumberOfEntries().)
- `obj.SetEntryTextProperty (vtkTextProperty p)` - Set/Get the text property.
- `vtkTextProperty = obj.GetEntryTextProperty ()` - Set/Get the text property.
- `obj.SetBorder (int )` - Set/Get the flag that controls whether a border will be drawn around the legend box.
- `int = obj.GetBorder ()` - Set/Get the flag that controls whether a border will be drawn around the legend box.
- `obj.BorderOn ()` - Set/Get the flag that controls whether a border will be drawn around the legend box.
- `obj.BorderOff ()` - Set/Get the flag that controls whether a border will be drawn around the legend box.
- `obj.SetLockBorder (int )` - Set/Get the flag that controls whether the border and legend placement is locked into the rectangle defined by (Position,Position2). If off, then the legend box will adjust its size so that the border fits nicely around the text and symbols. (The ivar is off by default.) Note: the legend box is guaranteed to lie within the original border definition.
- `int = obj.GetLockBorder ()` - Set/Get the flag that controls whether the border and legend placement is locked into the rectangle defined by (Position,Position2). If off, then the legend box will adjust its size so that the border fits nicely around the text and symbols. (The ivar is off by default.) Note: the legend box is guaranteed to lie within the original border definition.
- `obj.LockBorderOn ()` - Set/Get the flag that controls whether the border and legend placement is locked into the rectangle defined by (Position,Position2). If off, then the legend box will adjust its size so that the border fits nicely around the text and symbols. (The ivar is off by default.) Note: the legend box is guaranteed to lie within the original border definition.
- `obj.LockBorderOff ()` - Set/Get the flag that controls whether the border and legend placement is locked into the rectangle defined by (Position,Position2). If off, then the legend box will adjust its size so that the border fits nicely around the text and symbols. (The ivar is off by default.) Note: the legend box is guaranteed to lie within the original border definition.
- `obj.SetBox (int )` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the legend box.
- `int = obj.GetBox ()` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the legend box.
- `obj.BoxOn ()` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the legend box.
- `obj.BoxOff ()` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the legend box.
- `vtkProperty2D = obj.GetBoxProperty ()` - Get the box vtkProperty2D.
- `obj.SetPadding (int )` - Set/Get the padding between the legend entries and the border. The value is specified in pixels.

- `int = obj.GetPaddingMinValue ()` - Set/Get the padding between the legend entries and the border. The value is specified in pixels.
- `int = obj.GetPaddingMaxValue ()` - Set/Get the padding between the legend entries and the border. The value is specified in pixels.
- `int = obj.GetPadding ()` - Set/Get the padding between the legend entries and the border. The value is specified in pixels.
- `obj.SetScalarVisibility (int )` - Turn on/off flag to control whether the symbol's scalar data is used to color the symbol. If off, the color of the vtkLegendBoxActor is used.
- `int = obj.GetScalarVisibility ()` - Turn on/off flag to control whether the symbol's scalar data is used to color the symbol. If off, the color of the vtkLegendBoxActor is used.
- `obj.ScalarVisibilityOn ()` - Turn on/off flag to control whether the symbol's scalar data is used to color the symbol. If off, the color of the vtkLegendBoxActor is used.
- `obj.ScalarVisibilityOff ()` - Turn on/off flag to control whether the symbol's scalar data is used to color the symbol. If off, the color of the vtkLegendBoxActor is used.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this scaled text actor. Overloads the virtual vtkProp method.

34.28 vtkLegendScaleActor

34.28.1 Usage

This class is used to annotate the render window. Its basic goal is to provide an indication of the scale of the scene. Four axes surrounding the render window indicate (in a variety of ways) the scale of what the camera is viewing. An option also exists for displaying a scale legend.

The axes can be programmed either to display distance scales or x-y coordinate values. By default, the scales display a distance. However, if you know that the view is down the z-axis, the scales can be programmed to display x-y coordinate values.

To create an instance of class vtkLegendScaleActor, simply invoke its constructor as follows

```
obj = vtkLegendScaleActor
```

34.28.2 Methods

The class vtkLegendScaleActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLegendScaleActor class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkLegendScaleActor = obj.NewInstance ()` - Standard methods for the class.
- `vtkLegendScaleActor = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.
- `obj.SetLabelMode (int )` - Specify the mode for labeling the scale axes. By default, the axes are labeled with the distance between points (centered at a distance of 0.0). Alternatively if you know that the view is down the z-axis; the axes can be labeled with x-y coordinate values.
- `int = obj.GetLabelModeMinValue ()` - Specify the mode for labeling the scale axes. By default, the axes are labeled with the distance between points (centered at a distance of 0.0). Alternatively if you know that the view is down the z-axis; the axes can be labeled with x-y coordinate values.

- `int = obj.GetLabelModeMaxValue ()` - Specify the mode for labeling the scale axes. By default, the axes are labeled with the distance between points (centered at a distance of 0.0). Alternatively if you know that the view is down the z-axis; the axes can be labeled with x-y coordinate values.
- `int = obj.GetLabelMode ()` - Specify the mode for labeling the scale axes. By default, the axes are labeled with the distance between points (centered at a distance of 0.0). Alternatively if you know that the view is down the z-axis; the axes can be labeled with x-y coordinate values.
- `obj.SetLabelModeToDistance ()` - Specify the mode for labeling the scale axes. By default, the axes are labeled with the distance between points (centered at a distance of 0.0). Alternatively if you know that the view is down the z-axis; the axes can be labeled with x-y coordinate values.
- `obj.SetLabelModeToXYCoordinates ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.SetRightAxisVisibility (int )` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `int = obj.GetRightAxisVisibility ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.RightAxisVisibilityOn ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.RightAxisVisibilityOff ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.SetTopAxisVisibility (int )` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `int = obj.GetTopAxisVisibility ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.TopAxisVisibilityOn ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.TopAxisVisibilityOff ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.SetLeftAxisVisibility (int )` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `int = obj.GetLeftAxisVisibility ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.LeftAxisVisibilityOn ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.LeftAxisVisibilityOff ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.SetBottomAxisVisibility (int )` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `int = obj.GetBottomAxisVisibility ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.BottomAxisVisibilityOn ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.

- `obj.BottomAxisVisibilityOff ()` - Set/Get the flags that control which of the four axes to display (top, bottom, left and right). By default, all the axes are displayed.
- `obj.SetLegendVisibility (int )` - Indicate whether the legend scale should be displayed or not. The default is On.
- `int = obj.GetLegendVisibility ()` - Indicate whether the legend scale should be displayed or not. The default is On.
- `obj.LegendVisibilityOn ()` - Indicate whether the legend scale should be displayed or not. The default is On.
- `obj.LegendVisibilityOff ()` - Indicate whether the legend scale should be displayed or not. The default is On.
- `obj.AllAxesOn ()` - Convenience method that turns all the axes either on or off.
- `obj.AllAxesOff ()` - Convenience method that turns all the axes either on or off.
- `obj.AllAnnotationsOn ()` - Convenience method that turns all the axes and the legend scale.
- `obj.AllAnnotationsOff ()` - Convenience method that turns all the axes and the legend scale.
- `obj.SetRightBorderOffset (int )` - Set/Get the offset of the right axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 50.
- `int = obj.GetRightBorderOffsetMinValue ()` - Set/Get the offset of the right axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 50.
- `int = obj.GetRightBorderOffsetMaxValue ()` - Set/Get the offset of the right axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 50.
- `int = obj.GetRightBorderOffset ()` - Set/Get the offset of the right axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 50.
- `obj.SetTopBorderOffset (int )` - Set/Get the offset of the top axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 30.
- `int = obj.GetTopBorderOffsetMinValue ()` - Set/Get the offset of the top axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 30.
- `int = obj.GetTopBorderOffsetMaxValue ()` - Set/Get the offset of the top axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 30.
- `int = obj.GetTopBorderOffset ()` - Set/Get the offset of the top axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 30.
- `obj.SetLeftBorderOffset (int )` - Set/Get the offset of the left axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 50.

- `int = obj.GetLeftBorderOffsetMinValue ()` - Set/Get the offset of the left axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 50.

- `int = obj.GetLeftBorderOffsetMaxValue ()` - Set/Get the offset of the left axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 50.

- `int = obj.GetLeftBorderOffset ()` - Set/Get the offset of the left axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 50.

- `obj.SetBottomBorderOffset (int )` - Set/Get the offset of the bottom axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 30.

- `int = obj.GetBottomBorderOffsetMinValue ()` - Set/Get the offset of the bottom axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 30.

- `int = obj.GetBottomBorderOffsetMaxValue ()` - Set/Get the offset of the bottom axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 30.

- `int = obj.GetBottomBorderOffset ()` - Set/Get the offset of the bottom axis from the border. This number is expressed in pixels, and represents the approximate distance of the axes from the sides of the renderer. The default is 30.

- `obj.SetCornerOffsetFactor (double )` - Get/Set the corner offset. This is the offset factor used to offset the axes at the corners. Default value is 2.0.

- `double = obj.GetCornerOffsetFactorMinValue ()` - Get/Set the corner offset. This is the offset factor used to offset the axes at the corners. Default value is 2.0.

- `double = obj.GetCornerOffsetFactorMaxValue ()` - Get/Set the corner offset. This is the offset factor used to offset the axes at the corners. Default value is 2.0.

- `double = obj.GetCornerOffsetFactor ()` - Get/Set the corner offset. This is the offset factor used to offset the axes at the corners. Default value is 2.0.

- `vtkTextProperty = obj.GetLegendTitleProperty ()` - Set/Get the labels text properties for the legend title and labels.

- `vtkTextProperty = obj.GetLegendLabelProperty ()` - Set/Get the labels text properties for the legend title and labels.

- `vtkAxisActor2D = obj.GetRightAxis ()` - These are methods to retrieve the vtkAxisActors used to represent the four axes that form this representation. Users may retrieve and then modify these axes to control their appearance.

- `vtkAxisActor2D = obj.GetTopAxis ()` - These are methods to retrieve the vtkAxisActors used to represent the four axes that form this representation. Users may retrieve and then modify these axes to control their appearance.

- `vtkAxisActor2D = obj.GetLeftAxis ()` - These are methods to retrieve the vtkAxisActors used to represent the four axes that form this representation. Users may retrieve and then modify these axes to control their appearance.

- `vtkAxisActor2D = obj.GetBottomAxis ()` - These are methods to retrieve the vtkAxisActors used to represent the four axes that form this representation. Users may retrieve and then modify these axes to control their appearance.
- `obj.BuildRepresentation (vtkViewport viewport)`
- `obj.GetActors2D (vtkPropCollection )`
- `obj.ReleaseGraphicsResources (vtkWindow )`
- `int = obj.RenderOverlay (vtkViewport )`
- `int = obj.RenderOpaqueGeometry (vtkViewport )`

34.29 vtkLSDynaReader

34.29.1 Usage

This filter reads LS-Dyna databases.

The Set/GetFileName() routines are actually wrappers around the Set/GetDatabaseDirectory() members; the actual filename you choose is irrelevant – only the directory name is used. This is done in order to accommodate ParaView.

Note that this reader produces 7 output meshes. These meshes are required as several attributes are defined on subsets of the mesh. Below is a list of meshes in the order they are output and an explanation of which attributes are unique to each mesh: - solid (3D) elements: number of integration points are different than 2D - thick shell elements: number of integration points are different than planar 2D - shell (2D) elements: number of integration points are different than 3D - rigid surfaces: can't have deflection, only velocity, accel, etc. - road surfaces: have only a "segment ID" (serves as material ID) and a velocity. - beam elements: have Frenet (TNB) frame and cross-section attributes (shape and size) - spherical particle hydrodynamics (SPH) elements: have a radius of influence, internal energy, etc. Because each mesh has its own cell attributes, the vtkLSDynaReader has a rather large API. Instead of a single set of routines to query and set cell array names and status, one exists for each possible output mesh. Also, GetNumberOfCells() will return the sum of all the cells in all 7 meshes. If you want the number of cells in a specific mesh, there are separate routines for each mesh type.

.SECTION "Developer Notes"

To create an instance of class vtkLSDynaReader, simply invoke its constructor as follows

```
obj = vtkLSDynaReader
```

34.29.2 Methods

The class vtkLSDynaReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLSDynaReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLSDynaReader = obj.NewInstance ()`
- `vtkLSDynaReader = obj.SafeDownCast (vtkObject o)`
- `obj.DebugDump ()` - A routine to call Dump() from within a lame debugger that won't properly pass a C++ iostream object like cout.
- `int = obj.CanReadFile (string fname)` - Determine if the file can be readed with this reader.

- `obj.SetDatabaseDirectory (string )` - Get/Set the directory containing the LS-Dyna database and determine whether it is valid.
- `string = obj.GetDatabaseDirectory ()` - Get/Set the directory containing the LS-Dyna database and determine whether it is valid.
- `int = obj.IsDatabaseValid ()` - Get/Set the directory containing the LS-Dyna database and determine whether it is valid.
- `obj.SetFileName (string )` - Get/Set the filename. The Set/GetFileName() routines are actually wrappers around the Set/GetDatabaseDirectory() members; the actual filename you choose is irrelevant – only the directory name is used. This is done in order to accommodate ParaView.
- `string = obj.GetFileName ()` - Get/Set the filename. The Set/GetFileName() routines are actually wrappers around the Set/GetDatabaseDirectory() members; the actual filename you choose is irrelevant – only the directory name is used. This is done in order to accommodate ParaView.
- `string = obj.GetTitle ()` - The title of the database is a 40 or 80 character text description stored at the front of a d3plot file. Do not call this function before setting the database directory and calling UpdateInformation().
- `int = obj.GetDimensionality ()` - Retrieve the dimension of points in the database. This should return 2 or 3. Do not call this function before setting the database directory and calling UpdateInformation().
- `vtkIdType = obj.GetNumberOfNodes ()` - Retrieve the number of points in the database. Do not call this function before setting the database directory and calling UpdateInformation().
- `vtkIdType = obj.GetNumberOfCells ()` - Retrieve the number of cells of a given type in the database. Do not call this function before setting the database directory and calling UpdateInformation().

  Note that GetNumberOfCells() returns the sum of GetNumberOfContinuumCells() and GetNumberOfParticleCells().
- `vtkIdType = obj.GetNumberOfContinuumCells ()` - Retrieve the number of cells of a given type in the database. Do not call this function before setting the database directory and calling UpdateInformation().

  Note that GetNumberOfContinuumCells() returns the sum of GetNumberOfSolidCells(), GetNumberOfThickShellCells(), GetNumberOfShellCells(), GetNumberOfRigidBodyCells(), GetNumberOfRoadSurfaceCells(), and GetNumberOfBeamCells().
- `vtkIdType = obj.GetNumberOfSolidCells ()` - Retrieve the number of cells of a given type in the database. Do not call this function before setting the database directory and calling UpdateInformation().
- `vtkIdType = obj.GetNumberOfThickShellCells ()` - Retrieve the number of cells of a given type in the database. Do not call this function before setting the database directory and calling UpdateInformation().
- `vtkIdType = obj.GetNumberOfShellCells ()` - Retrieve the number of cells of a given type in the database. Do not call this function before setting the database directory and calling UpdateInformation().
- `vtkIdType = obj.GetNumberOfRigidBodyCells ()` - Retrieve the number of cells of a given type in the database. Do not call this function before setting the database directory and calling UpdateInformation().
- `vtkIdType = obj.GetNumberOfRoadSurfaceCells ()` - Retrieve the number of cells of a given type in the database. Do not call this function before setting the database directory and calling UpdateInformation().

- `vtkIdType = obj.GetNumberOfBeamCells ()` - Retrieve the number of cells of a given type in the database. Do not call this function before setting the database directory and calling UpdateInformation().
- `vtkIdType = obj.GetNumberOfParticleCells ()` - Retrieve the number of cells of a given type in the database. Do not call this function before setting the database directory and calling UpdateInformation().
- `vtkIdType = obj.GetNumberOfTimeSteps ()` - Retrieve information about the time extents of the LS-Dyna database. Do not call these functions before setting the database directory and calling UpdateInformation().
- `obj.SetTimeStep (vtkIdType )` - Retrieve information about the time extents of the LS-Dyna database. Do not call these functions before setting the database directory and calling UpdateInformation().
- `vtkIdType = obj.GetTimeStep ()` - Retrieve information about the time extents of the LS-Dyna database. Do not call these functions before setting the database directory and calling UpdateInformation().
- `double = obj.GetTimeValue (vtkIdType )` - Retrieve information about the time extents of the LS-Dyna database. Do not call these functions before setting the database directory and calling UpdateInformation().
- `int = obj. GetTimeStepRange ()` - Retrieve information about the time extents of the LS-Dyna database. Do not call these functions before setting the database directory and calling UpdateInformation().
- `obj.SetTimeStepRange (int , int )` - Retrieve information about the time extents of the LS-Dyna database. Do not call these functions before setting the database directory and calling UpdateInformation().
- `obj.SetTimeStepRange (int a[2])` - Retrieve information about the time extents of the LS-Dyna database. Do not call these functions before setting the database directory and calling UpdateInformation().
- `int = obj.GetNumberOfPointArrays ()` - These methods allow you to load only selected subsets of the nodal variables defined over the mesh.
- `string = obj.GetPointArrayName (int )` - These methods allow you to load only selected subsets of the nodal variables defined over the mesh.
- `obj.SetPointArrayStatus (int arr, int status)` - These methods allow you to load only selected subsets of the nodal variables defined over the mesh.
- `obj.SetPointArrayStatus (string arrName, int status)` - These methods allow you to load only selected subsets of the nodal variables defined over the mesh.
- `int = obj.GetPointArrayStatus (int arr)` - These methods allow you to load only selected subsets of the nodal variables defined over the mesh.
- `int = obj.GetPointArrayStatus (string arrName)` - These methods allow you to load only selected subsets of the nodal variables defined over the mesh.
- `int = obj.GetNumberOfComponentsInPointArray (int arr)` - These methods allow you to load only selected subsets of the nodal variables defined over the mesh.
- `int = obj.GetNumberOfComponentsInPointArray (string arrName)` - These methods allow you to load only selected subsets of the nodal variables defined over the mesh.

- `int = obj.GetNumberOfCellArrays (int cellType)` - Routines that allow the status of a cell variable to be adjusted or queried independent of the output mesh. The cellType parameter should be one of: LS_POINT, LS_BEAM, LS_SHELL, LS_THICK_SHELL, LS_SOLID, LS_RIGID_BODY, or LS_ROAD_SURFACE

- `string = obj.GetCellArrayName (int cellType, int arr)` - Routines that allow the status of a cell variable to be adjusted or queried independent of the output mesh. The cellType parameter should be one of: LS_POINT, LS_BEAM, LS_SHELL, LS_THICK_SHELL, LS_SOLID, LS_RIGID_BODY, or LS_ROAD_SURFACE

- `obj.SetCellArrayStatus (int cellType, int arr, int status)` - Routines that allow the status of a cell variable to be adjusted or queried independent of the output mesh. The cellType parameter should be one of: LS_POINT, LS_BEAM, LS_SHELL, LS_THICK_SHELL, LS_SOLID, LS_RIGID_BODY, or LS_ROAD_SURFACE

- `obj.SetCellArrayStatus (int cellType, string arrName, int status)` - Routines that allow the status of a cell variable to be adjusted or queried independent of the output mesh. The cellType parameter should be one of: LS_POINT, LS_BEAM, LS_SHELL, LS_THICK_SHELL, LS_SOLID, LS_RIGID_BODY, or LS_ROAD_SURFACE

- `int = obj.GetCellArrayStatus (int cellType, int arr)` - Routines that allow the status of a cell variable to be adjusted or queried independent of the output mesh. The cellType parameter should be one of: LS_POINT, LS_BEAM, LS_SHELL, LS_THICK_SHELL, LS_SOLID, LS_RIGID_BODY, or LS_ROAD_SURFACE

- `int = obj.GetCellArrayStatus (int cellType, string arrName)` - Routines that allow the status of a cell variable to be adjusted or queried independent of the output mesh. The cellType parameter should be one of: LS_POINT, LS_BEAM, LS_SHELL, LS_THICK_SHELL, LS_SOLID, LS_RIGID_BODY, or LS_ROAD_SURFACE

- `int = obj.GetNumberOfComponentsInCellArray (int cellType, int arr)` - Routines that allow the status of a cell variable to be adjusted or queried independent of the output mesh. The cellType parameter should be one of: LS_POINT, LS_BEAM, LS_SHELL, LS_THICK_SHELL, LS_SOLID, LS_RIGID_BODY, or LS_ROAD_SURFACE

- `int = obj.GetNumberOfComponentsInCellArray (int cellType, string arrName)` - Routines that allow the status of a cell variable to be adjusted or queried independent of the output mesh. The cellType parameter should be one of: LS_POINT, LS_BEAM, LS_SHELL, LS_THICK_SHELL, LS_SOLID, LS_RIGID_BODY, or LS_ROAD_SURFACE

- `int = obj.GetNumberOfSolidArrays ()` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.

- `string = obj.GetSolidArrayName (int )` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.

- `obj.SetSolidArrayStatus (int arr, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.

- `obj.SetSolidArrayStatus (string arrName, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.

- `int = obj.GetSolidArrayStatus (int arr)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.

- `int = obj.GetSolidArrayStatus (string arrName)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.

- `int = obj.GetNumberOfComponentsInSolidArray (int a)`

- `int = obj.GetNumberOfComponentsInSolidArray (string arrName)`
- `int = obj.GetNumberOfThickShellArrays ()` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `string = obj.GetThickShellArrayName (int )` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetThickShellArrayStatus (int arr, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetThickShellArrayStatus (string arrName, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetThickShellArrayStatus (int arr)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetThickShellArrayStatus (string arrName)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetNumberOfComponentsInThickShellArray (int a)`
- `int = obj.GetNumberOfComponentsInThickShellArray (string arrName)`
- `int = obj.GetNumberOfShellArrays ()` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `string = obj.GetShellArrayName (int )` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetShellArrayStatus (int arr, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetShellArrayStatus (string arrName, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetShellArrayStatus (int arr)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetShellArrayStatus (string arrName)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetNumberOfComponentsInShellArray (int a)`
- `int = obj.GetNumberOfComponentsInShellArray (string arrName)`
- `int = obj.GetNumberOfRigidBodyArrays ()` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `string = obj.GetRigidBodyArrayName (int )` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetRigidBodyArrayStatus (int arr, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetRigidBodyArrayStatus (string arrName, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetRigidBodyArrayStatus (int arr)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.

- `int = obj.GetRigidBodyArrayStatus (string arrName)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetNumberOfComponentsInRigidBodyArray (int a)`
- `int = obj.GetNumberOfComponentsInRigidBodyArray (string arrName)`
- `int = obj.GetNumberOfRoadSurfaceArrays ()` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `string = obj.GetRoadSurfaceArrayName (int )` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetRoadSurfaceArrayStatus (int arr, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetRoadSurfaceArrayStatus (string arrName, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetRoadSurfaceArrayStatus (int arr)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetRoadSurfaceArrayStatus (string arrName)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetNumberOfComponentsInRoadSurfaceArray (int a)`
- `int = obj.GetNumberOfComponentsInRoadSurfaceArray (string arrName)`
- `int = obj.GetNumberOfBeamArrays ()` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `string = obj.GetBeamArrayName (int )` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetBeamArrayStatus (int arr, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetBeamArrayStatus (string arrName, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetBeamArrayStatus (int arr)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetBeamArrayStatus (string arrName)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetNumberOfComponentsInBeamArray (int a)`
- `int = obj.GetNumberOfComponentsInBeamArray (string arrName)`
- `int = obj.GetNumberOfParticleArrays ()` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `string = obj.GetParticleArrayName (int )` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetParticleArrayStatus (int arr, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `obj.SetParticleArrayStatus (string arrName, int status)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.

- `int = obj.GetParticleArrayStatus (int arr)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetParticleArrayStatus (string arrName)` - These methods allow you to load only selected subsets of the cell variables defined over the mesh.
- `int = obj.GetNumberOfComponentsInParticleArray (int a)`
- `int = obj.GetNumberOfComponentsInParticleArray (string arrName)`
- `obj.SetDeformedMesh (int )` - Should deflected coordinates be used, or should the mesh remain undeflected? By default, this is true but its value is ignored if the nodal "Deflection" array is not set to be loaded.
- `int = obj.GetDeformedMesh ()` - Should deflected coordinates be used, or should the mesh remain undeflected? By default, this is true but its value is ignored if the nodal "Deflection" array is not set to be loaded.
- `obj.DeformedMeshOn ()` - Should deflected coordinates be used, or should the mesh remain undeflected? By default, this is true but its value is ignored if the nodal "Deflection" array is not set to be loaded.
- `obj.DeformedMeshOff ()` - Should deflected coordinates be used, or should the mesh remain undeflected? By default, this is true but its value is ignored if the nodal "Deflection" array is not set to be loaded.
- `obj.SetRemoveDeletedCells (int )` - Should dead cells be removed from the mesh? Cells are marked dead by setting the corresponding entry in the ¡b¿cell¡/b¿ array "Death" to 0. Cells that are not dead have the corresponding entry in the cell array "Death" set to their material ID. By default, this is true but its value is ignored if the cell "Death" array is not set to be loaded. It is also ignored if the database's element deletion option is set to denote ¡b¿points¡/b¿ (not cells) as deleted; in that case, "Death" will appear to be a point array.
- `int = obj.GetRemoveDeletedCells ()` - Should dead cells be removed from the mesh? Cells are marked dead by setting the corresponding entry in the ¡b¿cell¡/b¿ array "Death" to 0. Cells that are not dead have the corresponding entry in the cell array "Death" set to their material ID. By default, this is true but its value is ignored if the cell "Death" array is not set to be loaded. It is also ignored if the database's element deletion option is set to denote ¡b¿points¡/b¿ (not cells) as deleted; in that case, "Death" will appear to be a point array.
- `obj.RemoveDeletedCellsOn ()` - Should dead cells be removed from the mesh? Cells are marked dead by setting the corresponding entry in the ¡b¿cell¡/b¿ array "Death" to 0. Cells that are not dead have the corresponding entry in the cell array "Death" set to their material ID. By default, this is true but its value is ignored if the cell "Death" array is not set to be loaded. It is also ignored if the database's element deletion option is set to denote ¡b¿points¡/b¿ (not cells) as deleted; in that case, "Death" will appear to be a point array.
- `obj.RemoveDeletedCellsOff ()` - Should dead cells be removed from the mesh? Cells are marked dead by setting the corresponding entry in the ¡b¿cell¡/b¿ array "Death" to 0. Cells that are not dead have the corresponding entry in the cell array "Death" set to their material ID. By default, this is true but its value is ignored if the cell "Death" array is not set to be loaded. It is also ignored if the database's element deletion option is set to denote ¡b¿points¡/b¿ (not cells) as deleted; in that case, "Death" will appear to be a point array.
- `obj.SetSplitByMaterialId (int )` - Split each part into submeshes based on material ID. By default, this is false and all cells of a given type (solid, thick shell, shell, ...) are in a single mesh.
- `int = obj.GetSplitByMaterialId ()` - Split each part into submeshes based on material ID. By default, this is false and all cells of a given type (solid, thick shell, shell, ...) are in a single mesh.

- `obj.SplitByMaterialIdOn ()` - Split each part into submeshes based on material ID. By default, this is false and all cells of a given type (solid, thick shell, shell, ...) are in a single mesh.

- `obj.SplitByMaterialIdOff ()` - Split each part into submeshes based on material ID. By default, this is false and all cells of a given type (solid, thick shell, shell, ...) are in a single mesh.

- `obj.SetInputDeck (string )` - The name of the input deck corresponding to the current database. This is used to determine the part names associated with each material ID. This file may be in two formats: a valid LSDyna input deck or a short XML summary. If the file begins with "¡?xml" then the summary format is used. Otherwise, the keyword format is used and a summary file will be created if write permissions exist in the directory containing the keyword file. The newly created summary will have ".k" or ".key" stripped from the end of the keyword filename and ".lsdyna" appended.

- `string = obj.GetInputDeck ()` - The name of the input deck corresponding to the current database. This is used to determine the part names associated with each material ID. This file may be in two formats: a valid LSDyna input deck or a short XML summary. If the file begins with "¡?xml" then the summary format is used. Otherwise, the keyword format is used and a summary file will be created if write permissions exist in the directory containing the keyword file. The newly created summary will have ".k" or ".key" stripped from the end of the keyword filename and ".lsdyna" appended.

- `int = obj.GetNumberOfPartArrays ()` - These methods allow you to load only selected parts of the input. If InputDeck points to a valid keyword file (or summary), then part names will be taken from that file. Otherwise, when arbitrary material numbering is used, parts will be named "PartXXX (MatlYYY)" where XXX is an increasing sequential number and YYY is the respective material ID. If no input deck is specified and arbitrary arbitrary material numbering is not used, parts will be named "PartXXX" where XXX is a sequential material ID.

- `string = obj.GetPartArrayName (int )` - These methods allow you to load only selected parts of the input. If InputDeck points to a valid keyword file (or summary), then part names will be taken from that file. Otherwise, when arbitrary material numbering is used, parts will be named "PartXXX (MatlYYY)" where XXX is an increasing sequential number and YYY is the respective material ID. If no input deck is specified and arbitrary arbitrary material numbering is not used, parts will be named "PartXXX" where XXX is a sequential material ID.

- `obj.SetPartArrayStatus (int arr, int status)` - These methods allow you to load only selected parts of the input. If InputDeck points to a valid keyword file (or summary), then part names will be taken from that file. Otherwise, when arbitrary material numbering is used, parts will be named "PartXXX (MatlYYY)" where XXX is an increasing sequential number and YYY is the respective material ID. If no input deck is specified and arbitrary arbitrary material numbering is not used, parts will be named "PartXXX" where XXX is a sequential material ID.

- `obj.SetPartArrayStatus (string partName, int status)` - These methods allow you to load only selected parts of the input. If InputDeck points to a valid keyword file (or summary), then part names will be taken from that file. Otherwise, when arbitrary material numbering is used, parts will be named "PartXXX (MatlYYY)" where XXX is an increasing sequential number and YYY is the respective material ID. If no input deck is specified and arbitrary arbitrary material numbering is not used, parts will be named "PartXXX" where XXX is a sequential material ID.

- `int = obj.GetPartArrayStatus (int arr)` - These methods allow you to load only selected parts of the input. If InputDeck points to a valid keyword file (or summary), then part names will be taken from that file. Otherwise, when arbitrary material numbering is used, parts will be named "PartXXX (MatlYYY)" where XXX is an increasing sequential number and YYY is the respective material ID. If no input deck is specified and arbitrary arbitrary material numbering is not used, parts will be named "PartXXX" where XXX is a sequential material ID.

- `int = obj.GetPartArrayStatus (string partName)` - These methods allow you to load only selected parts of the input. If InputDeck points to a valid keyword file (or summary), then part names

will be taken from that file. Otherwise, when arbitrary material numbering is used, parts will be named "PartXXX (MatlYYY)" where XXX is an increasing sequential number and YYY is the respective material ID. If no input deck is specified and arbitrary arbitrary material numbering is not used, parts will be named "PartXXX" where XXX is a sequential material ID.

34.30 vtkPCAAnalysisFilter

34.30.1 Usage

vtkPCAAnalysisFilter is a filter that takes as input a set of aligned pointsets (any object derived from vtkPointSet) and performs a principal component analysis of the coordinates. This can be used to visualise the major or minor modes of variation seen in a set of similar biological objects with corresponding landmarks. vtkPCAAnalysisFilter is designed to work with the output from the vtkProcrustesAnalysisFilter

Call SetNumberOfInputs(n) before calling SetInput(0) ... SetInput(n-1). Retrieve the outputs using GetOutput(0) ... GetOutput(n-1).

vtkPCAAnalysisFilter is an implementation of (for example):

T. Cootes et al. : Active Shape Models - their training and application. Computer Vision and Image Understanding, 61(1):38-59, 1995.

The material can also be found in Tim Cootes' ever-changing online report published at his website: http://www.isbe.man.ac.uk/ bim/

To create an instance of class vtkPCAAnalysisFilter, simply invoke its constructor as follows

```
obj = vtkPCAAnalysisFilter
```

34.30.2 Methods

The class vtkPCAAnalysisFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPCAAnalysisFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPCAAnalysisFilter = obj.NewInstance ()`
- `vtkPCAAnalysisFilter = obj.SafeDownCast (vtkObject o)`
- `vtkFloatArray = obj.GetEvals ()` - Get the vector of eigenvalues sorted in descending order
- `obj.SetNumberOfInputs (int n)` - Specify how many pointsets are going to be given as input.
- `obj.SetInput (int idx, vtkPointSet p)` - Specify the input pointset with index idx. Call SetNumberOfInputs before calling this function.
- `obj.SetInput (int idx, vtkDataObject input)` - Specify the input pointset with index idx. Call SetNumberOfInputs before calling this function.
- `vtkPointSet = obj.GetInput (int idx)` - Retrieve the input with index idx (usually only used for pipeline tracing).
- `obj.GetParameterisedShape (vtkFloatArray b, vtkPointSet shape)` - Fills the shape with:

  mean + b[0] * sqrt(eigenvalue[0]) * eigenvector[0] + b[1] * sqrt(eigenvalue[1]) * eigenvector[1] ... + b[sizeb-1] * sqrt(eigenvalue[bsize-1]) * eigenvector[bsize-1]

  here b are the parameters expressed in standard deviations bsize is the number of parameters in the b vector This function assumes that shape is allready allocated with the right size, it just moves the points.

- `obj.GetShapeParameters (vtkPointSet shape, vtkFloatArray b, int bsize)` - Return the bsize parameters b that best model the given shape (in standard deviations). That is that the given shape will be approximated by:

  shape mean + b[0] * sqrt(eigenvalue[0]) * eigenvector[0] + b[1] * sqrt(eigenvalue[1]) * eigenvector[1] ... + b[bsize-1] * sqrt(eigenvalue[bsize-1]) * eigenvector[bsize-1]

- `int = obj.GetModesRequiredFor (double proportion)` - Retrieve how many modes are necessary to model the given proportion of the variation. proportion should be between 0 and 1

34.31 vtkPExodusIIReader

34.31.1 Usage

vtkPExodusIIReader is a unstructured grid source object that reads ExodusII files. Most of the meta data associated with the file is loaded when UpdateInformation is called. This includes information like Title, number of blocks, number and names of arrays. This data can be retrieved from methods in this reader. Separate arrays that are meant to be a single vector, are combined internally for convenience. To be combined, the array names have to be identical except for a trailing X,Y and Z (or x,y,z). By default all cell and point arrays are loaded. However, the user can flag arrays not to load with the methods "SetPointDataArrayLoadFlag" and "SetCellDataArrayLoadFlag". The reader responds to piece requests by loading only a range of the possible blocks. Unused points are filtered out internally.

To create an instance of class vtkPExodusIIReader, simply invoke its constructor as follows

```
obj = vtkPExodusIIReader
```

34.31.2 Methods

The class vtkPExodusIIReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPExodusIIReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPExodusIIReader = obj.NewInstance ()`
- `vtkPExodusIIReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController c)` - Set/get the communication object used to relay a list of files from the rank 0 process to all others. This is the only interprocess communication required by vtkPExodusIIReader.
- `vtkMultiProcessController = obj.GetController ()` - Set/get the communication object used to relay a list of files from the rank 0 process to all others. This is the only interprocess communication required by vtkPExodusIIReader.
- `obj.SetFilePattern (string )` - These methods tell the reader that the data is ditributed across multiple files. This is for distributed execution. It this case, pieces are mapped to files. The pattern should have one format the file number. FileNumberRange is used to generate file numbers. I was thinking of having an arbitrary list of file numbers. This may happen in the future. (That is why there is no GetFileNumberRange method.
- `string = obj.GetFilePattern ()` - These methods tell the reader that the data is ditributed across multiple files. This is for distributed execution. It this case, pieces are mapped to files. The pattern should have one format the file number. FileNumberRange is used to generate file numbers. I was thinking of having an arbitrary list of file numbers. This may happen in the future. (That is why there is no GetFileNumberRange method.

- `obj.SetFilePrefix (string )` - These methods tell the reader that the data is ditributed across multiple files. This is for distributed execution. It this case, pieces are mapped to files. The pattern should have one format the file number. FileNumberRange is used to generate file numbers. I was thinking of having an arbitrary list of file numbers. This may happen in the future. (That is why there is no GetFileNumberRange method.

- `string = obj.GetFilePrefix ()` - These methods tell the reader that the data is ditributed across multiple files. This is for distributed execution. It this case, pieces are mapped to files. The pattern should have one format the file number. FileNumberRange is used to generate file numbers. I was thinking of having an arbitrary list of file numbers. This may happen in the future. (That is why there is no GetFileNumberRange method.

- `obj.SetFileRange (int , int )` - Set the range of files that are being loaded. The range for single file should add to 0.

- `obj.SetFileRange (int r)` - Set the range of files that are being loaded. The range for single file should add to 0.

- `int = obj. GetFileRange ()` - Set the range of files that are being loaded. The range for single file should add to 0.

- `obj.SetFileName (string name)`

- `int = obj.GetNumberOfFileNames ()` - Return the number of files to be read.

- `int = obj.GetNumberOfFiles ()` - Return the number of files to be read.

- `vtkIdType = obj.GetTotalNumberOfElements ()`

- `vtkIdType = obj.GetTotalNumberOfNodes ()`

- `obj.UpdateTimeInformation ()` - Calls UpdateTimeInformation() on all serial readers so they'll re-read their time info from the file. The last time step that they all have in common is stored in LastCommonTimeStep, which is used in RequestInformation() to override the output time-specific information keys with the range of times that ALL readers can actually read.

- `obj.Broadcast (vtkMultiProcessController ctrl)` - Sends metadata (that read from the input file, not settings modified through this API) from the rank 0 node to all other processes in a job.

34.32 vtkPExodusReader

34.32.1 Usage

vtkPExodusReader is a unstructured grid source object that reads PExodusReaderII files. Most of the meta data associated with the file is loaded when UpdateInformation is called. This includes information like Title, number of blocks, number and names of arrays. This data can be retrieved from methods in this reader. Separate arrays that are meant to be a single vector, are combined internally for convenience. To be combined, the array names have to be identical except for a trailing X,Y and Z (or x,y,z). By default all cell and point arrays are loaded. However, the user can flag arrays not to load with the methods "SetPointDataArrayLoadFlag" and "SetCellDataArrayLoadFlag". The reader responds to piece requests by loading only a range of the possible blocks. Unused points are filtered out internally.

To create an instance of class vtkPExodusReader, simply invoke its constructor as follows

```
obj = vtkPExodusReader
```

34.32.2 Methods

The class vtkPExodusReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPExodusReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPExodusReader = obj.NewInstance ()`
- `vtkPExodusReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFilePattern (string )` - These methods tell the reader that the data is ditributed across multiple files. This is for distributed execution. It this case, pieces are mapped to files. The pattern should have one format the file number. FileNumberRange is used to generate file numbers. I was thinking of having an arbitrary list of file numbers. This may happen in the future. (That is why there is no GetFileNumberRange method.
- `string = obj.GetFilePattern ()` - These methods tell the reader that the data is ditributed across multiple files. This is for distributed execution. It this case, pieces are mapped to files. The pattern should have one format the file number. FileNumberRange is used to generate file numbers. I was thinking of having an arbitrary list of file numbers. This may happen in the future. (That is why there is no GetFileNumberRange method.
- `obj.SetFilePrefix (string )` - These methods tell the reader that the data is ditributed across multiple files. This is for distributed execution. It this case, pieces are mapped to files. The pattern should have one format the file number. FileNumberRange is used to generate file numbers. I was thinking of having an arbitrary list of file numbers. This may happen in the future. (That is why there is no GetFileNumberRange method.
- `string = obj.GetFilePrefix ()` - These methods tell the reader that the data is ditributed across multiple files. This is for distributed execution. It this case, pieces are mapped to files. The pattern should have one format the file number. FileNumberRange is used to generate file numbers. I was thinking of having an arbitrary list of file numbers. This may happen in the future. (That is why there is no GetFileNumberRange method.
- `obj.SetFileRange (int , int )` - Set the range of files that are being loaded. The range for single file should add to 0.
- `obj.SetFileRange (int r)` - Set the range of files that are being loaded. The range for single file should add to 0.
- `int = obj. GetFileRange ()` - Set the range of files that are being loaded. The range for single file should add to 0.
- `obj.SetFileName (string name)`
- `int = obj.GetNumberOfFileNames ()` - Return the number of files to be read.
- `int = obj.GetNumberOfFiles ()` - Return the number of files to be read.
- `obj.SetGenerateFileIdArray (int flag)`
- `int = obj.GetGenerateFileIdArray ()`
- `obj.GenerateFileIdArrayOn ()`
- `obj.GenerateFileIdArrayOff ()`

- `int = obj.GetTotalNumberOfElements ()`
- `int = obj.GetTotalNumberOfNodes ()`
- `int = obj.GetNumberOfVariableArrays ()`
- `string = obj.GetVariableArrayName (int a\_which)`
- `obj.EnableDSPFiltering ()`
- `obj.AddFilter (vtkDSPFilterDefinition a\_filter)`
- `obj.StartAddingFilter ()`
- `obj.AddFilterInputVar (string name)`
- `obj.AddFilterOutputVar (string name)`
- `obj.AddFilterNumeratorWeight (double weight)`
- `obj.AddFilterForwardNumeratorWeight (double weight)`
- `obj.AddFilterDenominatorWeight (double weight)`
- `obj.FinishAddingFilter ()`
- `obj.RemoveFilter (string a\_outputVariableName)`

34.33 vtkPieChartActor

34.33.1 Usage

vtkPieChartActor generates a pie chart from an array of numbers defined in field data (a vtkDataObject). To use this class, you must specify an input data object. You'll probably also want to specify the position of the plot be setting the Position and Position2 instance variables, which define a rectangle in which the plot lies. There are also many other instance variables that control the look of the plot includes its title, and legend.

Set the text property/attributes of the title and the labels through the vtkTextProperty objects associated with these components.

To create an instance of class vtkPieChartActor, simply invoke its constructor as follows

```
obj = vtkPieChartActor
```

34.33.2 Methods

The class vtkPieChartActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPieChartActor class.

- `string = obj.GetClassName ()` - Standard methods for type information and printing.
- `int = obj.IsA (string name)` - Standard methods for type information and printing.
- `vtkPieChartActor = obj.NewInstance ()` - Standard methods for type information and printing.
- `vtkPieChartActor = obj.SafeDownCast (vtkObject o)` - Standard methods for type information and printing.
- `obj.SetInput (vtkDataObject )` - Set the input to the pie chart actor.

- `vtkDataObject = obj.GetInput ()` - Get the input data object to this actor.
- `obj.SetTitleVisibility (int )` - Enable/Disable the display of a plot title.
- `int = obj.GetTitleVisibility ()` - Enable/Disable the display of a plot title.
- `obj.TitleVisibilityOn ()` - Enable/Disable the display of a plot title.
- `obj.TitleVisibilityOff ()` - Enable/Disable the display of a plot title.
- `obj.SetTitle (string )` - Set/Get the title of the pie chart.
- `string = obj.GetTitle ()` - Set/Get the title of the pie chart.
- `obj.SetTitleTextProperty (vtkTextProperty p)` - Set/Get the title text property. The property controls the appearance of the plot title.
- `vtkTextProperty = obj.GetTitleTextProperty ()` - Set/Get the title text property. The property controls the appearance of the plot title.
- `obj.SetLabelVisibility (int )` - Enable/Disable the display of pie piece labels.
- `int = obj.GetLabelVisibility ()` - Enable/Disable the display of pie piece labels.
- `obj.LabelVisibilityOn ()` - Enable/Disable the display of pie piece labels.
- `obj.LabelVisibilityOff ()` - Enable/Disable the display of pie piece labels.
- `obj.SetLabelTextProperty (vtkTextProperty p)` - Set/Get the labels text property. This controls the appearance of all pie piece labels.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - Set/Get the labels text property. This controls the appearance of all pie piece labels.
- `obj.SetPieceColor (int i, double r, double g, double b)` - Specify colors for each piece of pie. If not specified, they are automatically generated.
- `obj.SetPieceColor (int i, double color[3])` - Specify colors for each piece of pie. If not specified, they are automatically generated.
- `obj.SetPieceLabel (int i, string )` - Specify the names for each piece of pie. not specified, then an integer number is automatically generated.
- `string = obj.GetPieceLabel (int i)` - Specify the names for each piece of pie. not specified, then an integer number is automatically generated.
- `obj.SetLegendVisibility (int )` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `int = obj.GetLegendVisibility ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.LegendVisibilityOn ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.LegendVisibilityOff ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `vtkLegendBoxActor = obj.GetLegendActor ()` - Retrieve handles to the legend box. This is useful if you would like to manually control the legend appearance.
- `int = obj.RenderOverlay (vtkViewport )` - Draw the pie plot.

- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Draw the pie plot.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Does this prop have some translucent polygonal geometry?
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.

34.34 vtkPolyDataSilhouette

34.34.1 Usage

vtkPolyDataSilhouette extracts a subset of a polygonal mesh edges to generate an outline (silhouette) of the corresponding 3D object. In addition, this filter can also extracts sharp edges (aka feature angles). In order to use this filter you must specify the a point of view (origin) or a direction (vector). given this direction or origin, a silhouette is generated wherever the surface's normal is orthogonal to the view direction.

To create an instance of class vtkPolyDataSilhouette, simply invoke its constructor as follows

```
obj = vtkPolyDataSilhouette
```

34.34.2 Methods

The class vtkPolyDataSilhouette has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataSilhouette class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataSilhouette = obj.NewInstance ()`
- `vtkPolyDataSilhouette = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnableFeatureAngle (int )` - Enables or Disables generation of silhouette edges along sharp edges
- `int = obj.GetEnableFeatureAngle ()` - Enables or Disables generation of silhouette edges along sharp edges
- `obj.SetFeatureAngle (double )` - Sets/Gets minimal angle for sharp edges detection. Default is 60
- `double = obj.GetFeatureAngle ()` - Sets/Gets minimal angle for sharp edges detection. Default is 60
- `obj.SetBorderEdges (int )` - Enables or Disables generation of border edges. Note: borders exist only in case of non closed surface
- `int = obj.GetBorderEdges ()` - Enables or Disables generation of border edges. Note: borders exist only in case of non closed surface
- `obj.BorderEdgesOn ()` - Enables or Disables generation of border edges. Note: borders exist only in case of non closed surface

- `obj.BorderEdgesOff ()` - Enables or Disables generation of border edges. Note: borders exist only in case of non closed surface
- `obj.SetPieceInvariant (int )` - Enables or Disables piece invariance. This is usefull when dealing with multi-block data sets. Note: requires one level of ghost cells
- `int = obj.GetPieceInvariant ()` - Enables or Disables piece invariance. This is usefull when dealing with multi-block data sets. Note: requires one level of ghost cells
- `obj.PieceInvariantOn ()` - Enables or Disables piece invariance. This is usefull when dealing with multi-block data sets. Note: requires one level of ghost cells
- `obj.PieceInvariantOff ()` - Enables or Disables piece invariance. This is usefull when dealing with multi-block data sets. Note: requires one level of ghost cells
- `obj.SetDirection (int )` - Specify how view direction is computed. By default, the camera origin (eye) is used.
- `int = obj.GetDirection ()` - Specify how view direction is computed. By default, the camera origin (eye) is used.
- `obj.SetDirectionToSpecifiedVector ()` - Specify how view direction is computed. By default, the camera origin (eye) is used.
- `obj.SetDirectionToSpecifiedOrigin ()` - Specify how view direction is computed. By default, the camera origin (eye) is used.
- `obj.SetDirectionToCameraVector ()` - Specify how view direction is computed. By default, the camera origin (eye) is used.
- `obj.SetDirectionToCameraOrigin ()` - Specify a camera that is used to define the view direction. This ivar only has effect if the direction is set to VTK_DIRECTION_CAMERA_ORIGIN or VTK_DIRECTION_CAMERA_VECTOR, and a camera is specified.
- `obj.SetCamera (vtkCamera )` - Specify a camera that is used to define the view direction. This ivar only has effect if the direction is set to VTK_DIRECTION_CAMERA_ORIGIN or VTK_DIRECTION_CAMERA_VECT and a camera is specified.
- `vtkCamera = obj.GetCamera ()` - Specify a camera that is used to define the view direction. This ivar only has effect if the direction is set to VTK_DIRECTION_CAMERA_ORIGIN or VTK_DIRECTION_CAMERA_VECT and a camera is specified.
- `obj.SetProp3D (vtkProp3D )` - Specify a transformation matrix (via the vtkProp3D::GetMatrix() method) that is used to include the effects of transformation. This ivar only has effect if the direction is set to VTK_DIRECTION_CAMERA_ORIGIN or VTK_DIRECTION_CAMERA_VECTOR, and a camera is specified. Specifying the vtkProp3D is optional.
- `vtkProp3D = obj.GetProp3D ()` - Specify a transformation matrix (via the vtkProp3D::GetMatrix() method) that is used to include the effects of transformation. This ivar only has effect if the direction is set to VTK_DIRECTION_CAMERA_ORIGIN or VTK_DIRECTION_CAMERA_VECTOR, and a camera is specified. Specifying the vtkProp3D is optional.
- `obj.SetVector (double , double , double )` - Set/Get the sort direction. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedVector(). The edge detection occurs in the direction of the vector.
- `obj.SetVector (double a[3])` - Set/Get the sort direction. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedVector(). The edge detection occurs in the direction of the vector.

- `double = obj. GetVector ()` - Set/Get the sort direction. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedVector(). The edge detection occurs in the direction of the vector.
- `obj.SetOrigin (double , double , double )` - Set/Get the sort origin. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedOrigin(). The edge detection occurs in the direction of the origin to each edge's center.
- `obj.SetOrigin (double a[3])` - Set/Get the sort origin. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedOrigin(). The edge detection occurs in the direction of the origin to each edge's center.
- `double = obj. GetOrigin ()` - Set/Get the sort origin. This ivar only has effect if the sort direction is set to SetDirectionToSpecifiedOrigin(). The edge detection occurs in the direction of the origin to each edge's center.
- `long = obj.GetMTime ()` - Return MTime also considering the dependent objects: the camera and/or the prop3D.

34.35 vtkPolyDataToImageStencil

34.35.1 Usage

The vtkPolyDataToImageStencil class will convert a surface mesh into an image stencil that can be used to mask an image with vtkImageStencil, or used to calculate statistics within the enclosed region with vtkImageAccumulate.

To create an instance of class vtkPolyDataToImageStencil, simply invoke its constructor as follows

```
obj = vtkPolyDataToImageStencil
```

34.35.2 Methods

The class vtkPolyDataToImageStencil has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataToImageStencil class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataToImageStencil = obj.NewInstance ()`
- `vtkPolyDataToImageStencil = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkPolyData )` - Specify the implicit function to convert into a stencil.
- `vtkPolyData = obj.GetInput ()` - Specify the implicit function to convert into a stencil.
- `obj.SetInformationInput (vtkImageData )` - Set a vtkImageData that has the Spacing, Origin, and WholeExtent that will be used for the stencil. This input should be set to the image that you wish to apply the stencil to. If you use this method, then any values set with the SetOutputSpacing, SetOutputOrigin, and SetOutputWholeExtent methods will be ignored.
- `vtkImageData = obj.GetInformationInput ()` - Set a vtkImageData that has the Spacing, Origin, and WholeExtent that will be used for the stencil. This input should be set to the image that you wish to apply the stencil to. If you use this method, then any values set with the SetOutputSpacing, SetOutputOrigin, and SetOutputWholeExtent methods will be ignored.

- `obj.SetOutputOrigin (double , double , double )` - Set the Origin to be used for the stencil. It should be set to the Origin of the image you intend to apply the stencil to. The default value is (0,0,0).
- `obj.SetOutputOrigin (double a[3])` - Set the Origin to be used for the stencil. It should be set to the Origin of the image you intend to apply the stencil to. The default value is (0,0,0).
- `double = obj. GetOutputOrigin ()` - Set the Origin to be used for the stencil. It should be set to the Origin of the image you intend to apply the stencil to. The default value is (0,0,0).
- `obj.SetOutputSpacing (double , double , double )` - Set the Spacing to be used for the stencil. It should be set to the Spacing of the image you intend to apply the stencil to. The default value is (1,1,1)
- `obj.SetOutputSpacing (double a[3])` - Set the Spacing to be used for the stencil. It should be set to the Spacing of the image you intend to apply the stencil to. The default value is (1,1,1)
- `double = obj. GetOutputSpacing ()` - Set the Spacing to be used for the stencil. It should be set to the Spacing of the image you intend to apply the stencil to. The default value is (1,1,1)
- `obj.SetOutputWholeExtent (int , int , int , int , int , int )` - Set the whole extent for the stencil (anything outside this extent will be considered to be "outside" the stencil). If this is not set, then the stencil will always use the requested UpdateExtent as the stencil extent.
- `obj.SetOutputWholeExtent (int a[6])` - Set the whole extent for the stencil (anything outside this extent will be considered to be "outside" the stencil). If this is not set, then the stencil will always use the requested UpdateExtent as the stencil extent.
- `int = obj. GetOutputWholeExtent ()` - Set the whole extent for the stencil (anything outside this extent will be considered to be "outside" the stencil). If this is not set, then the stencil will always use the requested UpdateExtent as the stencil extent.
- `obj.SetTolerance (double )` - The tolerance to apply in when determining whether a voxel is inside the stencil, given as a fraction of a voxel. Only used in X and Y, not in Z.
- `double = obj.GetToleranceMinValue ()` - The tolerance to apply in when determining whether a voxel is inside the stencil, given as a fraction of a voxel. Only used in X and Y, not in Z.
- `double = obj.GetToleranceMaxValue ()` - The tolerance to apply in when determining whether a voxel is inside the stencil, given as a fraction of a voxel. Only used in X and Y, not in Z.
- `double = obj.GetTolerance ()` - The tolerance to apply in when determining whether a voxel is inside the stencil, given as a fraction of a voxel. Only used in X and Y, not in Z.

34.36 vtkProcrustesAlignmentFilter

34.36.1 Usage

vtkProcrustesAlignmentFilter is a filter that takes a set of pointsets (any object derived from vtkPointSet) and aligns them in a least-squares sense to their mutual mean. The algorithm is iterated until convergence, as the mean must be recomputed after each alignment.

Call SetNumberOfInputs(n) before calling SetInput(0) ... SetInput(n-1).

Retrieve the outputs using GetOutput(0) ... GetOutput(n-1).

The default (in vtkLandmarkTransform) is for a similarity alignment. For a rigid-body alignment (to build a 'size-and-shape' model) use:

GetLandmarkTransform()-¿SetModeToRigidBody().

Affine alignments are not normally used but are left in for completeness:

GetLandmarkTransform()-¿SetModeToAffine().

vtkProcrustesAlignmentFilter is an implementation of:

J.C. Gower (1975) Generalized Procrustes Analysis. Psychometrika, 40:33-51.
To create an instance of class vtkProcrustesAlignmentFilter, simply invoke its constructor as follows

```
obj = vtkProcrustesAlignmentFilter
```

34.36.2 Methods

The class vtkProcrustesAlignmentFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProcrustesAlignmentFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProcrustesAlignmentFilter = obj.NewInstance ()`
- `vtkProcrustesAlignmentFilter = obj.SafeDownCast (vtkObject o)`
- `vtkLandmarkTransform = obj.GetLandmarkTransform ()` - Get the internal landmark transform. Use it to constrain the number of degrees of freedom of the alignment (i.e. rigid body, similarity, etc.). The default is a similarity alignment.
- `vtkPoints = obj.GetMeanPoints ()` - Get the estimated mean point cloud
- `obj.SetNumberOfInputs (int n)` - Specify how many pointsets are going to be given as input.
- `obj.SetInput (int idx, vtkPointSet p)` - Specify the input pointset with index idx. Call SetNumberOfInputs before calling this function.
- `obj.SetInput (int idx, vtkDataObject input)` - Specify the input pointset with index idx. Call SetNumberOfInputs before calling this function.
- `obj.SetStartFromCentroid (bool )` - When on, the initial alignment is to the centroid of the cohort curves. When off, the alignment is to the centroid of the first input. Default is off for backward compatibility.
- `bool = obj.GetStartFromCentroid ()` - When on, the initial alignment is to the centroid of the cohort curves. When off, the alignment is to the centroid of the first input. Default is off for backward compatibility.
- `obj.StartFromCentroidOn ()` - When on, the initial alignment is to the centroid of the cohort curves. When off, the alignment is to the centroid of the first input. Default is off for backward compatibility.
- `obj.StartFromCentroidOff ()` - When on, the initial alignment is to the centroid of the cohort curves. When off, the alignment is to the centroid of the first input. Default is off for backward compatibility.
- `vtkPointSet = obj.GetInput (int idx)` - Retrieve the input point set with index idx (usually only for pipeline tracing).

34.37 vtkProjectedTerrainPath

34.37.1 Usage

vtkProjectedTerrainPath projects an input polyline onto a terrain. (The terrain is defined by a 2D height image and is the second input to the filter.) The polyline projection is controlled via several modes as follows. 1) Simple mode projects the polyline points onto the terrain, taking into account the height offset instance variable. 2) Non-occluded mode insures that no parts of the polyline are occluded by the terrain (e.g. a line

passes through a mountain). This may require recursive subdivision of the polyline. 3) Hug mode insures that the polyine points remain within a constant distance from the surface. This may also require recursive subdivision of the polyline. Note that both non-occluded mode and hug mode also take into account the height offset, so it is possible to create paths that hug terrain a certain distance above it. To use this filter, define two inputs: 1) a polyline, and 2) an image whose scalar values represent a height field. Then specify the mode, and the height offset to use.

An description of the algorithm is as follows. The filter begins by projecting the polyline points to the image (offset by the specified height offset). If the mode is non-occluded or hug, then the maximum error along each line segment is computed and placed into a priority queue. Each line segment is then split at the point of maximum error, and the two new line segments are evaluated for maximum error. This process continues until the line is not occluded by the terrain (non-occluded mode) or satisfies the error on variation from the surface (hug mode). (Note this process is repeated for each polyline in the input. Also, the maximum error is computed in two parts: a maximum positive error and maximum negative error. If the polyline is above the terrain–i.e., the height offset is positive–in non-occluded or hug mode all negative errors are eliminated. If the polyline is below the terrain–i.e., the height offset is negative–in non-occluded or hug mode all positive errors are eliminated.)

To create an instance of class vtkProjectedTerrainPath, simply invoke its constructor as follows

```
obj = vtkProjectedTerrainPath
```

34.37.2 Methods

The class vtkProjectedTerrainPath has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProjectedTerrainPath class.

- `string = obj.GetClassName ()` - Standard methids for printing and determining type information.
- `int = obj.IsA (string name)` - Standard methids for printing and determining type information.
- `vtkProjectedTerrainPath = obj.NewInstance ()` - Standard methids for printing and determining type information.
- `vtkProjectedTerrainPath = obj.SafeDownCast (vtkObject o)` - Standard methids for printing and determining type information.
- `obj.SetSource (vtkImageData source)` - Specify the second input (the terrain) onto which the polyline(s) should be projected.
- `vtkImageData = obj.GetSource ()` - Specify the second input (the terrain) onto which the polyline(s) should be projected.
- `obj.SetProjectionMode (int )` - Determine how to control the projection process. Simple projection just projects the original polyline points. Non-occluded projection insures that the polyline does not intersect the terrain surface. Hug projection is similar to non-occulded projection except that produces a path that is nearly parallel to the terrain (within the user specified height tolerance).
- `int = obj.GetProjectionModeMinValue ()` - Determine how to control the projection process. Simple projection just projects the original polyline points. Non-occluded projection insures that the polyline does not intersect the terrain surface. Hug projection is similar to non-occulded projection except that produces a path that is nearly parallel to the terrain (within the user specified height tolerance).
- `int = obj.GetProjectionModeMaxValue ()` - Determine how to control the projection process. Simple projection just projects the original polyline points. Non-occluded projection insures that the polyline does not intersect the terrain surface. Hug projection is similar to non-occulded projection except that produces a path that is nearly parallel to the terrain (within the user specified height tolerance).

- `int = obj.GetProjectionMode ()` - Determine how to control the projection process. Simple projection just projects the original polyline points. Non-occluded projection insures that the polyline does not intersect the terrain surface. Hug projection is similar to non-occulded projection except that produces a path that is nearly parallel to the terrain (within the user specified height tolerance).

- `obj.SetProjectionModeToSimple ()` - Determine how to control the projection process. Simple projection just projects the original polyline points. Non-occluded projection insures that the polyline does not intersect the terrain surface. Hug projection is similar to non-occulded projection except that produces a path that is nearly parallel to the terrain (within the user specified height tolerance).

- `obj.SetProjectionModeToNonOccluded ()` - Determine how to control the projection process. Simple projection just projects the original polyline points. Non-occluded projection insures that the polyline does not intersect the terrain surface. Hug projection is similar to non-occulded projection except that produces a path that is nearly parallel to the terrain (within the user specified height tolerance).

- `obj.SetProjectionModeToHug ()` - This is the height above (or below) the terrain that the projected path should be. Positive values indicate distances above the terrain; negative values indicate distances below the terrain.

- `obj.SetHeightOffset (double )` - This is the height above (or below) the terrain that the projected path should be. Positive values indicate distances above the terrain; negative values indicate distances below the terrain.

- `double = obj.GetHeightOffset ()` - This is the height above (or below) the terrain that the projected path should be. Positive values indicate distances above the terrain; negative values indicate distances below the terrain.

- `obj.SetHeightTolerance (double )` - This is the allowable variation in the altitude of the path with respect to the variation in the terrain. It only comes into play if the hug projection mode is enabled.

- `double = obj.GetHeightToleranceMinValue ()` - This is the allowable variation in the altitude of the path with respect to the variation in the terrain. It only comes into play if the hug projection mode is enabled.

- `double = obj.GetHeightToleranceMaxValue ()` - This is the allowable variation in the altitude of the path with respect to the variation in the terrain. It only comes into play if the hug projection mode is enabled.

- `double = obj.GetHeightTolerance ()` - This is the allowable variation in the altitude of the path with respect to the variation in the terrain. It only comes into play if the hug projection mode is enabled.

- `obj.SetMaximumNumberOfLines (vtkIdType )` - This instance variable can be used to limit the total number of line segments created during subdivision. Note that the number of input line segments will be the minimum number that cab be output.

- `vtkIdType = obj.GetMaximumNumberOfLinesMinValue ()` - This instance variable can be used to limit the total number of line segments created during subdivision. Note that the number of input line segments will be the minimum number that cab be output.

- `vtkIdType = obj.GetMaximumNumberOfLinesMaxValue ()` - This instance variable can be used to limit the total number of line segments created during subdivision. Note that the number of input line segments will be the minimum number that cab be output.

- `vtkIdType = obj.GetMaximumNumberOfLines ()` - This instance variable can be used to limit the total number of line segments created during subdivision. Note that the number of input line segments will be the minimum number that cab be output.

34.38 vtkRenderLargeImage

34.38.1 Usage

vtkRenderLargeImage provides methods needed to read a region from a file.

To create an instance of class vtkRenderLargeImage, simply invoke its constructor as follows

```
obj = vtkRenderLargeImage
```

34.38.2 Methods

The class vtkRenderLargeImage has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderLargeImage class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderLargeImage = obj.NewInstance ()`
- `vtkRenderLargeImage = obj.SafeDownCast (vtkObject o)`
- `obj.SetMagnification (int )` - The magnification of the current render window
- `int = obj.GetMagnification ()` - The magnification of the current render window
- `obj.SetInput (vtkRenderer )` - Indicates what renderer to get the pixel data from.
- `vtkRenderer = obj.GetInput ()` - Returns which renderer is being used as the source for the pixel data.
- `vtkImageData = obj.GetOutput ()` - Get the output data object for a port on this algorithm.

34.39 vtkRIBExporter

34.39.1 Usage

vtkRIBExporter is a concrete subclass of vtkExporter that writes a Renderman .RIB files. The input specifies a vtkRenderWindow. All visible actors and lights will be included in the rib file. The following file naming conventions apply: rib file - FilePrefix.rib image file created by RenderMan - FilePrefix.tif texture files - TexturePrefix_0xADDR_MTIME.tif This object does NOT generate an image file. The user must run either RenderMan or a RenderMan emulator like Blue Moon Ray Tracer (BMRT). vtk properties are convert to Renderman shaders as follows: Normal property, no texture map - plastic.sl Normal property with texture map - txtplastic.sl These two shaders must be compiled by the rendering package being used. vtkRIBExporter also supports custom shaders. The shaders are written using the Renderman Shading Language. See "The Renderman Companion", ISBN 0-201-50868, 1989 for details on writing shaders. vtkRIBProperty specifies the declarations and parameter settings for custom shaders. Tcl Example: generate a rib file for the current rendering. vtkRIBExporter myRIB myRIB SetInput $renWin myRIB SetFIlePrefix mine myRIB Write This will create a file mine.rib. After running this file through a Renderman renderer a file mine.tif will contain the rendered image.

To create an instance of class vtkRIBExporter, simply invoke its constructor as follows

```
obj = vtkRIBExporter
```

34.39.2 Methods

The class vtkRIBExporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRIBExporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRIBExporter = obj.NewInstance ()`
- `vtkRIBExporter = obj.SafeDownCast (vtkObject o)`
- `obj.SetSize (int , int )`
- `obj.SetSize (int  a[2])`
- `int = obj. GetSize ()`
- `obj.SetPixelSamples (int , int )`
- `obj.SetPixelSamples (int  a[2])`
- `int = obj. GetPixelSamples ()`
- `obj.SetFilePrefix (string )` - Specify the prefix of the files to write out. The resulting file names will have .RIB appended to them.
- `string = obj.GetFilePrefix ()` - Specify the prefix of the files to write out. The resulting file names will have .RIB appended to them.
- `obj.SetTexturePrefix (string )` - Specify the prefix of any generated texture files.
- `string = obj.GetTexturePrefix ()` - Specify the prefix of any generated texture files.
- `obj.SetBackground (int )` - Set/Get the background flag. Default is 0 (off). If set, the rib file will contain an image shader that will use the renderer window's background color. Normally, RenderMan does generate backgrounds. Backgrounds are composited into the scene with the tiffcomp program that comes with Pixar's RenderMan Toolkit. In fact, Pixar's Renderman will accept an image shader but only sets the alpha of the background. Images created this way will still have a black background but contain an alpha of 1 at all pixels and CANNOT be subsequently composited with other images using tiffcomp. However, other RenderMan compliant renderers like Blue Moon Ray Tracing (BMRT) do allow image shaders and properly set the background color. If this sounds too confusing, use the following rules: If you are using Pixar's Renderman, leave the Background off. Otherwise, try setting BackGroundOn and see if you get the desired results.
- `int = obj.GetBackground ()` - Set/Get the background flag. Default is 0 (off). If set, the rib file will contain an image shader that will use the renderer window's background color. Normally, RenderMan does generate backgrounds. Backgrounds are composited into the scene with the tiffcomp program that comes with Pixar's RenderMan Toolkit. In fact, Pixar's Renderman will accept an image shader but only sets the alpha of the background. Images created this way will still have a black background but contain an alpha of 1 at all pixels and CANNOT be subsequently composited with other images using tiffcomp. However, other RenderMan compliant renderers like Blue Moon Ray Tracing (BMRT) do allow image shaders and properly set the background color. If this sounds too confusing, use the following rules: If you are using Pixar's Renderman, leave the Background off. Otherwise, try setting BackGroundOn and see if you get the desired results.

- `obj.BackgroundOn ()` - Set/Get the background flag. Default is 0 (off). If set, the rib file will contain an image shader that will use the renderer window's background color. Normally, RenderMan does generate backgrounds. Backgrounds are composited into the scene with the tiffcomp program that comes with Pixar's RenderMan Toolkit. In fact, Pixar's Renderman will accept an image shader but only sets the alpha of the background. Images created this way will still have a black background but contain an alpha of 1 at all pixels and CANNOT be subsequently composited with other images using tiffcomp. However, other RenderMan compliant renderers like Blue Moon Ray Tracing (BMRT) do allow image shaders and properly set the background color. If this sounds too confusing, use the following rules: If you are using Pixar's Renderman, leave the Background off. Otherwise, try setting BackGroundOn and see if you get the desired results.

- `obj.BackgroundOff ()` - Set/Get the background flag. Default is 0 (off). If set, the rib file will contain an image shader that will use the renderer window's background color. Normally, RenderMan does generate backgrounds. Backgrounds are composited into the scene with the tiffcomp program that comes with Pixar's RenderMan Toolkit. In fact, Pixar's Renderman will accept an image shader but only sets the alpha of the background. Images created this way will still have a black background but contain an alpha of 1 at all pixels and CANNOT be subsequently composited with other images using tiffcomp. However, other RenderMan compliant renderers like Blue Moon Ray Tracing (BMRT) do allow image shaders and properly set the background color. If this sounds too confusing, use the following rules: If you are using Pixar's Renderman, leave the Background off. Otherwise, try setting BackGroundOn and see if you get the desired results.

- `obj.SetExportArrays (int )` - Set or get the ExportArrays. If ExportArrays is set, then all point data, field data, and cell data arrays will get exported together with polygons.

- `int = obj.GetExportArraysMinValue ()` - Set or get the ExportArrays. If ExportArrays is set, then all point data, field data, and cell data arrays will get exported together with polygons.

- `int = obj.GetExportArraysMaxValue ()` - Set or get the ExportArrays. If ExportArrays is set, then all point data, field data, and cell data arrays will get exported together with polygons.

- `obj.ExportArraysOn ()` - Set or get the ExportArrays. If ExportArrays is set, then all point data, field data, and cell data arrays will get exported together with polygons.

- `obj.ExportArraysOff ()` - Set or get the ExportArrays. If ExportArrays is set, then all point data, field data, and cell data arrays will get exported together with polygons.

- `int = obj.GetExportArrays ()` - Set or get the ExportArrays. If ExportArrays is set, then all point data, field data, and cell data arrays will get exported together with polygons.

34.40 vtkRIBLight

34.40.1 Usage

vtkRIBLight is a subclass of vtkLight that allows the user to specify light source shaders and shadow casting lights for use with RenderMan.

To create an instance of class vtkRIBLight, simply invoke its constructor as follows

```
obj = vtkRIBLight
```

34.40.2 Methods

The class vtkRIBLight has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRIBLight class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRIBLight = obj.NewInstance ()`
- `vtkRIBLight = obj.SafeDownCast (vtkObject o)`
- `obj.ShadowsOn ()`
- `obj.ShadowsOff ()`
- `obj.SetShadows (int )`
- `int = obj.GetShadows ()`
- `obj.Render (vtkRenderer ren, int index)`

34.41 vtkRIBProperty

34.41.1 Usage

vtkRIBProperty is a subclass of vtkProperty that allows the user to specify named shaders for use with RenderMan. Both a surface shader and displacement shader can be specified. Parameters for the shaders can be declared and set.

To create an instance of class vtkRIBProperty, simply invoke its constructor as follows

```
obj = vtkRIBProperty
```

34.41.2 Methods

The class vtkRIBProperty has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRIBProperty class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRIBProperty = obj.NewInstance ()`
- `vtkRIBProperty = obj.SafeDownCast (vtkObject o)`
- `obj.SetSurfaceShader (string )` - Specify the name of a surface shader.
- `string = obj.GetSurfaceShader ()` - Specify the name of a surface shader.
- `obj.SetDisplacementShader (string )` - Specify the name of a displacement shader.
- `string = obj.GetDisplacementShader ()` - Specify the name of a displacement shader.
- `obj.SetVariable (string variable, string declaration)` - Specify declarations for variables..
- `obj.AddVariable (string variable, string declaration)` - Specify declarations for variables..
- `string = obj.GetDeclarations ()` - Get variable declarations
- `obj.SetParameter (string parameter, string value)` - Specify parameter values for variables.
- `obj.AddParameter (string parameter, string value)` - Specify parameter values for variables.
- `string = obj.GetParameters ()` - Get parameters.

34.42 vtkSpiderPlotActor

34.42.1 Usage

vtkSpiderPlotActor generates a spider plot from an input field (i.e., vtkDataObject). A spider plot represents N-dimensional data by using a set of N axes that originate from the center of a circle, and form the spokes of a wheel (like a spider web). Each N-dimensional point is plotted as a polyline that forms a closed polygon; the vertices of the polygon are plotted against the radial axes.

To use this class, you must specify an input data object. You'll probably also want to specify the position of the plot be setting the Position and Position2 instance variables, which define a rectangle in which the plot lies. Another important parameter is the IndependentVariables ivar, which tells the instance how to interpret the field data (independent variables as the rows or columns of the field). There are also many other instance variables that control the look of the plot includes its title and legend.

Set the text property/attributes of the title and the labels through the vtkTextProperty objects associated with these components.

To create an instance of class vtkSpiderPlotActor, simply invoke its constructor as follows

```
obj = vtkSpiderPlotActor
```

34.42.2 Methods

The class vtkSpiderPlotActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSpiderPlotActor class.

- `string = obj.GetClassName ()` - Standard methods for type information and printing.
- `int = obj.IsA (string name)` - Standard methods for type information and printing.
- `vtkSpiderPlotActor = obj.NewInstance ()` - Standard methods for type information and printing.
- `vtkSpiderPlotActor = obj.SafeDownCast (vtkObject o)` - Standard methods for type information and printing.
- `obj.SetInput (vtkDataObject )` - Set the input to the spider plot actor.
- `vtkDataObject = obj.GetInput ()` - Get the input data object to this actor.
- `obj.SetIndependentVariables (int )` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `int = obj.GetIndependentVariablesMinValue ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `int = obj.GetIndependentVariablesMaxValue ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `int = obj.GetIndependentVariables ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `obj.SetIndependentVariablesToColumns ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.

- `obj.SetIndependentVariablesToRows ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `obj.SetTitleVisibility (int )` - Enable/Disable the display of a plot title.
- `int = obj.GetTitleVisibility ()` - Enable/Disable the display of a plot title.
- `obj.TitleVisibilityOn ()` - Enable/Disable the display of a plot title.
- `obj.TitleVisibilityOff ()` - Enable/Disable the display of a plot title.
- `obj.SetTitle (string )` - Set/Get the title of the spider plot.
- `string = obj.GetTitle ()` - Set/Get the title of the spider plot.
- `obj.SetTitleTextProperty (vtkTextProperty p)` - Set/Get the title text property.
- `vtkTextProperty = obj.GetTitleTextProperty ()` - Set/Get the title text property.
- `obj.SetLabelVisibility (int )`
- `int = obj.GetLabelVisibility ()`
- `obj.LabelVisibilityOn ()`
- `obj.LabelVisibilityOff ()`
- `obj.SetLabelTextProperty (vtkTextProperty p)` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.SetNumberOfRings (int )` - Specify the number of circumferential rings. If set to zero, then none will be shown; othewise the specified number will be shown.
- `int = obj.GetNumberOfRingsMinValue ()` - Specify the number of circumferential rings. If set to zero, then none will be shown; othewise the specified number will be shown.
- `int = obj.GetNumberOfRingsMaxValue ()` - Specify the number of circumferential rings. If set to zero, then none will be shown; othewise the specified number will be shown.
- `int = obj.GetNumberOfRings ()` - Specify the number of circumferential rings. If set to zero, then none will be shown; othewise the specified number will be shown.
- `obj.SetAxisLabel (int i, string )` - Specify the names of the radial spokes (i.e., the radial axes). If not specified, then an integer number is automatically generated.
- `string = obj.GetAxisLabel (int i)` - Specify the names of the radial spokes (i.e., the radial axes). If not specified, then an integer number is automatically generated.
- `obj.SetAxisRange (int i, double min, double max)` - Specify the range of data on each radial axis. If not specified, then the range is computed automatically.
- `obj.SetAxisRange (int i, double range[2])` - Specify the range of data on each radial axis. If not specified, then the range is computed automatically.
- `obj.GetAxisRange (int i, double range[2])` - Specify the range of data on each radial axis. If not specified, then the range is computed automatically.
- `obj.SetPlotColor (int i, double r, double g, double b)` - Specify colors for each plot. If not specified, they are automatically generated.

- `obj.SetPlotColor (int i, double color[3])` - Specify colors for each plot. If not specified, they are automatically generated.
- `obj.SetLegendVisibility (int )` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `int = obj.GetLegendVisibility ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.LegendVisibilityOn ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.LegendVisibilityOff ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `vtkLegendBoxActor = obj.GetLegendActor ()` - Retrieve handles to the legend box. This is useful if you would like to manually control the legend appearance.
- `int = obj.RenderOverlay (vtkViewport )` - Draw the spider plot.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Draw the spider plot.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Does this prop have some translucent polygonal geometry?
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.

34.43 vtkStructuredExtent

34.43.1 Usage

vtkStructuredExtent is an helper class that helps in arithmetic with structured extents. It defines a bunch of static methods (most of which are inlined) to aid in dealing with extents.

To create an instance of class vtkStructuredExtent, simply invoke its constructor as follows

```
obj = vtkStructuredExtent
```

34.43.2 Methods

The class vtkStructuredExtent has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredExtent class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredExtent = obj.NewInstance ()`
- `vtkStructuredExtent = obj.SafeDownCast (vtkObject o)`

34.44 vtkTemporalDataSetCache

34.44.1 Usage

vtkTemporalDataSetCache cache time step requests of a temporal dataset, when cached data is requested it is returned using a shallow copy. .SECTION Thanks Ken Martin (Kitware) and John Bidiscombe of CSCS - Swiss National Supercomputing Centre for creating and contributing this class. For related material, please refer to : John Biddiscombe, Berk Geveci, Ken Martin, Kenneth Moreland, David Thompson, "Time Dependent Processing in a Parallel Pipeline Architecture", IEEE Visualization 2007.

To create an instance of class vtkTemporalDataSetCache, simply invoke its constructor as follows

```
obj = vtkTemporalDataSetCache
```

34.44.2 Methods

The class vtkTemporalDataSetCache has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalDataSetCache class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTemporalDataSetCache = obj.NewInstance ()`
- `vtkTemporalDataSetCache = obj.SafeDownCast (vtkObject o)`
- `obj.SetCacheSize (int size)` - This is the maximum number of time steps that can be retained in memory. it defaults to 10.
- `int = obj.GetCacheSize ()` - This is the maximum number of time steps that can be retained in memory. it defaults to 10.

34.45 vtkTemporalInterpolator

34.45.1 Usage

vtkTemporalInterpolator interpolates between two time steps to produce new data for an arbitrary T. vtkTemporalInterpolator has three modes of operation. The default mode is to produce a continuous range of time values as output, which enables a filter downstream to request any value of T within the range. The second mode of operation is enabled by setting DiscreteTimeStepInterval to a non zero value. When this mode is activated, the filter will report a finite number of Time steps separated by deltaT between the original range of values. This mode is useful when a dataset of N time steps has one (or more) missing datasets for certain T values and you simply wish to smooth over the missing steps but otherwise use the original data. The third mode of operation is enabled by setting ResampleFactor to a non zero positive integer value. When this mode is activated, the filter will report a finite number of Time steps which contain the original steps, plus N new values between each original step 1/ResampleFactor time units apart. Note that if the input time steps are irregular, then using ResampleFactor will produce an irregular sequence of regular steps between each of the original irregular steps (clear enough, yes?).

@TODO Higher order interpolation schemes will require changes to the API as most calls assume only two timesteps are used.

To create an instance of class vtkTemporalInterpolator, simply invoke its constructor as follows

```
obj = vtkTemporalInterpolator
```

34.45.2 Methods

The class vtkTemporalInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalInterpolator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTemporalInterpolator = obj.NewInstance ()`
- `vtkTemporalInterpolator = obj.SafeDownCast (vtkObject o)`
- `obj.SetDiscreteTimeStepInterval (double )` - If you require a discrete number of outputs steps, to be generated from an input source - for example, you required N steps separated by T, then set DiscreteTimeStepInterval to T and you will get TIME_RANGE/DiscreteTimeStepInterval steps This is a useful option to use if you have a dataset with one missing time step and wish to 'fill-in' the missing data with an interpolated value from the steps either side
- `double = obj.GetDiscreteTimeStepInterval ()` - If you require a discrete number of outputs steps, to be generated from an input source - for example, you required N steps separated by T, then set DiscreteTimeStepInterval to T and you will get TIME_RANGE/DiscreteTimeStepInterval steps This is a useful option to use if you have a dataset with one missing time step and wish to 'fill-in' the missing data with an interpolated value from the steps either side
- `obj.SetResampleFactor (int )` - When ResampleFactor is a non zero positive integer, each pair of input time steps will be interpolated between with the number of steps specified. For example an input of 1,2,3,4,5 and a resample factor of 10, will produce steps 0f 1.0, 1.1, 1.2.....1.9, 2.0 etc NB. Irregular input steps will produce irregular output steps. Resample factor wuill only be used if DiscreteTimeStepInterval is zero otherwise the DiscreteTimeStepInterval takes precedence
- `int = obj.GetResampleFactor ()` - When ResampleFactor is a non zero positive integer, each pair of input time steps will be interpolated between with the number of steps specified. For example an input of 1,2,3,4,5 and a resample factor of 10, will produce steps 0f 1.0, 1.1, 1.2.....1.9, 2.0 etc NB. Irregular input steps will produce irregular output steps. Resample factor wuill only be used if DiscreteTimeStepInterval is zero otherwise the DiscreteTimeStepInterval takes precedence

34.46 vtkTemporalShiftScale

34.46.1 Usage

vtkTemporalShiftScale modify the time range or time steps of the data without changing the data itself. The data is not resampled by this filter, only the information accompanying the data is modified.

To create an instance of class vtkTemporalShiftScale, simply invoke its constructor as follows

```
obj = vtkTemporalShiftScale
```

34.46.2 Methods

The class vtkTemporalShiftScale has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalShiftScale class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkTemporalShiftScale = obj.NewInstance ()`
- `vtkTemporalShiftScale = obj.SafeDownCast (vtkObject o)`
- `obj.SetPreShift (double )` - Apply a translation to the data before scaling. To convert T5,100 to T0,1 use Preshift=-5, Scale=1/95, PostShift=0 To convert T5,105 to T5,10 use Preshift=-5, Scale=5/100, PostShift=5
- `double = obj.GetPreShift ()` - Apply a translation to the data before scaling. To convert T5,100 to T0,1 use Preshift=-5, Scale=1/95, PostShift=0 To convert T5,105 to T5,10 use Preshift=-5, Scale=5/100, PostShift=5
- `obj.SetPostShift (double )` - Apply a translation to the time
- `double = obj.GetPostShift ()` - Apply a translation to the time
- `obj.SetScale (double )` - Apply a scale to the time.
- `double = obj.GetScale ()` - Apply a scale to the time.
- `obj.SetPeriodic (int )` - If Periodic is true, requests for time will be wrapped around so that the source appears to be a periodic time source. If data exists for times 0,N-1, setting periodic to true will cause time 0 to be produced when time N, 2N, 2N etc is requested. This effectively gives the source the ability to generate time data indefinitely in a loop. When combined with Shift/Scale, the time becomes periodic in the shifted and scaled time frame of reference. Note: Since the input time may not start at zero, the wrapping of time from the end of one period to the start of the next, will subtract the initial time - a source with T5..6 repeated periodicaly will have output time 5..6..7..8 etc.
- `int = obj.GetPeriodic ()` - If Periodic is true, requests for time will be wrapped around so that the source appears to be a periodic time source. If data exists for times 0,N-1, setting periodic to true will cause time 0 to be produced when time N, 2N, 2N etc is requested. This effectively gives the source the ability to generate time data indefinitely in a loop. When combined with Shift/Scale, the time becomes periodic in the shifted and scaled time frame of reference. Note: Since the input time may not start at zero, the wrapping of time from the end of one period to the start of the next, will subtract the initial time - a source with T5..6 repeated periodicaly will have output time 5..6..7..8 etc.
- `obj.PeriodicOn ()` - If Periodic is true, requests for time will be wrapped around so that the source appears to be a periodic time source. If data exists for times 0,N-1, setting periodic to true will cause time 0 to be produced when time N, 2N, 2N etc is requested. This effectively gives the source the ability to generate time data indefinitely in a loop. When combined with Shift/Scale, the time becomes periodic in the shifted and scaled time frame of reference. Note: Since the input time may not start at zero, the wrapping of time from the end of one period to the start of the next, will subtract the initial time - a source with T5..6 repeated periodicaly will have output time 5..6..7..8 etc.
- `obj.PeriodicOff ()` - If Periodic is true, requests for time will be wrapped around so that the source appears to be a periodic time source. If data exists for times 0,N-1, setting periodic to true will cause time 0 to be produced when time N, 2N, 2N etc is requested. This effectively gives the source the ability to generate time data indefinitely in a loop. When combined with Shift/Scale, the time becomes periodic in the shifted and scaled time frame of reference. Note: Since the input time may not start at zero, the wrapping of time from the end of one period to the start of the next, will subtract the initial time - a source with T5..6 repeated periodicaly will have output time 5..6..7..8 etc.
- `obj.SetPeriodicEndCorrection (int )` - if Periodic time is enabled, this flag determines if the last time step is the same as the first. If PeriodicEndCorrection is true, then it is assumed that the input data goes from 0-1 (or whatever scaled/shifted actual time) and time 1 is the same as time 0 so that steps will be 0,1,2,3...N,1,2,3...N,1,2,3 where step N is the same as 0 and step 0 is not repeated. When this flag is false the data is assumed to be literal and output is of the form 0,1,2,3...N,0,1,2,3... By default this flag is ON

- `int = obj.GetPeriodicEndCorrection ()` - if Periodic time is enabled, this flag determines if the last time step is the same as the first. If PeriodicEndCorrection is true, then it is assumed that the input data goes from 0-1 (or whatever scaled/shifted actual time) and time 1 is the same as time 0 so that steps will be 0,1,2,3...N,1,2,3...N,1,2,3 where step N is the same as 0 and step 0 is not repeated. When this flag is false the data is assumed to be literal and output is of the form 0,1,2,3...N,0,1,2,3... By default this flag is ON

- `obj.PeriodicEndCorrectionOn ()` - if Periodic time is enabled, this flag determines if the last time step is the same as the first. If PeriodicEndCorrection is true, then it is assumed that the input data goes from 0-1 (or whatever scaled/shifted actual time) and time 1 is the same as time 0 so that steps will be 0,1,2,3...N,1,2,3...N,1,2,3 where step N is the same as 0 and step 0 is not repeated. When this flag is false the data is assumed to be literal and output is of the form 0,1,2,3...N,0,1,2,3... By default this flag is ON

- `obj.PeriodicEndCorrectionOff ()` - if Periodic time is enabled, this flag determines if the last time step is the same as the first. If PeriodicEndCorrection is true, then it is assumed that the input data goes from 0-1 (or whatever scaled/shifted actual time) and time 1 is the same as time 0 so that steps will be 0,1,2,3...N,1,2,3...N,1,2,3 where step N is the same as 0 and step 0 is not repeated. When this flag is false the data is assumed to be literal and output is of the form 0,1,2,3...N,0,1,2,3... By default this flag is ON

- `obj.SetMaximumNumberOfPeriods (double )` - if Periodic time is enabled, this controls how many time periods time is reported for. A filter cannot output an infinite number of time steps and therefore a finite number of periods is generated when reporting time.

- `double = obj.GetMaximumNumberOfPeriods ()` - if Periodic time is enabled, this controls how many time periods time is reported for. A filter cannot output an infinite number of time steps and therefore a finite number of periods is generated when reporting time.

34.47 vtkTemporalSnapToTimeStep

34.47.1 Usage

vtkTemporalSnapToTimeStep modify the time range or time steps of the data without changing the data itself. The data is not resampled by this filter, only the information accompanying the data is modified.

To create an instance of class vtkTemporalSnapToTimeStep, simply invoke its constructor as follows

```
obj = vtkTemporalSnapToTimeStep
```

34.47.2 Methods

The class vtkTemporalSnapToTimeStep has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalSnapToTimeStep class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkTemporalSnapToTimeStep = obj.NewInstance ()`

- `vtkTemporalSnapToTimeStep = obj.SafeDownCast (vtkObject o)`

- `obj.SetSnapMode (int )`

- `int = obj.GetSnapMode ()`

- `obj.SetSnapModeToNearest ()`
- `obj.SetSnapModeToNextBelowOrEqual ()`
- `obj.SetSnapModeToNextAboveOrEqual ()`

34.48 vtkThinPlateSplineTransform

34.48.1 Usage

vtkThinPlateSplineTransform describes a nonlinear warp transform defined by a set of source and target landmarks. Any point on the mesh close to a source landmark will be moved to a place close to the corresponding target landmark. The points in between are interpolated smoothly using Bookstein's Thin Plate Spline algorithm.

To obtain a correct TPS warp, use the R2LogR kernel if your data is 2D, and the R kernel if your data is 3D. Or you can specify your own RBF. (Hence this class is more general than a pure TPS transform.)

To create an instance of class vtkThinPlateSplineTransform, simply invoke its constructor as follows

```
obj = vtkThinPlateSplineTransform
```

34.48.2 Methods

The class vtkThinPlateSplineTransform has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkThinPlateSplineTransform class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkThinPlateSplineTransform = obj.NewInstance ()`
- `vtkThinPlateSplineTransform = obj.SafeDownCast (vtkObject o)`
- `double = obj.GetSigma ()` - Specify the 'stiffness' of the spline. The default is 1.0.
- `obj.SetSigma (double )` - Specify the 'stiffness' of the spline. The default is 1.0.
- `obj.SetBasis (int basis)` - Specify the radial basis function to use. The default is R2LogR which is appropriate for 2D. Use —R— (SetBasisToR) if your data is 3D. Alternatively specify your own basis function, however this will mean that the transform will no longer be a true thin-plate spline.
- `int = obj.GetBasis ()` - Specify the radial basis function to use. The default is R2LogR which is appropriate for 2D. Use —R— (SetBasisToR) if your data is 3D. Alternatively specify your own basis function, however this will mean that the transform will no longer be a true thin-plate spline.
- `obj.SetBasisToR ()` - Specify the radial basis function to use. The default is R2LogR which is appropriate for 2D. Use —R— (SetBasisToR) if your data is 3D. Alternatively specify your own basis function, however this will mean that the transform will no longer be a true thin-plate spline.
- `obj.SetBasisToR2LogR ()` - Specify the radial basis function to use. The default is R2LogR which is appropriate for 2D. Use —R— (SetBasisToR) if your data is 3D. Alternatively specify your own basis function, however this will mean that the transform will no longer be a true thin-plate spline.
- `string = obj.GetBasisAsString ()` - Specify the radial basis function to use. The default is R2LogR which is appropriate for 2D. Use —R— (SetBasisToR) if your data is 3D. Alternatively specify your own basis function, however this will mean that the transform will no longer be a true thin-plate spline.

- `obj.SetSourceLandmarks (vtkPoints source)` - Set the source landmarks for the warp. If you add or change the vtkPoints object, you must call Modified() on it or the transformation might not update.
- `vtkPoints = obj.GetSourceLandmarks ()` - Set the source landmarks for the warp. If you add or change the vtkPoints object, you must call Modified() on it or the transformation might not update.
- `obj.SetTargetLandmarks (vtkPoints target)` - Set the target landmarks for the warp. If you add or change the vtkPoints object, you must call Modified() on it or the transformation might not update.
- `vtkPoints = obj.GetTargetLandmarks ()` - Set the target landmarks for the warp. If you add or change the vtkPoints object, you must call Modified() on it or the transformation might not update.
- `long = obj.GetMTime ()` - Get the MTime.
- `vtkAbstractTransform = obj.MakeTransform ()` - Make another transform of the same type.

34.49 vtkTransformToGrid

34.49.1 Usage

vtkTransformToGrid takes any transform as input and produces a grid for use by a vtkGridTransform. This can be used, for example, to invert a grid transform, concatenate two grid transforms, or to convert a thin plate spline transform into a grid transform.

To create an instance of class vtkTransformToGrid, simply invoke its constructor as follows

```
obj = vtkTransformToGrid
```

34.49.2 Methods

The class vtkTransformToGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransformToGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransformToGrid = obj.NewInstance ()`
- `vtkTransformToGrid = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkAbstractTransform )` - Set/Get the transform which will be converted into a grid.
- `vtkAbstractTransform = obj.GetInput ()` - Set/Get the transform which will be converted into a grid.
- `obj.SetGridExtent (int , int , int , int , int , int )` - Get/Set the extent of the grid.
- `obj.SetGridExtent (int  a[6])` - Get/Set the extent of the grid.
- `int = obj. GetGridExtent ()` - Get/Set the extent of the grid.
- `obj.SetGridOrigin (double , double , double )` - Get/Set the origin of the grid.
- `obj.SetGridOrigin (double  a[3])` - Get/Set the origin of the grid.
- `double = obj. GetGridOrigin ()` - Get/Set the origin of the grid.

- `obj.SetGridSpacing (double , double , double )` - Get/Set the spacing between samples in the grid.
- `obj.SetGridSpacing (double  a[3])` - Get/Set the spacing between samples in the grid.
- `double = obj. GetGridSpacing ()` - Get/Set the spacing between samples in the grid.
- `obj.SetGridScalarType (int )` - Get/Set the scalar type of the grid. The default is double.
- `int = obj.GetGridScalarType ()` - Get/Set the scalar type of the grid. The default is double.
- `obj.SetGridScalarTypeToFloat ()` - Get/Set the scalar type of the grid. The default is double.
- `obj.SetGridScalarTypeToShort ()` - Get/Set the scalar type of the grid. The default is double.
- `obj.SetGridScalarTypeToUnsignedShort ()` - Get/Set the scalar type of the grid. The default is double.
- `obj.SetGridScalarTypeToUnsignedChar ()` - Get/Set the scalar type of the grid. The default is double.
- `obj.SetGridScalarTypeToChar ()` - Get/Set the scalar type of the grid. The default is double.
- `double = obj.GetDisplacementScale ()` - Get the scale and shift to convert integer grid elements into real values: dx = scale*di + shift. If the grid is of double type, then scale = 1 and shift = 0.
- `double = obj.GetDisplacementShift ()` - Get the scale and shift to convert integer grid elements into real values: dx = scale*di + shift. If the grid is of double type, then scale = 1 and shift = 0.
- `vtkImageData = obj.GetOutput ()` - Get the output data object for a port on this algorithm.

34.50 vtkVectorText

34.50.1 Usage

To create an instance of class vtkVectorText, simply invoke its constructor as follows

```
obj = vtkVectorText
```

34.50.2 Methods

The class vtkVectorText has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVectorText class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVectorText = obj.NewInstance ()`
- `vtkVectorText = obj.SafeDownCast (vtkObject o)`
- `obj.SetText (string )` - Set/Get the text to be drawn.
- `string = obj.GetText ()` - Set/Get the text to be drawn.

34.51 vtkVideoSource

34.51.1 Usage

vtkVideoSource is a superclass for video input interfaces for VTK. The goal is to provide an interface which is very similar to the interface of a VCR, where the 'tape' is an internal frame buffer capable of holding a preset number of video frames. Specialized versions of this class record input from various video input sources. This base class records input from a noise source.

To create an instance of class vtkVideoSource, simply invoke its constructor as follows

```
obj = vtkVideoSource
```

34.51.2 Methods

The class vtkVideoSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVideoSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVideoSource = obj.NewInstance ()`
- `vtkVideoSource = obj.SafeDownCast (vtkObject o)`
- `obj.Record ()` - Record incoming video at the specified FrameRate. The recording continues indefinitely until Stop() is called.
- `obj.Play ()` - Play through the 'tape' sequentially at the specified frame rate. If you have just finished Recoding, you should call Rewind() first.
- `obj.Stop ()` - Stop recording or playing.
- `obj.Rewind ()` - Rewind to the frame with the earliest timestamp. Record operations will start on the following frame, therefore if you want to re-record over this frame you must call Seek(-1) before calling Grab() or Record().
- `obj.FastForward ()` - FastForward to the last frame that was recorded (i.e. to the frame that has the most recent timestamp).
- `obj.Seek (int n)` - Seek forwards or backwards by the specified number of frames (positive is forward, negative is backward).
- `obj.Grab ()` - Grab a single video frame.
- `int = obj.GetRecording ()` - Are we in record mode? (record mode and play mode are mutually exclusive).
- `int = obj.GetPlaying ()` - Are we in play mode? (record mode and play mode are mutually exclusive).
- `obj.SetFrameSize (int x, int y, int z)` - Set the full-frame size. This must be an allowed size for the device, the device may either refuse a request for an illegal frame size or automatically choose a new frame size. The default is usually 320x240x1, but can be device specific. The 'depth' should always be 1 (unless you have a device that can handle 3D acquisition).

- `obj.SetFrameSize (int dim[3])` - Set the full-frame size. This must be an allowed size for the device, the device may either refuse a request for an illegal frame size or automatically choose a new frame size. The default is usually 320x240x1, but can be device specific. The 'depth' should always be 1 (unless you have a device that can handle 3D acquisition).

- `int = obj. GetFrameSize ()` - Set the full-frame size. This must be an allowed size for the device, the device may either refuse a request for an illegal frame size or automatically choose a new frame size. The default is usually 320x240x1, but can be device specific. The 'depth' should always be 1 (unless you have a device that can handle 3D acquisition).

- `obj.SetFrameRate (float rate)` - Request a particular frame rate (default 30 frames per second).

- `float = obj.GetFrameRate ()` - Request a particular frame rate (default 30 frames per second).

- `obj.SetOutputFormat (int format)` - Set the output format. This must be appropriate for device, usually only VTK_LUMINANCE, VTK_RGB, and VTK_RGBA are supported.

- `obj.SetOutputFormatToLuminance ()` - Set the output format. This must be appropriate for device, usually only VTK_LUMINANCE, VTK_RGB, and VTK_RGBA are supported.

- `obj.SetOutputFormatToRGB ()` - Set the output format. This must be appropriate for device, usually only VTK_LUMINANCE, VTK_RGB, and VTK_RGBA are supported.

- `obj.SetOutputFormatToRGBA ()` - Set the output format. This must be appropriate for device, usually only VTK_LUMINANCE, VTK_RGB, and VTK_RGBA are supported.

- `int = obj.GetOutputFormat ()` - Set the output format. This must be appropriate for device, usually only VTK_LUMINANCE, VTK_RGB, and VTK_RGBA are supported.

- `obj.SetFrameBufferSize (int FrameBufferSize)` - Set size of the frame buffer, i.e. the number of frames that the 'tape' can store.

- `int = obj.GetFrameBufferSize ()` - Set size of the frame buffer, i.e. the number of frames that the 'tape' can store.

- `obj.SetNumberOfOutputFrames (int )` - Set the number of frames to copy to the output on each execute. The frames will be concatenated along the Z dimension, with the most recent frame first. Default: 1

- `int = obj.GetNumberOfOutputFrames ()` - Set the number of frames to copy to the output on each execute. The frames will be concatenated along the Z dimension, with the most recent frame first. Default: 1

- `obj.AutoAdvanceOn ()` - Set whether to automatically advance the buffer before each grab. Default: on

- `obj.AutoAdvanceOff ()` - Set whether to automatically advance the buffer before each grab. Default: on

- `obj.SetAutoAdvance (int )` - Set whether to automatically advance the buffer before each grab. Default: on

- `int = obj.GetAutoAdvance ()` - Set whether to automatically advance the buffer before each grab. Default: on

- `obj.SetClipRegion (int r[6])` - Set the clip rectangle for the frames. The video will be clipped before it is copied into the framebuffer. Changing the ClipRegion will destroy the current contents of the framebuffer. The default ClipRegion is (0,VTK_INT_MAX,0,VTK_INT_MAX,0,VTK_INT_MAX).

- `obj.SetClipRegion (int x0, int x1, int y0, int y1, int z0, int z1)` - Set the clip rectangle for the frames. The video will be clipped before it is copied into the framebuffer. Changing the ClipRegion will destroy the current contents of the framebuffer. The default ClipRegion is (0,VTK_INT_MAX,0,VTK_INT_MAX,0,VTK_INT_MAX).

- `int = obj. GetClipRegion ()` - Set the clip rectangle for the frames. The video will be clipped before it is copied into the framebuffer. Changing the ClipRegion will destroy the current contents of the framebuffer. The default ClipRegion is (0,VTK_INT_MAX,0,VTK_INT_MAX,0,VTK_INT_MAX).

- `obj.SetOutputWholeExtent (int , int , int , int , int , int )` - Get/Set the WholeExtent of the output. This can be used to either clip or pad the video frame. This clipping/padding is done when the frame is copied to the output, and does not change the contents of the framebuffer. This is useful e.g. for expanding the output size to a power of two for texture mapping. The default is (0,-1,0,-1,0,-1) which causes the entire frame to be copied to the output.

- `obj.SetOutputWholeExtent (int a[6])` - Get/Set the WholeExtent of the output. This can be used to either clip or pad the video frame. This clipping/padding is done when the frame is copied to the output, and does not change the contents of the framebuffer. This is useful e.g. for expanding the output size to a power of two for texture mapping. The default is (0,-1,0,-1,0,-1) which causes the entire frame to be copied to the output.

- `int = obj. GetOutputWholeExtent ()` - Get/Set the WholeExtent of the output. This can be used to either clip or pad the video frame. This clipping/padding is done when the frame is copied to the output, and does not change the contents of the framebuffer. This is useful e.g. for expanding the output size to a power of two for texture mapping. The default is (0,-1,0,-1,0,-1) which causes the entire frame to be copied to the output.

- `obj.SetDataSpacing (double , double , double )` - Set/Get the pixel spacing. Default: (1.0,1.0,1.0)

- `obj.SetDataSpacing (double a[3])` - Set/Get the pixel spacing. Default: (1.0,1.0,1.0)

- `double = obj. GetDataSpacing ()` - Set/Get the pixel spacing. Default: (1.0,1.0,1.0)

- `obj.SetDataOrigin (double , double , double )` - Set/Get the coordinates of the lower, left corner of the frame. Default: (0.0,0.0,0.0)

- `obj.SetDataOrigin (double a[3])` - Set/Get the coordinates of the lower, left corner of the frame. Default: (0.0,0.0,0.0)

- `double = obj. GetDataOrigin ()` - Set/Get the coordinates of the lower, left corner of the frame. Default: (0.0,0.0,0.0)

- `obj.SetOpacity (float )` - For RGBA output only (4 scalar components), set the opacity. This will not modify the existing contents of the framebuffer, only subsequently grabbed frames.

- `float = obj.GetOpacity ()` - For RGBA output only (4 scalar components), set the opacity. This will not modify the existing contents of the framebuffer, only subsequently grabbed frames.

- `int = obj.GetFrameCount ()` - This value is incremented each time a frame is grabbed. reset it to zero (or any other value) at any time.

- `obj.SetFrameCount (int )` - This value is incremented each time a frame is grabbed. reset it to zero (or any other value) at any time.

- `int = obj.GetFrameIndex ()` - Get the frame index relative to the 'beginning of the tape'. This value wraps back to zero if it increases past the FrameBufferSize.

- `double = obj.GetFrameTimeStamp (int frame)` - Get a time stamp in seconds (resolution of milliseconds) for a video frame. Time began on Jan 1, 1970. You can specify a number (negative or positive) to specify the position of the video frame relative to the current frame.

- `double = obj.GetFrameTimeStamp ()` - Get a time stamp in seconds (resolution of milliseconds) for the Output. Time began on Jan 1, 1970. This timestamp is only valid after the Output has been Updated.
- `obj.Initialize ()` - Initialize the hardware. This is called automatically on the first Update or Grab.
- `int = obj.GetInitialized ()` - Initialize the hardware. This is called automatically on the first Update or Grab.
- `obj.ReleaseSystemResources ()` - Release the video driver. This method must be called before application exit, or else the application might hang during exit.
- `obj.InternalGrab ()` - The internal function which actually does the grab. You will definitely want to override this if you develop a vtkVideoSource subclass.
- `obj.SetStartTimeStamp (double t)` - And internal variable which marks the beginning of a Record session. These methods are for internal use only.
- `double = obj.GetStartTimeStamp ()` - And internal variable which marks the beginning of a Record session. These methods are for internal use only.

34.52 vtkVRMLImporter

34.52.1 Usage

vtkVRMLImporter imports VRML 2.0 files into vtk.

To create an instance of class vtkVRMLImporter, simply invoke its constructor as follows

```
obj = vtkVRMLImporter
```

34.52.2 Methods

The class vtkVRMLImporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVRMLImporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVRMLImporter = obj.NewInstance ()`
- `vtkVRMLImporter = obj.SafeDownCast (vtkObject o)`
- `vtkObject = obj.GetVRMLDEFObject (string name)` - In the VRML spec you can DEF and USE nodes (name them), This routine will return the associated VTK object which was created as a result of the DEF mechanism Send in the name from the VRML file, get the VTK object. You will have to check and correctly cast the object since this only returns vtkObjects.
- `obj.enterNode (string )` - Needed by the yacc/lex grammar used
- `obj.exitNode ()` - Needed by the yacc/lex grammar used
- `obj.enterField (string )` - Needed by the yacc/lex grammar used
- `obj.exitField ()` - Needed by the yacc/lex grammar used
- `obj.useNode (string )` - Needed by the yacc/lex grammar used
- `obj.SetFileName (string )` - Specify the name of the file to read.
- `string = obj.GetFileName ()` - Specify the name of the file to read.

34.53 vtkWeightedTransformFilter

34.53.1 Usage

To create an instance of class vtkWeightedTransformFilter, simply invoke its constructor as follows

```
obj = vtkWeightedTransformFilter
```

34.53.2 Methods

The class vtkWeightedTransformFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWeightedTransformFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWeightedTransformFilter = obj.NewInstance ()`
- `vtkWeightedTransformFilter = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Return the MTime also considering the filter's transforms.
- `obj.SetWeightArray (string )` - WeightArray is the string name of the DataArray in the input's FieldData that holds the weighting coefficients for each point. The filter will first look for the array in the input's PointData FieldData. If the array isn't there, the filter looks in the input's FieldData. The WeightArray can have tuples of any length, but must have a tuple for every point in the input data set. This array transforms points, normals, and vectors.
- `string = obj.GetWeightArray ()` - WeightArray is the string name of the DataArray in the input's FieldData that holds the weighting coefficients for each point. The filter will first look for the array in the input's PointData FieldData. If the array isn't there, the filter looks in the input's FieldData. The WeightArray can have tuples of any length, but must have a tuple for every point in the input data set. This array transforms points, normals, and vectors.
- `obj.SetTransformIndexArray (string )` - TransformIndexArray is the string name of the DataArray in the input's FieldData that holds the indices for the transforms for each point. These indices are used to select which transforms each weight of the DataArray refers. If the TransformIndexArray is not specified, the weights of each point are assumed to map directly to a transform. This DataArray must be of type UnsignedShort, which effectively limits the number of transforms to 65536 if a transform index array is used.

  The filter will first look for the array in the input's PointData FieldData. If the array isn't there, the filter looks in the input's FieldData. The TransformIndexArray can have tuples of any length, but must have a tuple for every point in the input data set. This array transforms points, normals, and vectors.
- `string = obj.GetTransformIndexArray ()` - TransformIndexArray is the string name of the DataArray in the input's FieldData that holds the indices for the transforms for each point. These indices are used to select which transforms each weight of the DataArray refers. If the TransformIndexArray is not specified, the weights of each point are assumed to map directly to a transform. This DataArray must be of type UnsignedShort, which effectively limits the number of transforms to 65536 if a transform index array is used.

  The filter will first look for the array in the input's PointData FieldData. If the array isn't there, the filter looks in the input's FieldData. The TransformIndexArray can have tuples of any length, but must have a tuple for every point in the input data set. This array transforms points, normals, and vectors.

- `obj.SetCellDataWeightArray (string )` - The CellDataWeightArray is analogous to the WeightArray, except for CellData. The array is searched for first in the CellData FieldData, then in the input's FieldData. The data array must have a tuple for each cell. This array is used to transform only normals and vectors.
- `string = obj.GetCellDataWeightArray ()` - The CellDataWeightArray is analogous to the WeightArray, except for CellData. The array is searched for first in the CellData FieldData, then in the input's FieldData. The data array must have a tuple for each cell. This array is used to transform only normals and vectors.
- `obj.SetCellDataTransformIndexArray (string )`
- `string = obj.GetCellDataTransformIndexArray ()`
- `obj.SetTransform (vtkAbstractTransform transform, int num)` - Set or Get one of the filter's transforms. The transform number must be less than the number of transforms allocated for the object. Setting a transform slot to NULL is equivalent to assigning an overriding weight of zero to that filter slot.
- `vtkAbstractTransform = obj.GetTransform (int num)` - Set or Get one of the filter's transforms. The transform number must be less than the number of transforms allocated for the object. Setting a transform slot to NULL is equivalent to assigning an overriding weight of zero to that filter slot.
- `obj.SetNumberOfTransforms (int num)` - Set the number of transforms for the filter. References to non-existent filter numbers in the data array is equivalent to a weight of zero (i.e., no contribution of that filter or weight). The maximum number of transforms is limited to 65536 if transform index arrays are used.
- `int = obj.GetNumberOfTransforms ()` - Set the number of transforms for the filter. References to non-existent filter numbers in the data array is equivalent to a weight of zero (i.e., no contribution of that filter or weight). The maximum number of transforms is limited to 65536 if transform index arrays are used.
- `obj.AddInputValuesOn ()` - If AddInputValues is true, the output values of this filter will be offset from the input values. The effect is exactly equivalent to having an identity transform of weight 1 added into each output point.
- `obj.AddInputValuesOff ()` - If AddInputValues is true, the output values of this filter will be offset from the input values. The effect is exactly equivalent to having an identity transform of weight 1 added into each output point.
- `obj.SetAddInputValues (int )` - If AddInputValues is true, the output values of this filter will be offset from the input values. The effect is exactly equivalent to having an identity transform of weight 1 added into each output point.
- `int = obj.GetAddInputValues ()` - If AddInputValues is true, the output values of this filter will be offset from the input values. The effect is exactly equivalent to having an identity transform of weight 1 added into each output point.

34.54 vtkX3DExporter

34.54.1 Usage

vtkX3DExporter is a render window exporter which writes out the renderered scene into an X3D file. X3D is an XML-based format for representation 3D scenes (similar to VRML). Check out http://www.web3d.org/x3d/ for more details. .SECTION Thanks X3DExporter is contributed by Christophe Mouton at EDF.

To create an instance of class vtkX3DExporter, simply invoke its constructor as follows

```
obj = vtkX3DExporter
```

34.54.2 Methods

The class vtkX3DExporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkX3DExporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkX3DExporter = obj.NewInstance ()`
- `vtkX3DExporter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Set/Get the output file name.
- `string = obj.GetFileName ()` - Set/Get the output file name.
- `obj.SetSpeed (double )` - Specify the Speed of navigation. Default is 4.
- `double = obj.GetSpeed ()` - Specify the Speed of navigation. Default is 4.
- `obj.SetBinary (int )` - Turn on binary mode
- `int = obj.GetBinaryMinValue ()` - Turn on binary mode
- `int = obj.GetBinaryMaxValue ()` - Turn on binary mode
- `obj.BinaryOn ()` - Turn on binary mode
- `obj.BinaryOff ()` - Turn on binary mode
- `int = obj.GetBinary ()` - Turn on binary mode
- `obj.SetFastest (int )` - In binary mode use fastest instead of best compression
- `int = obj.GetFastestMinValue ()` - In binary mode use fastest instead of best compression
- `int = obj.GetFastestMaxValue ()` - In binary mode use fastest instead of best compression
- `obj.FastestOn ()` - In binary mode use fastest instead of best compression
- `obj.FastestOff ()` - In binary mode use fastest instead of best compression
- `int = obj.GetFastest ()` - In binary mode use fastest instead of best compression

34.55 vtkXYPlotActor

34.55.1 Usage

vtkXYPlotActor creates an x-y plot of data from one or more input data sets or field data. The class plots dataset scalar values (y-axis) against the points (x-axis). The x-axis values are generated by taking the point ids, computing a cumulative arc length, or a normalized arc length. More than one input data set can be specified to generate multiple plots. Alternatively, if field data is supplied as input, the class plots one component against another. (The user must specify which component to use as the x-axis and which for the y-axis.)

To use this class to plot dataset(s), you must specify one or more input datasets containing scalar and point data. You'll probably also want to invoke a method to control how the point coordinates are converted into x values (by default point ids are used).

To use this class to plot field data, you must specify one or more input data objects with its associated field data. You'll also want to specify which component to use as the x-axis and which to use as the y-axis. Note that when plotting field data, the x and y values are used directly (i.e., there are no options to normalize the components).

Once you've set up the plot, you'll want to position it. The PositionCoordinate defines the lower-left location of the x-y plot (specified in normalized viewport coordinates) and the Position2Coordinate define the upper-right corner. (Note: the Position2Coordinate is relative to PositionCoordinate, so you can move the vtkXYPlotActor around the viewport by setting just the PositionCoordinate.) The combination of the two position coordinates specifies a rectangle in which the plot will lie.

Optional features include the ability to specify axes labels, label format and plot title. You can also manually specify the x and y plot ranges (by default they are computed automatically). The Border instance variable is used to create space between the boundary of the plot window (specified by PositionCoordinate and Position2Coordinate) and the plot itself.

The font property of the plot title can be modified through the TitleTextProperty attribute. The font property of the axes titles and labels can be modified through the AxisTitleTextProperty and AxisLabelTextProperty attributes. You may also use the GetXAxisActor2D or GetYAxisActor2D methods to access each individual axis actor to modify their font properties. In the same way, the GetLegendBoxActor method can be used to access the legend box actor to modify its font properties.

There are several advanced features as well. You can assign per curve properties (such as color and a plot symbol). (Note that each input dataset and/or data object creates a single curve.) Another option is to add a plot legend that graphically indicates the correspondance between the curve, curve symbols, and the data source. You can also exchange the x and y axes if you prefer you plot orientation that way.

To create an instance of class vtkXYPlotActor, simply invoke its constructor as follows

```
obj = vtkXYPlotActor
```

34.55.2 Methods

The class vtkXYPlotActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXYPlotActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXYPlotActor = obj.NewInstance ()`
- `vtkXYPlotActor = obj.SafeDownCast (vtkObject o)`
- `obj.AddInput (vtkDataSet in, string arrayName, int component)` - Add a dataset to the list of data to append. The array name specifies which point array to plot. The array must be a vtkDataArray subclass, i.e. a numeric array. If the array name is NULL, then the default scalars are used. The array can have multiple components, but only the first component is ploted.
- `obj.AddInput (vtkDataSet in)` - Remove a dataset from the list of data to append.
- `obj.RemoveInput (vtkDataSet in, string arrayName, int component)` - Remove a dataset from the list of data to append.
- `obj.RemoveInput (vtkDataSet in)` - This removes all of the data set inputs, but does not change the data object inputs.
- `obj.RemoveAllInputs ()` - This removes all of the data set inputs, but does not change the data object inputs.
- `vtkDataSetCollection = obj.GetInputList ()` - If plotting points by value, which component to use to determine the value. This sets a value per each input dataset (i.e., the ith dataset).

- `obj.SetPointComponent (int i, int comp)` - If plotting points by value, which component to use to determine the value. This sets a value per each input dataset (i.e., the ith dataset).

- `int = obj.GetPointComponent (int i)` - If plotting points by value, which component to use to determine the value. This sets a value per each input dataset (i.e., the ith dataset).

- `obj.SetXValues (int )` - Specify how the independent (x) variable is computed from the points. The independent variable can be the scalar/point index (i.e., point id), the accumulated arc length along the points, the normalized arc length, or by component value. If plotting datasets (e.g., points), the value that is used is specified by the PointComponent ivar. (Note: these methods also control how field data is plotted. Field data is usually plotted by value or index, if plotting length 1-dimensional length measures are used.)

- `int = obj.GetXValuesMinValue ()` - Specify how the independent (x) variable is computed from the points. The independent variable can be the scalar/point index (i.e., point id), the accumulated arc length along the points, the normalized arc length, or by component value. If plotting datasets (e.g., points), the value that is used is specified by the PointComponent ivar. (Note: these methods also control how field data is plotted. Field data is usually plotted by value or index, if plotting length 1-dimensional length measures are used.)

- `int = obj.GetXValuesMaxValue ()` - Specify how the independent (x) variable is computed from the points. The independent variable can be the scalar/point index (i.e., point id), the accumulated arc length along the points, the normalized arc length, or by component value. If plotting datasets (e.g., points), the value that is used is specified by the PointComponent ivar. (Note: these methods also control how field data is plotted. Field data is usually plotted by value or index, if plotting length 1-dimensional length measures are used.)

- `int = obj.GetXValues ()` - Specify how the independent (x) variable is computed from the points. The independent variable can be the scalar/point index (i.e., point id), the accumulated arc length along the points, the normalized arc length, or by component value. If plotting datasets (e.g., points), the value that is used is specified by the PointComponent ivar. (Note: these methods also control how field data is plotted. Field data is usually plotted by value or index, if plotting length 1-dimensional length measures are used.)

- `obj.SetXValuesToIndex ()` - Specify how the independent (x) variable is computed from the points. The independent variable can be the scalar/point index (i.e., point id), the accumulated arc length along the points, the normalized arc length, or by component value. If plotting datasets (e.g., points), the value that is used is specified by the PointComponent ivar. (Note: these methods also control how field data is plotted. Field data is usually plotted by value or index, if plotting length 1-dimensional length measures are used.)

- `obj.SetXValuesToArcLength ()` - Specify how the independent (x) variable is computed from the points. The independent variable can be the scalar/point index (i.e., point id), the accumulated arc length along the points, the normalized arc length, or by component value. If plotting datasets (e.g., points), the value that is used is specified by the PointComponent ivar. (Note: these methods also control how field data is plotted. Field data is usually plotted by value or index, if plotting length 1-dimensional length measures are used.)

- `obj.SetXValuesToNormalizedArcLength ()` - Specify how the independent (x) variable is computed from the points. The independent variable can be the scalar/point index (i.e., point id), the accumulated arc length along the points, the normalized arc length, or by component value. If plotting datasets (e.g., points), the value that is used is specified by the PointComponent ivar. (Note: these methods also control how field data is plotted. Field data is usually plotted by value or index, if plotting length 1-dimensional length measures are used.)

- `obj.SetXValuesToValue ()` - Specify how the independent (x) variable is computed from the points. The independent variable can be the scalar/point index (i.e., point id), the accumulated arc length

along the points, the normalized arc length, or by component value. If plotting datasets (e.g., points), the value that is used is specified by the PointComponent ivar. (Note: these methods also control how field data is plotted. Field data is usually plotted by value or index, if plotting length 1-dimensional length measures are used.)

- `string = obj.GetXValuesAsString ()` - Specify how the independent (x) variable is computed from the points. The independent variable can be the scalar/point index (i.e., point id), the accumulated arc length along the points, the normalized arc length, or by component value. If plotting datasets (e.g., points), the value that is used is specified by the PointComponent ivar. (Note: these methods also control how field data is plotted. Field data is usually plotted by value or index, if plotting length 1-dimensional length measures are used.)
- `obj.AddDataObjectInput (vtkDataObject in)` - Add a dataset to the list of data to append.
- `obj.RemoveDataObjectInput (vtkDataObject in)` - Remove a dataset from the list of data to append.
- `vtkDataObjectCollection = obj.GetDataObjectInputList ()` - Indicate whether to plot rows or columns. If plotting rows, then the dependent variables is taken from a specified row, versus rows (y).
- `obj.SetDataObjectPlotMode (int )` - Indicate whether to plot rows or columns. If plotting rows, then the dependent variables is taken from a specified row, versus rows (y).
- `int = obj.GetDataObjectPlotModeMinValue ()` - Indicate whether to plot rows or columns. If plotting rows, then the dependent variables is taken from a specified row, versus rows (y).
- `int = obj.GetDataObjectPlotModeMaxValue ()` - Indicate whether to plot rows or columns. If plotting rows, then the dependent variables is taken from a specified row, versus rows (y).
- `int = obj.GetDataObjectPlotMode ()` - Indicate whether to plot rows or columns. If plotting rows, then the dependent variables is taken from a specified row, versus rows (y).
- `obj.SetDataObjectPlotModeToRows ()` - Indicate whether to plot rows or columns. If plotting rows, then the dependent variables is taken from a specified row, versus rows (y).
- `obj.SetDataObjectPlotModeToColumns ()` - Indicate whether to plot rows or columns. If plotting rows, then the dependent variables is taken from a specified row, versus rows (y).
- `string = obj.GetDataObjectPlotModeAsString ()` - Indicate whether to plot rows or columns. If plotting rows, then the dependent variables is taken from a specified row, versus rows (y).
- `obj.SetDataObjectXComponent (int i, int comp)` - Specify which component of the input data object to use as the independent variable for the ith input data object. (This ivar is ignored if plotting the index.) Note that the value is interpreted differently depending on DataObjectPlotMode. If the mode is Rows, then the value of DataObjectXComponent is the row number; otherwise it's the column number.
- `int = obj.GetDataObjectXComponent (int i)` - Specify which component of the input data object to use as the independent variable for the ith input data object. (This ivar is ignored if plotting the index.) Note that the value is interpreted differently depending on DataObjectPlotMode. If the mode is Rows, then the value of DataObjectXComponent is the row number; otherwise it's the column number.
- `obj.SetDataObjectYComponent (int i, int comp)` - Specify which component of the input data object to use as the dependent variable for the ith input data object. (This ivar is ignored if plotting the index.) Note that the value is interpreted differently depending on DataObjectPlotMode. If the mode is Rows, then the value of DataObjectYComponent is the row number; otherwise it's the column number.

- `int = obj.GetDataObjectYComponent (int i)` - Specify which component of the input data object to use as the dependent variable for the ith input data object. (This ivar is ignored if plotting the index.) Note that the value is interpreted differently depending on DataObjectPlotMode. If the mode is Rows, then the value of DataObjectYComponent is the row number; otherwise it's the column number.
- `obj.SetPlotColor (int i, double r, double g, double b)`
- `obj.SetPlotColor (int i, double color[3])`
- `double = obj.GetPlotColor (int i)`
- `obj.SetPlotSymbol (int i, vtkPolyData input)`
- `vtkPolyData = obj.GetPlotSymbol (int i)`
- `obj.SetPlotLabel (int i, string label)`
- `string = obj.GetPlotLabel (int i)`
- `int = obj.GetPlotCurvePoints ()`
- `obj.SetPlotCurvePoints (int )`
- `obj.PlotCurvePointsOn ()`
- `obj.PlotCurvePointsOff ()`
- `int = obj.GetPlotCurveLines ()`
- `obj.SetPlotCurveLines (int )`
- `obj.PlotCurveLinesOn ()`
- `obj.PlotCurveLinesOff ()`
- `obj.SetPlotLines (int i, int )`
- `int = obj.GetPlotLines (int i)`
- `obj.SetPlotPoints (int i, int )`
- `int = obj.GetPlotPoints (int i)`
- `obj.SetExchangeAxes (int )` - Enable/Disable exchange of the x-y axes (i.e., what was x becomes y, and vice-versa). Exchanging axes affects the labeling as well.
- `int = obj.GetExchangeAxes ()` - Enable/Disable exchange of the x-y axes (i.e., what was x becomes y, and vice-versa). Exchanging axes affects the labeling as well.
- `obj.ExchangeAxesOn ()` - Enable/Disable exchange of the x-y axes (i.e., what was x becomes y, and vice-versa). Exchanging axes affects the labeling as well.
- `obj.ExchangeAxesOff ()` - Enable/Disable exchange of the x-y axes (i.e., what was x becomes y, and vice-versa). Exchanging axes affects the labeling as well.
- `obj.SetReverseXAxis (int )` - Normally the x-axis is plotted from minimum to maximum. Setting this instance variable causes the x-axis to be plotted from maximum to minimum. Note that boolean always applies to the x-axis even if ExchangeAxes is set.
- `int = obj.GetReverseXAxis ()` - Normally the x-axis is plotted from minimum to maximum. Setting this instance variable causes the x-axis to be plotted from maximum to minimum. Note that boolean always applies to the x-axis even if ExchangeAxes is set.

- `obj.ReverseXAxisOn ()` - Normally the x-axis is plotted from minimum to maximum. Setting this instance variable causes the x-axis to be plotted from maximum to minimum. Note that boolean always applies to the x-axis even if ExchangeAxes is set.

- `obj.ReverseXAxisOff ()` - Normally the x-axis is plotted from minimum to maximum. Setting this instance variable causes the x-axis to be plotted from maximum to minimum. Note that boolean always applies to the x-axis even if ExchangeAxes is set.

- `obj.SetReverseYAxis (int )` - Normally the y-axis is plotted from minimum to maximum. Setting this instance variable causes the y-axis to be plotted from maximum to minimum. Note that boolean always applies to the y-axis even if ExchangeAxes is set.

- `int = obj.GetReverseYAxis ()` - Normally the y-axis is plotted from minimum to maximum. Setting this instance variable causes the y-axis to be plotted from maximum to minimum. Note that boolean always applies to the y-axis even if ExchangeAxes is set.

- `obj.ReverseYAxisOn ()` - Normally the y-axis is plotted from minimum to maximum. Setting this instance variable causes the y-axis to be plotted from maximum to minimum. Note that boolean always applies to the y-axis even if ExchangeAxes is set.

- `obj.ReverseYAxisOff ()` - Normally the y-axis is plotted from minimum to maximum. Setting this instance variable causes the y-axis to be plotted from maximum to minimum. Note that boolean always applies to the y-axis even if ExchangeAxes is set.

- `vtkLegendBoxActor = obj.GetLegendActor ()` - Retrieve handles to the legend box and glyph source. This is useful if you would like to change the default behavior of the legend box or glyph source. For example, the default glyph can be changed from a line to a vertex plus line, etc.)

- `vtkGlyphSource2D = obj.GetGlyphSource ()` - Retrieve handles to the legend box and glyph source. This is useful if you would like to change the default behavior of the legend box or glyph source. For example, the default glyph can be changed from a line to a vertex plus line, etc.)

- `obj.SetTitle (string )` - Set/Get the title of the x-y plot, and the title along the x and y axes.

- `string = obj.GetTitle ()` - Set/Get the title of the x-y plot, and the title along the x and y axes.

- `obj.SetXTitle (string )` - Set/Get the title of the x-y plot, and the title along the x and y axes.

- `string = obj.GetXTitle ()` - Set/Get the title of the x-y plot, and the title along the x and y axes.

- `obj.SetYTitle (string )` - Set/Get the title of the x-y plot, and the title along the x and y axes.

- `string = obj.GetYTitle ()` - Set/Get the title of the x-y plot, and the title along the x and y axes.

- `vtkAxisActor2D = obj.GetXAxisActor2D ()` - Retrieve handles to the X and Y axis (so that you can set their text properties for example)

- `vtkAxisActor2D = obj.GetYAxisActor2D ()` - Set the plot range (range of independent and dependent variables) to plot. Data outside of the range will be clipped. If the plot range of either the x or y variables is set to (v1,v2), where v1 == v2, then the range will be computed automatically. Note that the x-range values should be consistent with the way the independent variable is created (via INDEX, DISTANCE, or ARC_LENGTH).

- `obj.SetXRange (double , double )` - Set the plot range (range of independent and dependent variables) to plot. Data outside of the range will be clipped. If the plot range of either the x or y variables is set to (v1,v2), where v1 == v2, then the range will be computed automatically. Note that the x-range values should be consistent with the way the independent variable is created (via INDEX, DISTANCE, or ARC_LENGTH).

- `obj.SetXRange (double a[2])` - Set the plot range (range of independent and dependent variables) to plot. Data outside of the range will be clipped. If the plot range of either the x or y variables is set to (v1,v2), where v1 == v2, then the range will be computed automatically. Note that the x-range values should be consistent with the way the independent variable is created (via INDEX, DISTANCE, or ARC_LENGTH).

- `double = obj. GetXRange ()` - Set the plot range (range of independent and dependent variables) to plot. Data outside of the range will be clipped. If the plot range of either the x or y variables is set to (v1,v2), where v1 == v2, then the range will be computed automatically. Note that the x-range values should be consistent with the way the independent variable is created (via INDEX, DISTANCE, or ARC_LENGTH).

- `obj.SetYRange (double , double )` - Set the plot range (range of independent and dependent variables) to plot. Data outside of the range will be clipped. If the plot range of either the x or y variables is set to (v1,v2), where v1 == v2, then the range will be computed automatically. Note that the x-range values should be consistent with the way the independent variable is created (via INDEX, DISTANCE, or ARC_LENGTH).

- `obj.SetYRange (double a[2])` - Set the plot range (range of independent and dependent variables) to plot. Data outside of the range will be clipped. If the plot range of either the x or y variables is set to (v1,v2), where v1 == v2, then the range will be computed automatically. Note that the x-range values should be consistent with the way the independent variable is created (via INDEX, DISTANCE, or ARC_LENGTH).

- `double = obj. GetYRange ()` - Set the plot range (range of independent and dependent variables) to plot. Data outside of the range will be clipped. If the plot range of either the x or y variables is set to (v1,v2), where v1 == v2, then the range will be computed automatically. Note that the x-range values should be consistent with the way the independent variable is created (via INDEX, DISTANCE, or ARC_LENGTH).

- `obj.SetPlotRange (double xmin, double ymin, double xmax, double ymax)` - Set/Get the number of annotation labels to show along the x and y axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data. The convenience method SetNumberOfLables() sets the number of x and y labels to the same value.

- `obj.SetNumberOfXLabels (int )` - Set/Get the number of annotation labels to show along the x and y axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data. The convenience method SetNumberOfLables() sets the number of x and y labels to the same value.

- `int = obj.GetNumberOfXLabelsMinValue ()` - Set/Get the number of annotation labels to show along the x and y axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data. The convenience method SetNumberOfLables() sets the number of x and y labels to the same value.

- `int = obj.GetNumberOfXLabelsMaxValue ()` - Set/Get the number of annotation labels to show along the x and y axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data. The convenience method SetNumberOfLables() sets the number of x and y labels to the same value.

- `int = obj.GetNumberOfXLabels ()` - Set/Get the number of annotation labels to show along the x and y axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data. The convenience method SetNumberOfLables() sets the number of x and y labels to the same value.

- `obj.SetNumberOfYLabels (int )` - Set/Get the number of annotation labels to show along the x and y axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data. The convenience method SetNumberOfLables() sets the number of x and y labels to the same value.

- `int = obj.GetNumberOfYLabelsMinValue ()` - Set/Get the number of annotation labels to show along the x and y axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data. The convenience method SetNumberOfLables() sets the number of x and y labels to the same value.

- `int = obj.GetNumberOfYLabelsMaxValue ()` - Set/Get the number of annotation labels to show along the x and y axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data. The convenience method SetNumberOfLables() sets the number of x and y labels to the same value.

- `int = obj.GetNumberOfYLabels ()` - Set/Get the number of annotation labels to show along the x and y axes. This values is a suggestion: the number of labels may vary depending on the particulars of the data. The convenience method SetNumberOfLables() sets the number of x and y labels to the same value.

- `obj.SetNumberOfLabels (int num)` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.

- `obj.SetAdjustXLabels (int adjust)` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.

- `int = obj.GetAdjustXLabels ()` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.

- `obj.SetAdjustYLabels (int adjust)` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.

- `int = obj.GetAdjustYLabels ()` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.

- `obj.SetXTitlePosition (double position)` - Set/Get the position of the title of X or Y axis.

- `double = obj.GetXTitlePosition ()` - Set/Get the position of the title of X or Y axis.

- `obj.SetYTitlePosition (double position)` - Set/Get the position of the title of X or Y axis.

- `double = obj.GetYTitlePosition ()` - Set/Get the position of the title of X or Y axis.

- `obj.SetNumberOfXMinorTicks (int num)` - Set/Get the number of minor ticks in X or Y.

- `int = obj.GetNumberOfXMinorTicks ()` - Set/Get the number of minor ticks in X or Y.

- `obj.SetNumberOfYMinorTicks (int num)` - Set/Get the number of minor ticks in X or Y.

- `int = obj.GetNumberOfYMinorTicks ()` - Set/Get the number of minor ticks in X or Y.

- `obj.SetLegend (int )` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.

- `int = obj.GetLegend ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.LegendOn ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.LegendOff ()` - Enable/Disable the creation of a legend. If on, the legend labels will be created automatically unless the per plot legend symbol has been set.
- `obj.SetTitlePosition (double , double )` - Set/Get the position of the title. This has no effect if AdjustTitlePosition is true.
- `obj.SetTitlePosition (double a[2])` - Set/Get the position of the title. This has no effect if AdjustTitlePosition is true.
- `double = obj. GetTitlePosition ()` - Set/Get the position of the title. This has no effect if AdjustTitlePosition is true.
- `obj.SetAdjustTitlePosition (int )` - If true, the xyplot actor will adjust the position of the title automatically to be upper-middle. Default is true.
- `int = obj.GetAdjustTitlePosition ()` - If true, the xyplot actor will adjust the position of the title automatically to be upper-middle. Default is true.
- `obj.AdjustTitlePositionOn ()` - If true, the xyplot actor will adjust the position of the title automatically to be upper-middle. Default is true.
- `obj.AdjustTitlePositionOff ()` - If true, the xyplot actor will adjust the position of the title automatically to be upper-middle. Default is true.
- `obj.SetAdjustTitlePositionMode (int )` - If AdjustTitlePosition is truem, the xyplot actor will adjust the position of the title automatically depending on the given mode, the mode is a combination of the Alignment flags. by default: vtkXYPlotActor::AlignHCenter — vtkXYPlotActor::Top — vtkXYPlotActor::AlignAxisVCenter
- `int = obj.GetAdjustTitlePositionMode ()` - If AdjustTitlePosition is truem, the xyplot actor will adjust the position of the title automatically depending on the given mode, the mode is a combination of the Alignment flags. by default: vtkXYPlotActor::AlignHCenter — vtkXYPlotActor::Top — vtkXYPlotActor::AlignAxisVCenter
- `obj.SetLegendPosition (double , double )` - Use these methods to control the position of the legend. The variables LegendPosition and LegendPosition2 define the lower-left and upper-right position of the legend. The coordinates are expressed as normalized values with respect to the rectangle defined by PositionCoordinate and Position2Coordinate. Note that LegendPosition2 is relative to LegendPosition.
- `obj.SetLegendPosition (double a[2])` - Use these methods to control the position of the legend. The variables LegendPosition and LegendPosition2 define the lower-left and upper-right position of the legend. The coordinates are expressed as normalized values with respect to the rectangle defined by PositionCoordinate and Position2Coordinate. Note that LegendPosition2 is relative to LegendPosition.
- `double = obj. GetLegendPosition ()` - Use these methods to control the position of the legend. The variables LegendPosition and LegendPosition2 define the lower-left and upper-right position of the legend. The coordinates are expressed as normalized values with respect to the rectangle defined by PositionCoordinate and Position2Coordinate. Note that LegendPosition2 is relative to LegendPosition.
- `obj.SetLegendPosition2 (double , double )` - Use these methods to control the position of the legend. The variables LegendPosition and LegendPosition2 define the lower-left and upper-right position of the legend. The coordinates are expressed as normalized values with respect to the rectangle defined by PositionCoordinate and Position2Coordinate. Note that LegendPosition2 is relative to LegendPosition.

- `obj.SetLegendPosition2 (double a[2])` - Use these methods to control the position of the legend. The variables LegendPosition and LegendPosition2 define the lower-left and upper-right position of the legend. The coordinates are expressed as normalized values with respect to the rectangle defined by PositionCoordinate and Position2Coordinate. Note that LegendPosition2 is relative to LegendPosition.
- `double = obj. GetLegendPosition2 ()` - Use these methods to control the position of the legend. The variables LegendPosition and LegendPosition2 define the lower-left and upper-right position of the legend. The coordinates are expressed as normalized values with respect to the rectangle defined by PositionCoordinate and Position2Coordinate. Note that LegendPosition2 is relative to LegendPosition.
- `obj.SetTitleTextProperty (vtkTextProperty p)` - Set/Get the title text property.
- `vtkTextProperty = obj.GetTitleTextProperty ()` - Set/Get the title text property.
- `obj.SetAxisTitleTextProperty (vtkTextProperty p)` - Set/Get the title text property of all axes. Note that each axis can be controlled individually through the GetX/YAxisActor2D() methods.
- `vtkTextProperty = obj.GetAxisTitleTextProperty ()` - Set/Get the title text property of all axes. Note that each axis can be controlled individually through the GetX/YAxisActor2D() methods.
- `obj.SetAxisLabelTextProperty (vtkTextProperty p)` - Set/Get the labels text property of all axes. Note that each axis can be controlled individually through the GetX/YAxisActor2D() methods.
- `vtkTextProperty = obj.GetAxisLabelTextProperty ()` - Set/Get the labels text property of all axes. Note that each axis can be controlled individually through the GetX/YAxisActor2D() methods.
- `obj.SetLogx (int )` - Enable/Disable plotting of Log of x-values.
- `int = obj.GetLogx ()` - Enable/Disable plotting of Log of x-values.
- `obj.LogxOn ()` - Enable/Disable plotting of Log of x-values.
- `obj.LogxOff ()` - Enable/Disable plotting of Log of x-values.
- `obj.SetLabelFormat (string \_arg)` - Set/Get the format with which to print the labels . This sets both X and Y label formats. GetLabelFormat() returns X label format.
- `string = obj.GetLabelFormat ()` - Set/Get the format with which to print the X label.
- `obj.SetXLabelFormat (string \_arg)` - Set/Get the format with which to print the X label.
- `string = obj.GetXLabelFormat ()` - Set/Get the format with which to print the X label.
- `obj.SetYLabelFormat (string \_arg)` - Set/Get the format with which to print the Y label.
- `string = obj.GetYLabelFormat ()` - Set/Get the format with which to print the Y label.
- `obj.SetBorder (int )` - Set/Get the spacing between the plot window and the plot. The value is specified in pixels.
- `int = obj.GetBorderMinValue ()` - Set/Get the spacing between the plot window and the plot. The value is specified in pixels.
- `int = obj.GetBorderMaxValue ()` - Set/Get the spacing between the plot window and the plot. The value is specified in pixels.
- `int = obj.GetBorder ()` - Set/Get the spacing between the plot window and the plot. The value is specified in pixels.
- `int = obj.GetPlotPoints ()` - Set/Get whether the points are rendered. The point size can be set in the property object. This is a global flag which affects the plot only if per curve symbols are not defined.

- `obj.SetPlotPoints (int )` - Set/Get whether the points are rendered. The point size can be set in the property object. This is a global flag which affects the plot only if per curve symbols are not defined.
- `obj.PlotPointsOn ()` - Set/Get whether the points are rendered. The point size can be set in the property object. This is a global flag which affects the plot only if per curve symbols are not defined.
- `obj.PlotPointsOff ()` - Set/Get whether the points are rendered. The point size can be set in the property object. This is a global flag which affects the plot only if per curve symbols are not defined.
- `int = obj.GetPlotLines ()` - Set/Get whether the lines are rendered. The line width can be set in the property object.
- `obj.SetPlotLines (int )` - Set/Get whether the lines are rendered. The line width can be set in the property object.
- `obj.PlotLinesOn ()` - Set/Get whether the lines are rendered. The line width can be set in the property object.
- `obj.PlotLinesOff ()` - Set/Get whether the lines are rendered. The line width can be set in the property object.
- `obj.SetGlyphSize (double )` - Set/Get the factor that controls how big glyphs are in the plot. The number is expressed as a fraction of the length of the diagonal of the plot bounding box.
- `double = obj.GetGlyphSizeMinValue ()` - Set/Get the factor that controls how big glyphs are in the plot. The number is expressed as a fraction of the length of the diagonal of the plot bounding box.
- `double = obj.GetGlyphSizeMaxValue ()` - Set/Get the factor that controls how big glyphs are in the plot. The number is expressed as a fraction of the length of the diagonal of the plot bounding box.
- `double = obj.GetGlyphSize ()` - Set/Get the factor that controls how big glyphs are in the plot. The number is expressed as a fraction of the length of the diagonal of the plot bounding box.
- `obj.ViewportToPlotCoordinate (vtkViewport viewport)` - An alternate form of ViewportToPlotCoordinate() above. This method inputs the viewport coordinate pair (defined by the ivar ViewportCoordinate)and then stores them in the ivar PlotCoordinate.
- `obj.SetPlotCoordinate (double , double )` - An alternate form of ViewportToPlotCoordinate() above. This method inputs the viewport coordinate pair (defined by the ivar ViewportCoordinate)and then stores them in the ivar PlotCoordinate.
- `obj.SetPlotCoordinate (double  a[2])` - An alternate form of ViewportToPlotCoordinate() above. This method inputs the viewport coordinate pair (defined by the ivar ViewportCoordinate)and then stores them in the ivar PlotCoordinate.
- `double = obj. GetPlotCoordinate ()` - An alternate form of ViewportToPlotCoordinate() above. This method inputs the viewport coordinate pair (defined by the ivar ViewportCoordinate)and then stores them in the ivar PlotCoordinate.
- `obj.PlotToViewportCoordinate (vtkViewport viewport)` - An alternate form of PlotToViewportCoordinate() above. This method inputs the plot coordinate pair (defined in the ivar PlotCoordinate) and then stores them in the ivar ViewportCoordinate. (This method can be wrapped.)
- `obj.SetViewportCoordinate (double , double )` - An alternate form of PlotToViewportCoordinate() above. This method inputs the plot coordinate pair (defined in the ivar PlotCoordinate) and then stores them in the ivar ViewportCoordinate. (This method can be wrapped.)
- `obj.SetViewportCoordinate (double  a[2])` - An alternate form of PlotToViewportCoordinate() above. This method inputs the plot coordinate pair (defined in the ivar PlotCoordinate) and then stores them in the ivar ViewportCoordinate. (This method can be wrapped.)

- `double = obj. GetViewportCoordinate ()` - An alternate form of PlotToViewportCoordinate() above. This method inputs the plot coordinate pair (defined in the ivar PlotCoordinate) and then stores them in the ivar ViewportCoordinate. (This method can be wrapped.)
- `int = obj.IsInPlot (vtkViewport viewport, double u, double v)` - Is the specified viewport position within the plot area (as opposed to the region used by the plot plus the labels)?
- `obj.SetChartBox (int )` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the chart box.
- `int = obj.GetChartBox ()` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the chart box.
- `obj.ChartBoxOn ()` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the chart box.
- `obj.ChartBoxOff ()` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the chart box.
- `obj.SetChartBorder (int )` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the legend box.
- `int = obj.GetChartBorder ()` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the legend box.
- `obj.ChartBorderOn ()` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the legend box.
- `obj.ChartBorderOff ()` - Set/Get the flag that controls whether a box will be drawn/filled corresponding to the legend box.
- `vtkProperty2D = obj.GetChartBoxProperty ()` - Get the box vtkProperty2D.
- `obj.SetShowReferenceXLine (int )` - Set/Get if the X reference line is visible. hidden by default
- `int = obj.GetShowReferenceXLine ()` - Set/Get if the X reference line is visible. hidden by default
- `obj.ShowReferenceXLineOn ()` - Set/Get if the X reference line is visible. hidden by default
- `obj.ShowReferenceXLineOff ()` - Set/Get if the X reference line is visible. hidden by default
- `obj.SetReferenceXValue (double )`
- `double = obj.GetReferenceXValue ()`
- `obj.SetShowReferenceYLine (int )` - Set/Get if the Y reference line is visible. hidden by default
- `int = obj.GetShowReferenceYLine ()` - Set/Get if the Y reference line is visible. hidden by default
- `obj.ShowReferenceYLineOn ()` - Set/Get if the Y reference line is visible. hidden by default
- `obj.ShowReferenceYLineOff ()` - Set/Get if the Y reference line is visible. hidden by default
- `obj.SetReferenceYValue (double )`
- `double = obj.GetReferenceYValue ()`
- `long = obj.GetMTime ()` - Take into account the modified time of internal helper classes.

Chapter 35

Visualization Toolkit Imaging Classes

35.1 vtkBooleanTexture

35.1.1 Usage

vtkBooleanTexture is a filter to generate a 2D texture map based on combinations of inside, outside, and on region boundary. The "region" is implicitly represented via 2D texture coordinates. These texture coordinates are normally generated using a filter like vtkImplicitTextureCoords, which generates the texture coordinates for any implicit function.

vtkBooleanTexture generates the map according to the s-t texture coordinates plus the notion of being in, on, or outside of a region. An in region is when the texture coordinate is between (0,0.5-thickness/2). An out region is where the texture coordinate is (0.5+thickness/2). An on region is between (0.5-thickness/2,0.5+thickness/2). The combination in, on, and out for each of the s-t texture coordinates results in 16 possible combinations (see text). For each combination, a different value of intensity and transparency can be assigned. To assign maximum intensity and/or opacity use the value 255. A minimum value of 0 results in a black region (for intensity) and a fully transparent region (for transparency).

To create an instance of class vtkBooleanTexture, simply invoke its constructor as follows

```
obj = vtkBooleanTexture
```

35.1.2 Methods

The class vtkBooleanTexture has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBooleanTexture class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBooleanTexture = obj.NewInstance ()`
- `vtkBooleanTexture = obj.SafeDownCast (vtkObject o)`
- `obj.SetXSize (int )` - Set the X texture map dimension.
- `int = obj.GetXSize ()` - Set the X texture map dimension.
- `obj.SetYSize (int )` - Set the Y texture map dimension.
- `int = obj.GetYSize ()` - Set the Y texture map dimension.
- `obj.SetThickness (int )` - Set the thickness of the "on" region.

- `int = obj.GetThickness ()` - Set the thickness of the "on" region.
- `obj.SetInIn (char , char )` - Specify intensity/transparency for "in/in" region.
- `obj.SetInIn (char  a[2])` - Specify intensity/transparency for "in/in" region.
- `char = obj. GetInIn ()` - Specify intensity/transparency for "in/in" region.
- `obj.SetInOut (char , char )` - Specify intensity/transparency for "in/out" region.
- `obj.SetInOut (char  a[2])` - Specify intensity/transparency for "in/out" region.
- `char = obj. GetInOut ()` - Specify intensity/transparency for "in/out" region.
- `obj.SetOutIn (char , char )` - Specify intensity/transparency for "out/in" region.
- `obj.SetOutIn (char  a[2])` - Specify intensity/transparency for "out/in" region.
- `char = obj. GetOutIn ()` - Specify intensity/transparency for "out/in" region.
- `obj.SetOutOut (char , char )` - Specify intensity/transparency for "out/out" region.
- `obj.SetOutOut (char  a[2])` - Specify intensity/transparency for "out/out" region.
- `char = obj. GetOutOut ()` - Specify intensity/transparency for "out/out" region.
- `obj.SetOnOn (char , char )` - Specify intensity/transparency for "on/on" region.
- `obj.SetOnOn (char  a[2])` - Specify intensity/transparency for "on/on" region.
- `char = obj. GetOnOn ()` - Specify intensity/transparency for "on/on" region.
- `obj.SetOnIn (char , char )` - Specify intensity/transparency for "on/in" region.
- `obj.SetOnIn (char  a[2])` - Specify intensity/transparency for "on/in" region.
- `char = obj. GetOnIn ()` - Specify intensity/transparency for "on/in" region.
- `obj.SetOnOut (char , char )` - Specify intensity/transparency for "on/out" region.
- `obj.SetOnOut (char  a[2])` - Specify intensity/transparency for "on/out" region.
- `char = obj. GetOnOut ()` - Specify intensity/transparency for "on/out" region.
- `obj.SetInOn (char , char )` - Specify intensity/transparency for "in/on" region.
- `obj.SetInOn (char  a[2])` - Specify intensity/transparency for "in/on" region.
- `char = obj. GetInOn ()` - Specify intensity/transparency for "in/on" region.
- `obj.SetOutOn (char , char )` - Specify intensity/transparency for "out/on" region.
- `obj.SetOutOn (char  a[2])` - Specify intensity/transparency for "out/on" region.
- `char = obj. GetOutOn ()` - Specify intensity/transparency for "out/on" region.

35.2 vtkExtractVOI

35.2.1 Usage

vtkExtractVOI is a filter that selects a portion of an input structured points dataset, or subsamples an input dataset. (The selected portion of interested is referred to as the Volume Of Interest, or VOI.) The output of this filter is a structured points dataset. The filter treats input data of any topological dimension (i.e., point, line, image, or volume) and can generate output data of any topological dimension.

To use this filter set the VOI ivar which are i-j-k min/max indices that specify a rectangular region in the data. (Note that these are 0-offset.) You can also specify a sampling rate to subsample the data.

Typical applications of this filter are to extract a slice from a volume for image processing, subsampling large volumes to reduce data size, or extracting regions of a volume with interesting data.

To create an instance of class vtkExtractVOI, simply invoke its constructor as follows

```
obj = vtkExtractVOI
```

35.2.2 Methods

The class vtkExtractVOI has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractVOI class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractVOI = obj.NewInstance ()`
- `vtkExtractVOI = obj.SafeDownCast (vtkObject o)`
- `obj.SetVOI (int , int , int , int , int , int )` - Specify i-j-k (min,max) pairs to extract. The resulting structured points dataset can be of any topological dimension (i.e., point, line, image, or volume).
- `obj.SetVOI (int  a[6])` - Specify i-j-k (min,max) pairs to extract. The resulting structured points dataset can be of any topological dimension (i.e., point, line, image, or volume).
- `int = obj. GetVOI ()` - Specify i-j-k (min,max) pairs to extract. The resulting structured points dataset can be of any topological dimension (i.e., point, line, image, or volume).
- `obj.SetSampleRate (int , int , int )` - Set the sampling rate in the i, j, and k directions. If the rate is ¿ 1, then the resulting VOI will be subsampled representation of the input. For example, if the SampleRate=(2,2,2), every other point will be selected, resulting in a volume 1/8th the original size.
- `obj.SetSampleRate (int  a[3])` - Set the sampling rate in the i, j, and k directions. If the rate is ¿ 1, then the resulting VOI will be subsampled representation of the input. For example, if the SampleRate=(2,2,2), every other point will be selected, resulting in a volume 1/8th the original size.
- `int = obj. GetSampleRate ()` - Set the sampling rate in the i, j, and k directions. If the rate is ¿ 1, then the resulting VOI will be subsampled representation of the input. For example, if the SampleRate=(2,2,2), every other point will be selected, resulting in a volume 1/8th the original size.

35.3 vtkFastSplatter

35.3.1 Usage

vtkFastSplatter takes any vtkPointSet as input (of which vtkPolyData and vtkUnstructuredGrid inherit). Each point in the data set is considered to be an impulse. These impulses are convolved with a given splat image. In other words, the splat image is added to the final image at every place where there is an input point.

Note that point and cell data are thrown away. If you want a sampling of unstructured points consider vtkGaussianSplatter or vtkShepardMethod.

Use input port 0 for the impulse data (vtkPointSet), and input port 1 for the splat image (vtkImageData) .SECTION Bugs

Any point outside of the extents of the image is thrown away, even if it is close enough such that it's convolution with the splat image would overlap the extents.

To create an instance of class vtkFastSplatter, simply invoke its constructor as follows

```
obj = vtkFastSplatter
```

35.3.2 Methods

The class vtkFastSplatter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFastSplatter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFastSplatter = obj.NewInstance ()`
- `vtkFastSplatter = obj.SafeDownCast (vtkObject o)`
- `obj.SetModelBounds (double , double , double , double , double , double )` - Set / get the (xmin,xmax, ymin,ymax, zmin,zmax) bounding box in which the sampling is performed. If any of the (min,max) bounds values are min ¿= max, then the bounds will be computed automatically from the input data. Otherwise, the user-specified bounds will be used.
- `obj.SetModelBounds (double a[6])` - Set / get the (xmin,xmax, ymin,ymax, zmin,zmax) bounding box in which the sampling is performed. If any of the (min,max) bounds values are min ¿= max, then the bounds will be computed automatically from the input data. Otherwise, the user-specified bounds will be used.
- `double = obj. GetModelBounds ()` - Set / get the (xmin,xmax, ymin,ymax, zmin,zmax) bounding box in which the sampling is performed. If any of the (min,max) bounds values are min ¿= max, then the bounds will be computed automatically from the input data. Otherwise, the user-specified bounds will be used.
- `obj.SetOutputDimensions (int , int , int )` - Set/get the dimensions of the output image
- `obj.SetOutputDimensions (int a[3])` - Set/get the dimensions of the output image
- `int = obj. GetOutputDimensions ()` - Set/get the dimensions of the output image
- `obj.SetLimitMode (int )` - Set/get the way voxel values will be limited. If this is set to None (the default), the output can have arbitrarily large values. If set to clamp, the output will be clamped to [MinValue,MaxValue]. If set to scale, the output will be linearly scaled between MinValue and MaxValue.

- `int = obj.GetLimitMode ()` - Set/get the way voxel values will be limited. If this is set to None (the default), the output can have arbitrarily large values. If set to clamp, the output will be clamped to [MinValue,MaxValue]. If set to scale, the output will be linearly scaled between MinValue and MaxValue.

- `obj.SetLimitModeToNone ()` - Set/get the way voxel values will be limited. If this is set to None (the default), the output can have arbitrarily large values. If set to clamp, the output will be clamped to [MinValue,MaxValue]. If set to scale, the output will be linearly scaled between MinValue and MaxValue.

- `obj.SetLimitModeToClamp ()` - Set/get the way voxel values will be limited. If this is set to None (the default), the output can have arbitrarily large values. If set to clamp, the output will be clamped to [MinValue,MaxValue]. If set to scale, the output will be linearly scaled between MinValue and MaxValue.

- `obj.SetLimitModeToScale ()` - Set/get the way voxel values will be limited. If this is set to None (the default), the output can have arbitrarily large values. If set to clamp, the output will be clamped to [MinValue,MaxValue]. If set to scale, the output will be linearly scaled between MinValue and MaxValue.

- `obj.SetLimitModeToFreezeScale ()` - See the LimitMode method.

- `obj.SetMinValue (double )` - See the LimitMode method.

- `double = obj.GetMinValue ()` - See the LimitMode method.

- `obj.SetMaxValue (double )` - See the LimitMode method.

- `double = obj.GetMaxValue ()` - See the LimitMode method.

- `int = obj.GetNumberOfPointsSplatted ()` - This returns the number of points splatted (as opposed to discarded for being outside the image) during the previous pass.

- `obj.SetSplatConnection (vtkAlgorithmOutput )` - Convenience function for connecting the splat algorithm source. This is provided mainly for convenience using the filter with ParaView, VTK users should prefer SetInputConnection(1, splat) instead.

35.4 vtkGaussianSplatter

35.4.1 Usage

vtkGaussianSplatter is a filter that injects input points into a structured points (volume) dataset. As each point is injected, it "splats" or distributes values to nearby voxels. Data is distributed using an elliptical, Gaussian distribution function. The distribution function is modified using scalar values (expands distribution) or normals (creates ellipsoidal distribution rather than spherical).

In general, the Gaussian distribution function f(x) around a given splat point p is given by

f(x) = ScaleFactor * exp(ExponentFactor*((r/Radius)**2))

where x is the current voxel sample point; r is the distance —x-p— ExponentFactor ¡= 0.0, and ScaleFactor can be multiplied by the scalar value of the point p that is currently being splatted.

If points normals are present (and NormalWarping is on), then the splat function becomes elliptical (as compared to the spherical one described by the previous equation). The Gaussian distribution function then becomes:

f(x) = ScaleFactor * exp(ExponentFactor*(((rxy/E)**2 + z**2)/R**2))

where E is a user-defined eccentricity factor that controls the elliptical shape of the splat; z is the distance of the current voxel sample point along normal N; and rxy is the distance of x in the direction prependicular to N.

This class is typically used to convert point-valued distributions into a volume representation. The volume is then usually iso-surfaced or volume rendered to generate a visualization. It can be used to create surfaces from point distributions, or to create structure (i.e., topology) when none exists.

To create an instance of class vtkGaussianSplatter, simply invoke its constructor as follows

```
obj = vtkGaussianSplatter
```

35.4.2 Methods

The class vtkGaussianSplatter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGaussianSplatter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGaussianSplatter = obj.NewInstance ()`
- `vtkGaussianSplatter = obj.SafeDownCast (vtkObject o)`
- `obj.SetSampleDimensions (int i, int j, int k)` - Set / get the dimensions of the sampling structured point set. Higher values produce better results but are much slower.
- `obj.SetSampleDimensions (int dim[3])` - Set / get the dimensions of the sampling structured point set. Higher values produce better results but are much slower.
- `int = obj. GetSampleDimensions ()` - Set / get the dimensions of the sampling structured point set. Higher values produce better results but are much slower.
- `obj.SetModelBounds (double , double , double , double , double , double )` - Set / get the (xmin,xmax, ymin,ymax, zmin,zmax) bounding box in which the sampling is performed. If any of the (min,max) bounds values are min ¿= max, then the bounds will be computed automatically from the input data. Otherwise, the user-specified bounds will be used.
- `obj.SetModelBounds (double a[6])` - Set / get the (xmin,xmax, ymin,ymax, zmin,zmax) bounding box in which the sampling is performed. If any of the (min,max) bounds values are min ¿= max, then the bounds will be computed automatically from the input data. Otherwise, the user-specified bounds will be used.
- `double = obj. GetModelBounds ()` - Set / get the (xmin,xmax, ymin,ymax, zmin,zmax) bounding box in which the sampling is performed. If any of the (min,max) bounds values are min ¿= max, then the bounds will be computed automatically from the input data. Otherwise, the user-specified bounds will be used.
- `obj.SetRadius (double )` - Set / get the radius of propagation of the splat. This value is expressed as a percentage of the length of the longest side of the sampling volume. Smaller numbers greatly reduce execution time.
- `double = obj.GetRadiusMinValue ()` - Set / get the radius of propagation of the splat. This value is expressed as a percentage of the length of the longest side of the sampling volume. Smaller numbers greatly reduce execution time.
- `double = obj.GetRadiusMaxValue ()` - Set / get the radius of propagation of the splat. This value is expressed as a percentage of the length of the longest side of the sampling volume. Smaller numbers greatly reduce execution time.

- `double = obj.GetRadius ()` - Set / get the radius of propagation of the splat. This value is expressed as a percentage of the length of the longest side of the sampling volume. Smaller numbers greatly reduce execution time.
- `obj.SetScaleFactor (double )` - Multiply Gaussian splat distribution by this value. If ScalarWarping is on, then the Scalar value will be multiplied by the ScaleFactor times the Gaussian function.
- `double = obj.GetScaleFactorMinValue ()` - Multiply Gaussian splat distribution by this value. If ScalarWarping is on, then the Scalar value will be multiplied by the ScaleFactor times the Gaussian function.
- `double = obj.GetScaleFactorMaxValue ()` - Multiply Gaussian splat distribution by this value. If ScalarWarping is on, then the Scalar value will be multiplied by the ScaleFactor times the Gaussian function.
- `double = obj.GetScaleFactor ()` - Multiply Gaussian splat distribution by this value. If ScalarWarping is on, then the Scalar value will be multiplied by the ScaleFactor times the Gaussian function.
- `obj.SetExponentFactor (double )` - Set / get the sharpness of decay of the splats. This is the exponent constant in the Gaussian equation. Normally this is a negative value.
- `double = obj.GetExponentFactor ()` - Set / get the sharpness of decay of the splats. This is the exponent constant in the Gaussian equation. Normally this is a negative value.
- `obj.SetNormalWarping (int )` - Turn on/off the generation of elliptical splats. If normal warping is on, then the input normals affect the distribution of the splat. This boolean is used in combination with the Eccentricity ivar.
- `int = obj.GetNormalWarping ()` - Turn on/off the generation of elliptical splats. If normal warping is on, then the input normals affect the distribution of the splat. This boolean is used in combination with the Eccentricity ivar.
- `obj.NormalWarpingOn ()` - Turn on/off the generation of elliptical splats. If normal warping is on, then the input normals affect the distribution of the splat. This boolean is used in combination with the Eccentricity ivar.
- `obj.NormalWarpingOff ()` - Turn on/off the generation of elliptical splats. If normal warping is on, then the input normals affect the distribution of the splat. This boolean is used in combination with the Eccentricity ivar.
- `obj.SetEccentricity (double )` - Control the shape of elliptical splatting. Eccentricity is the ratio of the major axis (aligned along normal) to the minor (axes) aligned along other two axes. So Eccentricity ¿ 1 creates needles with the long axis in the direction of the normal; Eccentricity¡1 creates pancakes perpendicular to the normal vector.
- `double = obj.GetEccentricityMinValue ()` - Control the shape of elliptical splatting. Eccentricity is the ratio of the major axis (aligned along normal) to the minor (axes) aligned along other two axes. So Eccentricity ¿ 1 creates needles with the long axis in the direction of the normal; Eccentricity¡1 creates pancakes perpendicular to the normal vector.
- `double = obj.GetEccentricityMaxValue ()` - Control the shape of elliptical splatting. Eccentricity is the ratio of the major axis (aligned along normal) to the minor (axes) aligned along other two axes. So Eccentricity ¿ 1 creates needles with the long axis in the direction of the normal; Eccentricity¡1 creates pancakes perpendicular to the normal vector.
- `double = obj.GetEccentricity ()` - Control the shape of elliptical splatting. Eccentricity is the ratio of the major axis (aligned along normal) to the minor (axes) aligned along other two axes. So Eccentricity ¿ 1 creates needles with the long axis in the direction of the normal; Eccentricity¡1 creates pancakes perpendicular to the normal vector.

- `obj.SetScalarWarping (int )` - Turn on/off the scaling of splats by scalar value.
- `int = obj.GetScalarWarping ()` - Turn on/off the scaling of splats by scalar value.
- `obj.ScalarWarpingOn ()` - Turn on/off the scaling of splats by scalar value.
- `obj.ScalarWarpingOff ()` - Turn on/off the scaling of splats by scalar value.
- `obj.SetCapping (int )` - Turn on/off the capping of the outer boundary of the volume to a specified cap value. This can be used to close surfaces (after iso-surfacing) and create other effects.
- `int = obj.GetCapping ()` - Turn on/off the capping of the outer boundary of the volume to a specified cap value. This can be used to close surfaces (after iso-surfacing) and create other effects.
- `obj.CappingOn ()` - Turn on/off the capping of the outer boundary of the volume to a specified cap value. This can be used to close surfaces (after iso-surfacing) and create other effects.
- `obj.CappingOff ()` - Turn on/off the capping of the outer boundary of the volume to a specified cap value. This can be used to close surfaces (after iso-surfacing) and create other effects.
- `obj.SetCapValue (double )` - Specify the cap value to use. (This instance variable only has effect if the ivar Capping is on.)
- `double = obj.GetCapValue ()` - Specify the cap value to use. (This instance variable only has effect if the ivar Capping is on.)
- `obj.SetAccumulationMode (int )` - Specify the scalar accumulation mode. This mode expresses how scalar values are combined when splats are overlapped. The Max mode acts like a set union operation and is the most commonly used; the Min mode acts like a set intersection, and the sum is just weird.
- `int = obj.GetAccumulationModeMinValue ()` - Specify the scalar accumulation mode. This mode expresses how scalar values are combined when splats are overlapped. The Max mode acts like a set union operation and is the most commonly used; the Min mode acts like a set intersection, and the sum is just weird.
- `int = obj.GetAccumulationModeMaxValue ()` - Specify the scalar accumulation mode. This mode expresses how scalar values are combined when splats are overlapped. The Max mode acts like a set union operation and is the most commonly used; the Min mode acts like a set intersection, and the sum is just weird.
- `int = obj.GetAccumulationMode ()` - Specify the scalar accumulation mode. This mode expresses how scalar values are combined when splats are overlapped. The Max mode acts like a set union operation and is the most commonly used; the Min mode acts like a set intersection, and the sum is just weird.
- `obj.SetAccumulationModeToMin ()` - Specify the scalar accumulation mode. This mode expresses how scalar values are combined when splats are overlapped. The Max mode acts like a set union operation and is the most commonly used; the Min mode acts like a set intersection, and the sum is just weird.
- `obj.SetAccumulationModeToMax ()` - Specify the scalar accumulation mode. This mode expresses how scalar values are combined when splats are overlapped. The Max mode acts like a set union operation and is the most commonly used; the Min mode acts like a set intersection, and the sum is just weird.
- `obj.SetAccumulationModeToSum ()` - Specify the scalar accumulation mode. This mode expresses how scalar values are combined when splats are overlapped. The Max mode acts like a set union operation and is the most commonly used; the Min mode acts like a set intersection, and the sum is just weird.

- `string = obj.GetAccumulationModeAsString ()` - Specify the scalar accumulation mode. This mode expresses how scalar values are combined when splats are overlapped. The Max mode acts like a set union operation and is the most commonly used; the Min mode acts like a set intersection, and the sum is just weird.

- `obj.SetNullValue (double )` - Set the Null value for output points not receiving a contribution from the input points. (This is the initial value of the voxel samples.)

- `double = obj.GetNullValue ()` - Set the Null value for output points not receiving a contribution from the input points. (This is the initial value of the voxel samples.)

- `obj.ComputeModelBounds (vtkDataSet input, vtkImageData output, vtkInformation outInfo)` - Compute the size of the sample bounding box automatically from the input data. This is an internal helper function.

35.5 vtkImageAccumulate

35.5.1 Usage

vtkImageAccumulate - This filter divides component space into discrete bins. It then counts the number of pixels associated with each bin. The output is this "scatter plot" (histogram values for 1D). The dimensionality of the output depends on how many components the input pixels have. Input pixels with one component generate a 1D histogram. This filter can only handle images with 1 to 3 scalar components. The input can be any type, but the output is always int. Some statistics are computed on the pixel values at the same time. The SetStencil and ReverseStencil functions allow the statistics to be computed on an arbitrary portion of the input data. See the documentation for vtkImageStencilData for more information.

This filter also support ignoring pixel with value equal to 0. Using this option with vtkImageMask may result in results being slightly off since 0 could be a valid value from your input.

To create an instance of class vtkImageAccumulate, simply invoke its constructor as follows

```
obj = vtkImageAccumulate
```

35.5.2 Methods

The class vtkImageAccumulate has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageAccumulate class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkImageAccumulate = obj.NewInstance ()`

- `vtkImageAccumulate = obj.SafeDownCast (vtkObject o)`

- `obj.SetComponentSpacing (double , double , double )` - Set/Get - The component spacing is the dimension of each bin. This ends up being the spacing of the output "image". If the number of input scalar components are less than three, then some of these spacing values are ignored. For a 1D histogram with 10 bins spanning the values 1000 to 2000, this spacing should be set to 100, 0, 0

- `obj.SetComponentSpacing (double a[3])` - Set/Get - The component spacing is the dimension of each bin. This ends up being the spacing of the output "image". If the number of input scalar components are less than three, then some of these spacing values are ignored. For a 1D histogram with 10 bins spanning the values 1000 to 2000, this spacing should be set to 100, 0, 0

- `double = obj. GetComponentSpacing ()` - Set/Get - The component spacing is the dimension of each bin. This ends up being the spacing of the output "image". If the number of input scalar components are less than three, then some of these spacing values are ignored. For a 1D histogram with 10 bins spanning the values 1000 to 2000, this spacing should be set to 100, 0, 0
- `obj.SetComponentOrigin (double , double , double )` - Set/Get - The component origin is the location of bin (0, 0, 0). Note that if the Component extent does not include the value (0,0,0), then this origin bin will not actually be in the output. The origin of the output ends up being the same as the componenet origin. For a 1D histogram with 10 bins spanning the values 1000 to 2000, this origin should be set to 1000, 0, 0
- `obj.SetComponentOrigin (double a[3])` - Set/Get - The component origin is the location of bin (0, 0, 0). Note that if the Component extent does not include the value (0,0,0), then this origin bin will not actually be in the output. The origin of the output ends up being the same as the componenet origin. For a 1D histogram with 10 bins spanning the values 1000 to 2000, this origin should be set to 1000, 0, 0
- `double = obj. GetComponentOrigin ()` - Set/Get - The component origin is the location of bin (0, 0, 0). Note that if the Component extent does not include the value (0,0,0), then this origin bin will not actually be in the output. The origin of the output ends up being the same as the componenet origin. For a 1D histogram with 10 bins spanning the values 1000 to 2000, this origin should be set to 1000, 0, 0
- `obj.SetComponentExtent (int extent[6])` - Set/Get - The component extent sets the number/extent of the bins. For a 1D histogram with 10 bins spanning the values 1000 to 2000, this extent should be set to 0, 9, 0, 0, 0, 0. The extent specifies inclusive min/max values. This implies that the top extent should be set to the number of bins - 1.
- `obj.SetComponentExtent (int minX, int maxX, int minY, int maxY, int minZ, int maxZ)` - Set/Get - The component extent sets the number/extent of the bins. For a 1D histogram with 10 bins spanning the values 1000 to 2000, this extent should be set to 0, 9, 0, 0, 0, 0. The extent specifies inclusive min/max values. This implies that the top extent should be set to the number of bins - 1.
- `obj.GetComponentExtent (int extent[6])` - Set/Get - The component extent sets the number/extent of the bins. For a 1D histogram with 10 bins spanning the values 1000 to 2000, this extent should be set to 0, 9, 0, 0, 0, 0. The extent specifies inclusive min/max values. This implies that the top extent should be set to the number of bins - 1.
- `int = obj.GetComponentExtent ()` - Use a stencil to specify which voxels to accumulate.
- `obj.SetStencil (vtkImageStencilData stencil)` - Use a stencil to specify which voxels to accumulate.
- `vtkImageStencilData = obj.GetStencil ()` - Use a stencil to specify which voxels to accumulate.
- `obj.SetReverseStencil (int )` - Reverse the stencil.
- `int = obj.GetReverseStencilMinValue ()` - Reverse the stencil.
- `int = obj.GetReverseStencilMaxValue ()` - Reverse the stencil.
- `obj.ReverseStencilOn ()` - Reverse the stencil.
- `obj.ReverseStencilOff ()` - Reverse the stencil.
- `int = obj.GetReverseStencil ()` - Reverse the stencil.
- `double = obj. GetMin ()` - Get the statistics information for the data.
- `double = obj. GetMax ()` - Get the statistics information for the data.

- `double = obj. GetMean ()` - Get the statistics information for the data.
- `double = obj. GetStandardDeviation ()` - Get the statistics information for the data.
- `long = obj.GetVoxelCount ()` - Get the statistics information for the data.
- `obj.SetIgnoreZero (int )` - Should the data with value 0 be ignored?
- `int = obj.GetIgnoreZeroMinValue ()` - Should the data with value 0 be ignored?
- `int = obj.GetIgnoreZeroMaxValue ()` - Should the data with value 0 be ignored?
- `int = obj.GetIgnoreZero ()` - Should the data with value 0 be ignored?
- `obj.IgnoreZeroOn ()` - Should the data with value 0 be ignored?
- `obj.IgnoreZeroOff ()` - Should the data with value 0 be ignored?

35.6 vtkImageAnisotropicDiffusion2D

35.6.1 Usage

vtkImageAnisotropicDiffusion2D diffuses a 2d image iteratively. The neighborhood of the diffusion is determined by the instance flags. If "Edges" is on the 4 edge connected voxels are included, and if "Corners" is on, the 4 corner connected voxels are included. "DiffusionFactor" determines how far a pixel value moves toward its neighbors, and is insensitive to the number of neighbors chosen. The diffusion is anisotropic because it only occurs when a gradient measure is below "GradientThreshold". Two gradient measures exist and are toggled by the "GradientMagnitudeThreshold" flag. When "GradientMagnitudeThreshold" is on, the magnitude of the gradient, computed by central differences, above "DiffusionThreshold" a voxel is not modified. The alternative measure examines each neighbor independently. The gradient between the voxel and the neighbor must be below the "DiffusionThreshold" for diffusion to occur with THAT neighbor.

To create an instance of class vtkImageAnisotropicDiffusion2D, simply invoke its constructor as follows

```
obj = vtkImageAnisotropicDiffusion2D
```

35.6.2 Methods

The class vtkImageAnisotropicDiffusion2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageAnisotropicDiffusion2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageAnisotropicDiffusion2D = obj.NewInstance ()`
- `vtkImageAnisotropicDiffusion2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfIterations (int num)` - This method sets the number of interations which also affects the input neighborhood needed to compute one output pixel. Each iterations requires an extra pixel layer on the neighborhood. This is only relavent when you are trying to stream or are requesting a sub extent of the "wholeExtent".
- `int = obj.GetNumberOfIterations ()` - Get the number of iterations.
- `obj.SetDiffusionThreshold (double )` - Set/Get the difference threshold that stops diffusion. when the difference between two pixel is greater than this threshold, the pixels are not diffused. This causes diffusion to avoid sharp edges. If the GradientMagnitudeThreshold is set, then gradient magnitude is used for comparison instead of pixel differences.

- `double = obj.GetDiffusionThreshold ()` - Set/Get the difference threshold that stops diffusion. when the difference between two pixel is greater than this threshold, the pixels are not diffused. This causes diffusion to avoid sharp edges. If the GradientMagnitudeThreshold is set, then gradient magnitude is used for comparison instead of pixel differences.
- `obj.SetDiffusionFactor (double )` - The diffusion factor specifies how much neighboring pixels effect each other. No diffusion occurs with a factor of 0, and a diffusion factor of 1 causes the pixel to become the average of all its neighbors.
- `double = obj.GetDiffusionFactor ()` - The diffusion factor specifies how much neighboring pixels effect each other. No diffusion occurs with a factor of 0, and a diffusion factor of 1 causes the pixel to become the average of all its neighbors.
- `obj.SetFaces (int )` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `int = obj.GetFaces ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.FacesOn ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.FacesOff ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.SetEdges (int )` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `int = obj.GetEdges ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.EdgesOn ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.EdgesOff ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.SetCorners (int )` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `int = obj.GetCorners ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.CornersOn ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.CornersOff ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.SetGradientMagnitudeThreshold (int )` - Switch between gradient magnitude threshold and pixel gradient threshold.
- `int = obj.GetGradientMagnitudeThreshold ()` - Switch between gradient magnitude threshold and pixel gradient threshold.
- `obj.GradientMagnitudeThresholdOn ()` - Switch between gradient magnitude threshold and pixel gradient threshold.
- `obj.GradientMagnitudeThresholdOff ()` - Switch between gradient magnitude threshold and pixel gradient threshold.

35.7 vtkImageAnisotropicDiffusion3D

35.7.1 Usage

vtkImageAnisotropicDiffusion3D diffuses an volume iteratively. The neighborhood of the diffusion is determined by the instance flags. if "Faces" is on, the 6 voxels adjoined by faces are included in the neighborhood. If "Edges" is on the 12 edge connected voxels are included, and if "Corners" is on, the 8 corner connected voxels are included. "DiffusionFactor" determines how far a pixel value moves toward its neighbors, and is insensitive to the number of neighbors chosen. The diffusion is anisotropic because it only occurs when a gradient measure is below "GradientThreshold". Two gradient measures exist and are toggled by the "GradientMagnitudeThreshold" flag. When "GradientMagnitudeThreshold" is on, the magnitude of the

gradient, computed by central differences, above "DiffusionThreshold" a voxel is not modified. The alternative measure examines each neighbor independently. The gradient between the voxel and the neighbor must be below the "DiffusionThreshold" for diffusion to occur with THAT neighbor.

To create an instance of class vtkImageAnisotropicDiffusion3D, simply invoke its constructor as follows

```
obj = vtkImageAnisotropicDiffusion3D
```

35.7.2 Methods

The class vtkImageAnisotropicDiffusion3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageAnisotropicDiffusion3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageAnisotropicDiffusion3D = obj.NewInstance ()`
- `vtkImageAnisotropicDiffusion3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfIterations (int num)` - This method sets the number of interations which also affects the input neighborhood needed to compute one output pixel. Each iterations requires an extra pixel layer on the neighborhood. This is only relavent when you are trying to stream or are requesting a sub extent of the "wholeExtent".
- `int = obj.GetNumberOfIterations ()` - Get the number of iterations.
- `obj.SetDiffusionThreshold (double )` - Set/Get the difference threshold that stops diffusion. when the difference between two pixel is greater than this threshold, the pixels are not diffused. This causes diffusion to avoid sharp edges. If the GradientMagnitudeThreshold is set, then gradient magnitude is used for comparison instead of pixel differences.
- `double = obj.GetDiffusionThreshold ()` - Set/Get the difference threshold that stops diffusion. when the difference between two pixel is greater than this threshold, the pixels are not diffused. This causes diffusion to avoid sharp edges. If the GradientMagnitudeThreshold is set, then gradient magnitude is used for comparison instead of pixel differences.
- `obj.SetDiffusionFactor (double )` - Set/Get the difference factor
- `double = obj.GetDiffusionFactor ()` - Set/Get the difference factor
- `obj.SetFaces (int )` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `int = obj.GetFaces ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.FacesOn ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.FacesOff ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.SetEdges (int )` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `int = obj.GetEdges ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.EdgesOn ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.EdgesOff ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.SetCorners (int )` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `int = obj.GetCorners ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).

- `obj.CornersOn ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.CornersOff ()` - Choose neighbors to diffuse (6 faces, 12 edges, 8 corners).
- `obj.SetGradientMagnitudeThreshold (int )` - Switch between gradient magnitude threshold and pixel gradient threshold.
- `int = obj.GetGradientMagnitudeThreshold ()` - Switch between gradient magnitude threshold and pixel gradient threshold.
- `obj.GradientMagnitudeThresholdOn ()` - Switch between gradient magnitude threshold and pixel gradient threshold.
- `obj.GradientMagnitudeThresholdOff ()` - Switch between gradient magnitude threshold and pixel gradient threshold.

35.8 vtkImageAppend

35.8.1 Usage

vtkImageAppend takes the components from multiple inputs and merges them into one output. The output images are append along the "AppendAxis". Except for the append axis, all inputs must have the same extent. All inputs must have the same number of scalar components. A future extension might be to pad or clip inputs to have the same extent. The output has the same origin and spacing as the first input. The origin and spacing of all other inputs are ignored. All inputs must have the same scalar type.

To create an instance of class vtkImageAppend, simply invoke its constructor as follows

```
obj = vtkImageAppend
```

35.8.2 Methods

The class vtkImageAppend has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageAppend class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageAppend = obj.NewInstance ()`
- `vtkImageAppend = obj.SafeDownCast (vtkObject o)`
- `obj.ReplaceNthInputConnection (int idx, vtkAlgorithmOutput input)` - Replace one of the input connections with a new input. You can only replace input connections that you previously created with AddInputConnection() or, in the case of the first input, with SetInputConnection().
- `obj.SetInput (int num, vtkDataObject input)` - Set an Input of this filter. This method is only for support of old-style pipeline connections. When writing new code you should use SetInputConnection(), AddInputConnection(), and ReplaceNthInputConnection() instead.
- `obj.SetInput (vtkDataObject input)` - Set an Input of this filter. This method is only for support of old-style pipeline connections. When writing new code you should use SetInputConnection(), AddInputConnection(), and ReplaceNthInputConnection() instead.
- `vtkDataObject = obj.GetInput (int num)` - Get one input to this filter. This method is only for support of old-style pipeline connections. When writing new code you should use vtkAlgorithm::GetInputConnection(0, num).

- `vtkDataObject = obj.GetInput ()` - Get one input to this filter. This method is only for support of old-style pipeline connections. When writing new code you should use vtkAlgorithm::GetInputConnection(0, num).
- `int = obj.GetNumberOfInputs ()` - Get the number of inputs to this filter. This method is only for support of old-style pipeline connections. When writing new code you should use vtkAlgorithm::GetNumberOfInputCon
- `obj.SetAppendAxis (int )` - This axis is expanded to hold the multiple images. The default AppendAxis is the X axis. If you want to create a volue from a series of XY images, then you should set the AppendAxis to 2 (Z axis).
- `int = obj.GetAppendAxis ()` - This axis is expanded to hold the multiple images. The default AppendAxis is the X axis. If you want to create a volue from a series of XY images, then you should set the AppendAxis to 2 (Z axis).
- `obj.SetPreserveExtents (int )` - By default "PreserveExtents" is off and the append axis is used. When "PreseveExtents" is on, the extent of the inputs is used to place the image in the output. The whole extent of the output is the union of the input whole extents. Any portion of the output not covered by the inputs is set to zero. The origin and spacing is taken from the first input.
- `int = obj.GetPreserveExtents ()` - By default "PreserveExtents" is off and the append axis is used. When "PreseveExtents" is on, the extent of the inputs is used to place the image in the output. The whole extent of the output is the union of the input whole extents. Any portion of the output not covered by the inputs is set to zero. The origin and spacing is taken from the first input.
- `obj.PreserveExtentsOn ()` - By default "PreserveExtents" is off and the append axis is used. When "PreseveExtents" is on, the extent of the inputs is used to place the image in the output. The whole extent of the output is the union of the input whole extents. Any portion of the output not covered by the inputs is set to zero. The origin and spacing is taken from the first input.
- `obj.PreserveExtentsOff ()` - By default "PreserveExtents" is off and the append axis is used. When "PreseveExtents" is on, the extent of the inputs is used to place the image in the output. The whole extent of the output is the union of the input whole extents. Any portion of the output not covered by the inputs is set to zero. The origin and spacing is taken from the first input.

35.9 vtkImageAppendComponents

35.9.1 Usage

vtkImageAppendComponents takes the components from two inputs and merges them into one output. If Input1 has M components, and Input2 has N components, the output will have M+N components with input1 components coming first.

To create an instance of class vtkImageAppendComponents, simply invoke its constructor as follows

```
obj = vtkImageAppendComponents
```

35.9.2 Methods

The class vtkImageAppendComponents has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageAppendComponents class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageAppendComponents = obj.NewInstance ()`

- `vtkImageAppendComponents = obj.SafeDownCast (vtkObject o)`
- `obj.ReplaceNthInputConnection (int idx, vtkAlgorithmOutput input)` - Replace one of the input connections with a new input. You can only replace input connections that you previously created with AddInputConnection() or, in the case of the first input, with SetInputConnection().
- `obj.SetInput (int num, vtkDataObject input)` - Set an Input of this filter. This method is only for support of old-style pipeline connections. When writing new code you should use SetInputConnection(), AddInputConnection(), and ReplaceNthInputConnection() instead.
- `obj.SetInput (vtkDataObject input)` - Set an Input of this filter. This method is only for support of old-style pipeline connections. When writing new code you should use SetInputConnection(), AddInputConnection(), and ReplaceNthInputConnection() instead.
- `vtkDataObject = obj.GetInput (int num)` - Get one input to this filter. This method is only for support of old-style pipeline connections. When writing new code you should use vtkAlgorithm::GetInputConnection(0, num).
- `vtkDataObject = obj.GetInput ()` - Get one input to this filter. This method is only for support of old-style pipeline connections. When writing new code you should use vtkAlgorithm::GetInputConnection(0, num).
- `int = obj.GetNumberOfInputs ()` - Get the number of inputs to this filter. This method is only for support of old-style pipeline connections. When writing new code you should use vtkAlgorithm::GetNumberOfInputCon

35.10 vtkImageBlend

35.10.1 Usage

vtkImageBlend takes L, LA, RGB, or RGBA images as input and blends them according to the alpha values and/or the opacity setting for each input.

The spacing, origin, extent, and number of components of the output are the same as those for the first input. If the input has an alpha component, then this component is copied unchanged into the output. In addition, if the first input has either one component or two components i.e. if it is either L (greyscale) or LA (greyscale + alpha) then all other inputs must also be L or LA.

Different blending modes are available:

Normal (default) : This is the standard blending mode used by OpenGL and other graphics packages. The output always has the same number of components and the same extent as the first input. The alpha value of the first input is not used in the blending computation, instead it is copied directly to the output.

```
output <- input[0]
foreach input i {
  foreach pixel px {
    r <- input[i](px)(alpha) * opacity[i]
    f <- (255 - r)
    output(px) <- output(px) * f + input(px) * r
  }
}
```

Compound : Images are compounded together and each component is scaled by the sum of the alpha/opacity values. Use the CompoundThreshold method to set specify a threshold in compound mode. Pixels with opacity*alpha less or equal than this threshold are ignored. The alpha value of the first input, if present, is NOT copied to the alpha value of the output. The output always has the same number of components and the same extent as the first input.

```
output <- 0
foreach pixel px {
  sum <- 0
  foreach input i {
    r <- input[i](px)(alpha) * opacity(i)
    sum <- sum + r
    if r > threshold {
      output(px) <- output(px) + input(px) * r
    }
  }
  output(px) <- output(px) / sum
}
```

To create an instance of class vtkImageBlend, simply invoke its constructor as follows

```
obj = vtkImageBlend
```

35.10.2 Methods

The class vtkImageBlend has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageBlend class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageBlend = obj.NewInstance ()`
- `vtkImageBlend = obj.SafeDownCast (vtkObject o)`
- `obj.ReplaceNthInputConnection (int idx, vtkAlgorithmOutput input)` - Replace one of the input connections with a new input. You can only replace input connections that you previously created with AddInputConnection() or, in the case of the first input, with SetInputConnection().
- `obj.SetInput (int num, vtkDataObject input)` - Set an Input of this filter. This method is only for support of old-style pipeline connections. When writing new code you should use SetInputConnection(), AddInputConnection(), and ReplaceNthInputConnection() instead.
- `obj.SetInput (vtkDataObject input)` - Set an Input of this filter. This method is only for support of old-style pipeline connections. When writing new code you should use SetInputConnection(), AddInputConnection(), and ReplaceNthInputConnection() instead.
- `vtkDataObject = obj.GetInput (int num)` - Get one input to this filter. This method is only for support of old-style pipeline connections. When writing new code you should use vtkAlgorithm::GetInputConnection(0, num).
- `vtkDataObject = obj.GetInput ()` - Get one input to this filter. This method is only for support of old-style pipeline connections. When writing new code you should use vtkAlgorithm::GetInputConnection(0, num).
- `int = obj.GetNumberOfInputs ()` - Get the number of inputs to this filter. This method is only for support of old-style pipeline connections. When writing new code you should use vtkAlgorithm::GetNumberOfInputCon
- `obj.SetOpacity (int idx, double opacity)` - Set the opacity of an input image: the alpha values of the image are multiplied by the opacity. The opacity of image idx=0 is ignored.

- `double = obj.GetOpacity (int idx)` - Set the opacity of an input image: the alpha values of the image are multiplied by the opacity. The opacity of image idx=0 is ignored.
- `obj.SetStencil (vtkImageStencilData stencil)` - Set a stencil to apply when blending the data.
- `vtkImageStencilData = obj.GetStencil ()` - Set a stencil to apply when blending the data.
- `obj.SetBlendMode (int )` - Set the blend mode
- `int = obj.GetBlendModeMinValue ()` - Set the blend mode
- `int = obj.GetBlendModeMaxValue ()` - Set the blend mode
- `int = obj.GetBlendMode ()` - Set the blend mode
- `obj.SetBlendModeToNormal ()` - Set the blend mode
- `obj.SetBlendModeToCompound ()` - Set the blend mode
- `string = obj.GetBlendModeAsString (void )` - Set the blend mode
- `obj.SetCompoundThreshold (double )` - Specify a threshold in compound mode. Pixels with opacity*alpha less or equal the threshold are ignored.
- `double = obj.GetCompoundThreshold ()` - Specify a threshold in compound mode. Pixels with opacity*alpha less or equal the threshold are ignored.

35.11 vtkImageButterworthHighPass

35.11.1 Usage

This filter only works on an image after it has been converted to frequency domain by a vtkImageFFT filter. A vtkImageRFFT filter can be used to convert the output back into the spatial domain. vtkImageButterworthHighPass the frequency components around 0 are attenuated. Input and output are in doubles, with two components (complex numbers). out(i, j) = 1 / (1 + pow(CutOff/Freq(i,j), 2*Order));

To create an instance of class vtkImageButterworthHighPass, simply invoke its constructor as follows

```
obj = vtkImageButterworthHighPass
```

35.11.2 Methods

The class vtkImageButterworthHighPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageButterworthHighPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageButterworthHighPass = obj.NewInstance ()`
- `vtkImageButterworthHighPass = obj.SafeDownCast (vtkObject o)`
- `obj.SetCutOff (double , double , double )` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetCutOff (double a[3])` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).

- `obj.SetCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetXCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetYCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetZCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj. GetCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetXCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetYCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetZCutOff ()` - The order determines sharpness of the cutoff curve.
- `obj.SetOrder (int )` - The order determines sharpness of the cutoff curve.
- `int = obj.GetOrder ()` - The order determines sharpness of the cutoff curve.

35.12 vtkImageButterworthLowPass

35.12.1 Usage

This filter only works on an image after it has been converted to frequency domain by a vtkImageFFT filter. A vtkImageRFFT filter can be used to convert the output back into the spatial domain. vtkImageButterworthLowPass the high frequency components are attenuated. Input and output are in doubles, with two components (complex numbers). out(i, j) = (1 + pow(CutOff/Freq(i,j), 2*Order));

To create an instance of class vtkImageButterworthLowPass, simply invoke its constructor as follows

```
obj = vtkImageButterworthLowPass
```

35.12.2 Methods

The class vtkImageButterworthLowPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageButterworthLowPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageButterworthLowPass = obj.NewInstance ()`
- `vtkImageButterworthLowPass = obj.SafeDownCast (vtkObject o)`
- `obj.SetCutOff (double , double , double )` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetCutOff (double  a[3])` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).

- `obj.SetCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetXCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetYCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetZCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj. GetCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetXCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetYCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetZCutOff ()` - The order determines sharpness of the cutoff curve.
- `obj.SetOrder (int )` - The order determines sharpness of the cutoff curve.
- `int = obj.GetOrder ()` - The order determines sharpness of the cutoff curve.

35.13 vtkImageCacheFilter

35.13.1 Usage

vtkImageCacheFilter keep a number of vtkImageDataObjects from previous updates to satisfy future updates without needing to update the input. It does not change the data at all. It just makes the pipeline more efficient at the expense of using extra memory.

To create an instance of class vtkImageCacheFilter, simply invoke its constructor as follows

```
obj = vtkImageCacheFilter
```

35.13.2 Methods

The class vtkImageCacheFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageCacheFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageCacheFilter = obj.NewInstance ()`
- `vtkImageCacheFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetCacheSize (int size)` - This is the maximum number of images that can be retained in memory. it defaults to 10.
- `int = obj.GetCacheSize ()` - This is the maximum number of images that can be retained in memory. it defaults to 10.

35.14 vtkImageCanvasSource2D

35.14.1 Usage

vtkImageCanvasSource2D is a source that starts as a blank image. you may add to the image with two-dimensional drawing routines. It can paint multi-spectral images.

To create an instance of class vtkImageCanvasSource2D, simply invoke its constructor as follows

```
obj = vtkImageCanvasSource2D
```

35.14.2 Methods

The class vtkImageCanvasSource2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageCanvasSource2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageCanvasSource2D = obj.NewInstance ()`
- `vtkImageCanvasSource2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetDrawColor (double , double , double , double )` - Set/Get DrawColor. This is the value that is used when filling data or drawing lines. Default is (0,0,0,0)
- `obj.SetDrawColor (double  a[4])` - Set/Get DrawColor. This is the value that is used when filling data or drawing lines. Default is (0,0,0,0)
- `double = obj. GetDrawColor ()` - Set/Get DrawColor. This is the value that is used when filling data or drawing lines. Default is (0,0,0,0)
- `obj.SetDrawColor (double a)` - Set DrawColor to (a, b, 0, 0)
- `obj.SetDrawColor (double a, double b)` - Set DrawColor to (a, b, c, 0)
- `obj.SetDrawColor (double a, double b, double c)` - Set the pixels inside the box (min0, max0, min1, max1) to the current DrawColor
- `obj.FillBox (int min0, int max0, int min1, int max1)` - Set the pixels inside the box (min0, max0, min1, max1) to the current DrawColor
- `obj.FillTube (int x0, int y0, int x1, int y1, double radius)` - Set the pixels inside the box (min0, max0, min1, max1) to the current DrawColor
- `obj.FillTriangle (int x0, int y0, int x1, int y1, int x2, int y2)` - Set the pixels inside the box (min0, max0, min1, max1) to the current DrawColor
- `obj.DrawCircle (int c0, int c1, double radius)` - Set the pixels inside the box (min0, max0, min1, max1) to the current DrawColor
- `obj.DrawPoint (int p0, int p1)` - Set the pixels inside the box (min0, max0, min1, max1) to the current DrawColor
- `obj.DrawSegment (int x0, int y0, int x1, int y1)` - Set the pixels inside the box (min0, max0, min1, max1) to the current DrawColor
- `obj.DrawSegment3D (double p0, double p1)` - Set the pixels inside the box (min0, max0, min1, max1) to the current DrawColor

- `obj.DrawSegment3D (double x1, double y1, double z1, double x2, double y2, double z2)` - Draw subimage of the input image in the canvas at position x0 and y0. The subimage is defined with sx, sy, width, and height.
- `obj.DrawImage (int x0, int y0, vtkImageData i)` - Draw subimage of the input image in the canvas at position x0 and y0. The subimage is defined with sx, sy, width, and height.
- `obj.DrawImage (int x0, int y0, vtkImageData , int sx, int sy, int width, int height)` - Draw subimage of the input image in the canvas at position x0 and y0. The subimage is defined with sx, sy, width, and height.
- `obj.FillPixel (int x, int y)` - Fill a colored area with another color. (like connectivity) All pixels connected (and with the same value) to pixel (x, y) get replaced by the current "DrawColor".
- `obj.SetExtent (int extent)` - These methods set the WholeExtent of the output It sets the size of the canvas. Extent is a min max 3D box. Minimums and maximums are inclusive.
- `obj.SetExtent (int x1, int x2, int y1, int y2, int z1, int z2)` - These methods set the WholeExtent of the output It sets the size of the canvas. Extent is a min max 3D box. Minimums and maximums are inclusive.
- `obj.SetDefaultZ (int )` - The drawing operations can only draw into one 2D XY plane at a time. If the canvas is a 3D volume, then this z value is used as the default for 2D operations. The default is 0.
- `int = obj.GetDefaultZ ()` - The drawing operations can only draw into one 2D XY plane at a time. If the canvas is a 3D volume, then this z value is used as the default for 2D operations. The default is 0.
- `obj.SetRatio (double , double , double )` - Set/Get Ratio. This is the value that is used to pre-multiply each (x, y, z) drawing coordinates (including DefaultZ). The default is (1, 1, 1)
- `obj.SetRatio (double a[3])` - Set/Get Ratio. This is the value that is used to pre-multiply each (x, y, z) drawing coordinates (including DefaultZ). The default is (1, 1, 1)
- `double = obj. GetRatio ()` - Set/Get Ratio. This is the value that is used to pre-multiply each (x, y, z) drawing coordinates (including DefaultZ). The default is (1, 1, 1)
- `obj.SetNumberOfScalarComponents (int i)` - Set the number of scalar components
- `int = obj.GetNumberOfScalarComponents () const` - Set the number of scalar components
- `obj.SetScalarTypeToFloat ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `obj.SetScalarTypeToDouble ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `obj.SetScalarTypeToInt ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `obj.SetScalarTypeToUnsignedInt ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

- `obj.SetScalarTypeToLong ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `obj.SetScalarTypeToUnsignedLong ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `obj.SetScalarTypeToShort ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `obj.SetScalarTypeToUnsignedShort ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `obj.SetScalarTypeToUnsignedChar ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `obj.SetScalarTypeToChar ()` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `obj.SetScalarType (int )` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility
- `int = obj.GetScalarType () const` - Set/Get the data scalar type (i.e VTK_DOUBLE). Note that these methods are setting and getting the pipeline scalar type. i.e. they are setting the type that the image data will be once it has executed. Until the REQUEST_DATA pass the actual scalars may be of some other type. This is for backwards compatibility

35.15 vtkImageCast

35.15.1 Usage

vtkImageCast filter casts the input type to match the output type in the image processing pipeline. The filter does nothing if the input already has the correct type. To specify the "CastTo" type, use "SetOutputScalarType" method.

To create an instance of class vtkImageCast, simply invoke its constructor as follows

```
obj = vtkImageCast
```

35.15.2 Methods

The class vtkImageCast has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageCast class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageCast = obj.NewInstance ()`
- `vtkImageCast = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutputScalarType (int )` - Set the desired output scalar type to cast to.
- `int = obj.GetOutputScalarType ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToFloat ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToDouble ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToInt ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToUnsignedInt ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToLong ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToUnsignedLong ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToShort ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToUnsignedShort ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToUnsignedChar ()` - Set the desired output scalar type to cast to.
- `obj.SetOutputScalarTypeToChar ()` - Set the desired output scalar type to cast to.
- `obj.SetClampOverflow (int )` - When the ClampOverflow flag is on, the data is thresholded so that the output value does not exceed the max or min of the data type. By default ClampOverflow is off.
- `int = obj.GetClampOverflow ()` - When the ClampOverflow flag is on, the data is thresholded so that the output value does not exceed the max or min of the data type. By default ClampOverflow is off.
- `obj.ClampOverflowOn ()` - When the ClampOverflow flag is on, the data is thresholded so that the output value does not exceed the max or min of the data type. By default ClampOverflow is off.
- `obj.ClampOverflowOff ()` - When the ClampOverflow flag is on, the data is thresholded so that the output value does not exceed the max or min of the data type. By default ClampOverflow is off.

35.16 vtkImageChangeInformation

35.16.1 Usage

vtkImageChangeInformation modify the spacing, origin, or extent of the data without changing the data itself. The data is not resampled by this filter, only the information accompanying the data is modified.

To create an instance of class vtkImageChangeInformation, simply invoke its constructor as follows

```
obj = vtkImageChangeInformation
```

35.16.2 Methods

The class vtkImageChangeInformation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageChangeInformation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageChangeInformation = obj.NewInstance ()`
- `vtkImageChangeInformation = obj.SafeDownCast (vtkObject o)`
- `obj.SetInformationInput (vtkImageData )` - Copy the information from another data set. By default, the information is copied from the input.
- `vtkImageData = obj.GetInformationInput ()` - Copy the information from another data set. By default, the information is copied from the input.
- `obj.SetOutputExtentStart (int , int , int )` - Specify new starting values for the extent explicitly. These values are used as WholeExtent[0], WholeExtent[2] and WholeExtent[4] of the output. The default is to the use the extent start of the Input, or of the InformationInput if InformationInput is set.
- `obj.SetOutputExtentStart (int a[3])` - Specify new starting values for the extent explicitly. These values are used as WholeExtent[0], WholeExtent[2] and WholeExtent[4] of the output. The default is to the use the extent start of the Input, or of the InformationInput if InformationInput is set.
- `int = obj. GetOutputExtentStart ()` - Specify new starting values for the extent explicitly. These values are used as WholeExtent[0], WholeExtent[2] and WholeExtent[4] of the output. The default is to the use the extent start of the Input, or of the InformationInput if InformationInput is set.
- `obj.SetOutputSpacing (double , double , double )` - Specify a new data spacing explicitly. The default is to use the spacing of the Input, or of the InformationInput if InformationInput is set.
- `obj.SetOutputSpacing (double a[3])` - Specify a new data spacing explicitly. The default is to use the spacing of the Input, or of the InformationInput if InformationInput is set.
- `double = obj. GetOutputSpacing ()` - Specify a new data spacing explicitly. The default is to use the spacing of the Input, or of the InformationInput if InformationInput is set.
- `obj.SetOutputOrigin (double , double , double )` - Specify a new data origin explicitly. The default is to use the origin of the Input, or of the InformationInput if InformationInput is set.
- `obj.SetOutputOrigin (double a[3])` - Specify a new data origin explicitly. The default is to use the origin of the Input, or of the InformationInput if InformationInput is set.
- `double = obj. GetOutputOrigin ()` - Specify a new data origin explicitly. The default is to use the origin of the Input, or of the InformationInput if InformationInput is set.
- `obj.SetCenterImage (int )` - Set the Origin of the output so that image coordinate (0,0,0) lies at the Center of the data set. This will override SetOutputOrigin. This is often a useful operation to apply before using vtkImageReslice to apply a transformation to an image.
- `obj.CenterImageOn ()` - Set the Origin of the output so that image coordinate (0,0,0) lies at the Center of the data set. This will override SetOutputOrigin. This is often a useful operation to apply before using vtkImageReslice to apply a transformation to an image.

- `obj.CenterImageOff ()` - Set the Origin of the output so that image coordinate (0,0,0) lies at the Center of the data set. This will override SetOutputOrigin. This is often a useful operation to apply before using vtkImageReslice to apply a transformation to an image.
- `int = obj.GetCenterImage ()` - Set the Origin of the output so that image coordinate (0,0,0) lies at the Center of the data set. This will override SetOutputOrigin. This is often a useful operation to apply before using vtkImageReslice to apply a transformation to an image.
- `obj.SetExtentTranslation (int , int , int )` - Apply a translation to the extent.
- `obj.SetExtentTranslation (int a[3])` - Apply a translation to the extent.
- `int = obj. GetExtentTranslation ()` - Apply a translation to the extent.
- `obj.SetSpacingScale (double , double , double )` - Apply a scale factor to the spacing.
- `obj.SetSpacingScale (double a[3])` - Apply a scale factor to the spacing.
- `double = obj. GetSpacingScale ()` - Apply a scale factor to the spacing.
- `obj.SetOriginTranslation (double , double , double )` - Apply a translation to the origin.
- `obj.SetOriginTranslation (double a[3])` - Apply a translation to the origin.
- `double = obj. GetOriginTranslation ()` - Apply a translation to the origin.
- `obj.SetOriginScale (double , double , double )` - Apply a scale to the origin. The scale is applied before the translation.
- `obj.SetOriginScale (double a[3])` - Apply a scale to the origin. The scale is applied before the translation.
- `double = obj. GetOriginScale ()` - Apply a scale to the origin. The scale is applied before the translation.

35.17 vtkImageCheckerboard

35.17.1 Usage

vtkImageCheckerboard displays two images as one using a checkerboard pattern. This filter can be used to compare two images. The checkerboard pattern is controlled by the NumberOfDivisions ivar. This controls the number of checkerboard divisions in the whole extent of the image.

To create an instance of class vtkImageCheckerboard, simply invoke its constructor as follows

```
obj = vtkImageCheckerboard
```

35.17.2 Methods

The class vtkImageCheckerboard has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageCheckerboard class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageCheckerboard = obj.NewInstance ()`
- `vtkImageCheckerboard = obj.SafeDownCast (vtkObject o)`

- `obj.SetNumberOfDivisions (int , int , int )` - Set/Get the number of divisions along each axis.
- `obj.SetNumberOfDivisions (int  a[3])` - Set/Get the number of divisions along each axis.
- `int = obj. GetNumberOfDivisions ()` - Set/Get the number of divisions along each axis.
- `obj.SetInput1 (vtkDataObject in)` - Set the two inputs to this filter
- `obj.SetInput2 (vtkDataObject in)`

35.18 vtkImageCityBlockDistance

35.18.1 Usage

vtkImageCityBlockDistance creates a distance map using the city block (Manhatten) distance measure. The input is a mask. Zero values are considered boundaries. The output pixel is the minimum of the input pixel and the distance to a boundary (or neighbor value + 1 unit). distance values are calculated in pixels. The filter works by taking 6 passes (for 3d distance map): 2 along each axis (forward and backward). Each pass keeps a running minimum distance. For some reason, I preserve the sign if the distance. If the input mask is initially negative, the output distances will be negative. Distances maps can have inside (negative regions) and outsides (positive regions).

To create an instance of class vtkImageCityBlockDistance, simply invoke its constructor as follows

```
obj = vtkImageCityBlockDistance
```

35.18.2 Methods

The class vtkImageCityBlockDistance has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageCityBlockDistance class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageCityBlockDistance = obj.NewInstance ()`
- `vtkImageCityBlockDistance = obj.SafeDownCast (vtkObject o)`

35.19 vtkImageClip

35.19.1 Usage

vtkImageClip will make an image smaller. The output must have an image extent which is the subset of the input. The filter has two modes of operation: 1: By default, the data is not copied in this filter. Only the whole extent is modified. 2: If ClipDataOn is set, then you will get no more that the clipped extent.

To create an instance of class vtkImageClip, simply invoke its constructor as follows

```
obj = vtkImageClip
```

35.19.2 Methods

The class vtkImageClip has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageClip class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageClip = obj.NewInstance ()`
- `vtkImageClip = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutputWholeExtent (int extent[6], vtkInformation outInfo)` - The whole extent of the output has to be set explicitly.
- `obj.SetOutputWholeExtent (int minX, int maxX, int minY, int maxY, int minZ, int maxZ)` - The whole extent of the output has to be set explicitly.
- `obj.GetOutputWholeExtent (int extent[6])` - The whole extent of the output has to be set explicitly.
- `int = obj.GetOutputWholeExtent ()`
- `obj.ResetOutputWholeExtent ()`
- `obj.SetClipData (int )` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `int = obj.GetClipData ()` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `obj.ClipDataOn ()` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `obj.ClipDataOff ()` - By default, ClipData is off, and only the WholeExtent is modified. the data's extent may actually be larger. When this flag is on, the data extent will be no more than the OutputWholeExtent.
- `obj.SetOutputWholeExtent (int piece, int numPieces)` - Hack set output by piece

35.20 vtkImageConnector

35.20.1 Usage

vtkImageConnector is a helper class for connectivity filters. It is not meant to be used directly. It implements a stack and breadth first search necessary for some connectivity filters. Filtered axes sets the dimensionality of the neighbor comparison, and cannot be more than three dimensions. As implemented, only voxels which share faces are considered neighbors.

To create an instance of class vtkImageConnector, simply invoke its constructor as follows

```
obj = vtkImageConnector
```

35.20.2 Methods

The class vtkImageConnector has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageConnector class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageConnector = obj.NewInstance ()`
- `vtkImageConnector = obj.SafeDownCast (vtkObject o)`
- `obj.RemoveAllSeeds ()`
- `obj.SetConnectedValue (char )` - Values used by the MarkRegion method
- `char = obj.GetConnectedValue ()` - Values used by the MarkRegion method
- `obj.SetUnconnectedValue (char )` - Values used by the MarkRegion method
- `char = obj.GetUnconnectedValue ()` - Values used by the MarkRegion method
- `obj.MarkData (vtkImageData data, int dimensionality, int ext[6])` - Input a data of 0's and "UnconnectedValue"s. Seeds of this object are used to find connected pixels. All pixels connected to seeds are set to ConnectedValue. The data has to be unsigned char.

35.21 vtkImageConstantPad

35.21.1 Usage

vtkImageConstantPad changes the image extent of its input. Any pixels outside of the original image extent are filled with a constant value (default is 0.0).

To create an instance of class vtkImageConstantPad, simply invoke its constructor as follows

```
obj = vtkImageConstantPad
```

35.21.2 Methods

The class vtkImageConstantPad has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageConstantPad class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageConstantPad = obj.NewInstance ()`
- `vtkImageConstantPad = obj.SafeDownCast (vtkObject o)`
- `obj.SetConstant (double )` - Set/Get the pad value.
- `double = obj.GetConstant ()` - Set/Get the pad value.

35.22 vtkImageContinuousDilate3D

35.22.1 Usage

vtkImageContinuousDilate3D replaces a pixel with the maximum over an ellipsoidal neighborhood. If KernelSize of an axis is 1, no processing is done on that axis.

To create an instance of class vtkImageContinuousDilate3D, simply invoke its constructor as follows

```
obj = vtkImageContinuousDilate3D
```

35.22.2 Methods

The class vtkImageContinuousDilate3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageContinuousDilate3D class.

- `string = obj.GetClassName ()` - Construct an instance of vtkImageContinuousDilate3D filter. By default zero values are dilated.
- `int = obj.IsA (string name)` - Construct an instance of vtkImageContinuousDilate3D filter. By default zero values are dilated.
- `vtkImageContinuousDilate3D = obj.NewInstance ()` - Construct an instance of vtkImageContinuousDilate3D filter. By default zero values are dilated.
- `vtkImageContinuousDilate3D = obj.SafeDownCast (vtkObject o)` - Construct an instance of vtkImageContinuousDilate3D filter. By default zero values are dilated.
- `obj.SetKernelSize (int size0, int size1, int size2)` - This method sets the size of the neighborhood. It also sets the default middle of the neighborhood and computes the elliptical foot print.

35.23 vtkImageContinuousErode3D

35.23.1 Usage

vtkImageContinuousErode3D replaces a pixel with the minimum over an ellipsoidal neighborhood. If KernelSize of an axis is 1, no processing is done on that axis.

To create an instance of class vtkImageContinuousErode3D, simply invoke its constructor as follows

```
obj = vtkImageContinuousErode3D
```

35.23.2 Methods

The class vtkImageContinuousErode3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageContinuousErode3D class.

- `string = obj.GetClassName ()` - Construct an instance of vtkImageContinuousErode3D filter. By default zero values are eroded.
- `int = obj.IsA (string name)` - Construct an instance of vtkImageContinuousErode3D filter. By default zero values are eroded.
- `vtkImageContinuousErode3D = obj.NewInstance ()` - Construct an instance of vtkImageContinuousErode3D filter. By default zero values are eroded.

- `vtkImageContinuousErode3D = obj.SafeDownCast (vtkObject o)` - Construct an instance of vtkImageContinuousErode3D filter. By default zero values are eroded.
- `obj.SetKernelSize (int size0, int size1, int size2)` - This method sets the size of the neighborhood. It also sets the default middle of the neighborhood and computes the elliptical foot print.

35.24 vtkImageConvolve

35.24.1 Usage

vtkImageConvolve convolves the image with a 3D NxNxN kernel or a 2D NxN kernal. The output image is cropped to the same size as the input.

To create an instance of class vtkImageConvolve, simply invoke its constructor as follows

```
obj = vtkImageConvolve
```

35.24.2 Methods

The class vtkImageConvolve has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageConvolve class.

- `string = obj.GetClassName ()` - Construct an instance of vtkImageConvolve filter.
- `int = obj.IsA (string name)` - Construct an instance of vtkImageConvolve filter.
- `vtkImageConvolve = obj.NewInstance ()` - Construct an instance of vtkImageConvolve filter.
- `vtkImageConvolve = obj.SafeDownCast (vtkObject o)` - Construct an instance of vtkImageConvolve filter.
- `int = obj. GetKernelSize ()` - Get the kernel size
- `obj.SetKernel3x3 (double kernel[9])` - Set the kernel to be a given 3x3 or 5x5 or 7x7 kernel.
- `obj.SetKernel5x5 (double kernel[25])` - Set the kernel to be a given 3x3 or 5x5 or 7x7 kernel.
- `obj.GetKernel3x3 (double kernel[9])` - Return an array that contains the kernel.
- `obj.GetKernel5x5 (double kernel[25])` - Return an array that contains the kernel.
- `obj.SetKernel3x3x3 (double kernel[27])` - Set the kernel to be a 3x3x3 or 5x5x5 or 7x7x7 kernel.
- `obj.GetKernel3x3x3 (double kernel[27])` - Return an array that contains the kernel

35.25 vtkImageCorrelation

35.25.1 Usage

vtkImageCorrelation finds the correlation between two data sets. SetDimensionality determines whether the Correlation will be 3D, 2D or 1D. The default is a 2D Correlation. The Output type will be double. The output size will match the size of the first input. The second input is considered the correlation kernel.

To create an instance of class vtkImageCorrelation, simply invoke its constructor as follows

```
obj = vtkImageCorrelation
```

35.25.2 Methods

The class vtkImageCorrelation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageCorrelation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageCorrelation = obj.NewInstance ()`
- `vtkImageCorrelation = obj.SafeDownCast (vtkObject o)`
- `obj.SetDimensionality (int )` - Determines how the input is interpreted (set of 2d slices ...). The default is 2.
- `int = obj.GetDimensionalityMinValue ()` - Determines how the input is interpreted (set of 2d slices ...). The default is 2.
- `int = obj.GetDimensionalityMaxValue ()` - Determines how the input is interpreted (set of 2d slices ...). The default is 2.
- `int = obj.GetDimensionality ()` - Determines how the input is interpreted (set of 2d slices ...). The default is 2.
- `obj.SetInput1 (vtkDataObject in)` - Set the correlation kernel.
- `obj.SetInput2 (vtkDataObject in)`

35.26 vtkImageCursor3D

35.26.1 Usage

vtkImageCursor3D will draw a cursor on a 2d image or 3d volume.

To create an instance of class vtkImageCursor3D, simply invoke its constructor as follows

```
obj = vtkImageCursor3D
```

35.26.2 Methods

The class vtkImageCursor3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageCursor3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageCursor3D = obj.NewInstance ()`
- `vtkImageCursor3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetCursorPosition (double , double , double )` - Sets/Gets the center point of the 3d cursor.
- `obj.SetCursorPosition (double  a[3])` - Sets/Gets the center point of the 3d cursor.
- `double = obj. GetCursorPosition ()` - Sets/Gets the center point of the 3d cursor.

- `obj.SetCursorValue (double )` - Sets/Gets what pixel value to draw the cursor in.
- `double = obj.GetCursorValue ()` - Sets/Gets what pixel value to draw the cursor in.
- `obj.SetCursorRadius (int )` - Sets/Gets the radius of the cursor. The radius determines how far the axis lines project out from the cursors center.
- `int = obj.GetCursorRadius ()` - Sets/Gets the radius of the cursor. The radius determines how far the axis lines project out from the cursors center.

35.27 vtkImageDataStreamer

35.27.1 Usage

To satisfy a request, this filter calls update on its input many times with smaller update extents. All processing up stream streams smaller pieces.

To create an instance of class vtkImageDataStreamer, simply invoke its constructor as follows

```
obj = vtkImageDataStreamer
```

35.27.2 Methods

The class vtkImageDataStreamer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageDataStreamer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageDataStreamer = obj.NewInstance ()`
- `vtkImageDataStreamer = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfStreamDivisions (int )` - Set how many pieces to divide the input into. void SetNumberOfStreamDivisions(int num); int GetNumberOfStreamDivisions();
- `int = obj.GetNumberOfStreamDivisions ()` - Set how many pieces to divide the input into. void SetNumberOfStreamDivisions(int num); int GetNumberOfStreamDivisions();
- `obj.Update ()`
- `obj.UpdateWholeExtent ()`
- `obj.SetExtentTranslator (vtkExtentTranslator )` - Get the extent translator that will be used to split the requests
- `vtkExtentTranslator = obj.GetExtentTranslator ()` - Get the extent translator that will be used to split the requests

35.28 vtkImageDecomposeFilter

35.28.1 Usage

This superclass molds the vtkImageIterateFilter superclass so it iterates over the axes. The filter uses dimensionality to determine how many axes to execute (starting from x). The filter also provides convenience methods for permuting information retrieved from input, output and vtkImageData.

To create an instance of class vtkImageDecomposeFilter, simply invoke its constructor as follows

```
obj = vtkImageDecomposeFilter
```

35.28.2 Methods

The class vtkImageDecomposeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageDecomposeFilter class.

- `string = obj.GetClassName ()` - Construct an instance of vtkImageDecomposeFilter filter with default dimensionality 3.
- `int = obj.IsA (string name)` - Construct an instance of vtkImageDecomposeFilter filter with default dimensionality 3.
- `vtkImageDecomposeFilter = obj.NewInstance ()` - Construct an instance of vtkImageDecomposeFilter filter with default dimensionality 3.
- `vtkImageDecomposeFilter = obj.SafeDownCast (vtkObject o)` - Construct an instance of vtkImageDecomposeFilter filter with default dimensionality 3.
- `obj.SetDimensionality (int dim)` - Dimensionality is the number of axes which are considered during execution. To process images dimensionality would be set to 2.
- `int = obj.GetDimensionality ()` - Dimensionality is the number of axes which are considered during execution. To process images dimensionality would be set to 2.

35.29 vtkImageDifference

35.29.1 Usage

vtkImageDifference takes two rgb unsigned char images and compares them. It allows the images to be slightly different. If AllowShift is on, then each pixel can be shifted by one pixel. Threshold is the allowable error for each pixel.

This is not a symetric filter and the difference computed is not symetric when AllowShift is on. Specifically in that case a pixel in SetImage input will be compared to the matching pixel in the input as well as to the input's eight connected neighbors. BUT... the opposite is not true. So for example if a valid image (SetImage) has a single white pixel in it, it will not find a match in the input image if the input image is black (because none of the nine suspect pixels are white). In contrast, if there is a single white pixel in the input image and the valid image (SetImage) is all black it will match with no error because all it has to do is find black pixels and even though the input image has a white pixel, its neighbors are not white.

To create an instance of class vtkImageDifference, simply invoke its constructor as follows

```
obj = vtkImageDifference
```

35.29.2 Methods

The class vtkImageDifference has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageDifference class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageDifference = obj.NewInstance ()`
- `vtkImageDifference = obj.SafeDownCast (vtkObject o)`
- `obj.SetImage (vtkDataObject image)` - Specify the Image to compare the input to.

- `vtkImageData = obj.GetImage ()` - Specify the Image to compare the input to.
- `double = obj.GetError (void )` - Return the total error in comparing the two images.
- `obj.GetError (double e)` - Return the total error in comparing the two images.
- `double = obj.GetThresholdedError (void )` - Return the total thresholded error in comparing the two images. The thresholded error is the error for a given pixel minus the threshold and clamped at a minimum of zero.
- `obj.GetThresholdedError (double e)` - Return the total thresholded error in comparing the two images. The thresholded error is the error for a given pixel minus the threshold and clamped at a minimum of zero.
- `obj.SetThreshold (int )` - Specify a threshold tolerance for pixel differences.
- `int = obj.GetThreshold ()` - Specify a threshold tolerance for pixel differences.
- `obj.SetAllowShift (int )` - Specify whether the comparison will allow a shift of one pixel between the images. If set, then the minimum difference between input images will be used to determine the difference. Otherwise, the difference is computed directly between pixels of identical row/column values.
- `int = obj.GetAllowShift ()` - Specify whether the comparison will allow a shift of one pixel between the images. If set, then the minimum difference between input images will be used to determine the difference. Otherwise, the difference is computed directly between pixels of identical row/column values.
- `obj.AllowShiftOn ()` - Specify whether the comparison will allow a shift of one pixel between the images. If set, then the minimum difference between input images will be used to determine the difference. Otherwise, the difference is computed directly between pixels of identical row/column values.
- `obj.AllowShiftOff ()` - Specify whether the comparison will allow a shift of one pixel between the images. If set, then the minimum difference between input images will be used to determine the difference. Otherwise, the difference is computed directly between pixels of identical row/column values.
- `obj.SetAveraging (int )` - Specify whether the comparison will include comparison of averaged 3x3 data between the images. For graphics renderings you normally would leave this on. For imaging operations it should be off.
- `int = obj.GetAveraging ()` - Specify whether the comparison will include comparison of averaged 3x3 data between the images. For graphics renderings you normally would leave this on. For imaging operations it should be off.
- `obj.AveragingOn ()` - Specify whether the comparison will include comparison of averaged 3x3 data between the images. For graphics renderings you normally would leave this on. For imaging operations it should be off.
- `obj.AveragingOff ()` - Specify whether the comparison will include comparison of averaged 3x3 data between the images. For graphics renderings you normally would leave this on. For imaging operations it should be off.

35.30 vtkImageDilateErode3D

35.30.1 Usage

vtkImageDilateErode3D will dilate one value and erode another. It uses an elliptical foot print, and only erodes/dilates on the boundary of the two values. The filter is restricted to the X, Y, and Z axes for now. It can degenerate to a 2 or 1 dimensional filter by setting the kernel size to 1 for a specific axis.

To create an instance of class vtkImageDilateErode3D, simply invoke its constructor as follows

```
obj = vtkImageDilateErode3D
```

35.30.2 Methods

The class vtkImageDilateErode3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageDilateErode3D class.

- `string = obj.GetClassName ()` - Construct an instance of vtkImageDilateErode3D filter. By default zero values are dilated.
- `int = obj.IsA (string name)` - Construct an instance of vtkImageDilateErode3D filter. By default zero values are dilated.
- `vtkImageDilateErode3D = obj.NewInstance ()` - Construct an instance of vtkImageDilateErode3D filter. By default zero values are dilated.
- `vtkImageDilateErode3D = obj.SafeDownCast (vtkObject o)` - Construct an instance of vtkImageDilateErode3D filter. By default zero values are dilated.
- `obj.SetKernelSize (int size0, int size1, int size2)` - This method sets the size of the neighborhood. It also sets the default middle of the neighborhood and computes the elliptical foot print.
- `obj.SetDilateValue (double )` - Set/Get the Dilate and Erode values to be used by this filter.
- `double = obj.GetDilateValue ()` - Set/Get the Dilate and Erode values to be used by this filter.
- `obj.SetErodeValue (double )` - Set/Get the Dilate and Erode values to be used by this filter.
- `double = obj.GetErodeValue ()` - Set/Get the Dilate and Erode values to be used by this filter.

35.31 vtkImageDivergence

35.31.1 Usage

vtkImageDivergence takes a 3D vector field and creates a scalar field which which represents the rate of change of the vector field. The definition of Divergence: Given V = P(x,y,z), Q(x,y,z), R(x,y,z), Divergence = dP/dx + dQ/dy + dR/dz.

To create an instance of class vtkImageDivergence, simply invoke its constructor as follows

```
obj = vtkImageDivergence
```

35.31.2 Methods

The class vtkImageDivergence has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageDivergence class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageDivergence = obj.NewInstance ()`
- `vtkImageDivergence = obj.SafeDownCast (vtkObject o)`

35.32 vtkImageDotProduct

35.32.1 Usage

vtkImageDotProduct interprets the scalar components of two images as vectors and takes the dot product vector by vector (pixel by pixel).

To create an instance of class vtkImageDotProduct, simply invoke its constructor as follows

```
obj = vtkImageDotProduct
```

35.32.2 Methods

The class vtkImageDotProduct has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageDotProduct class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageDotProduct = obj.NewInstance ()`
- `vtkImageDotProduct = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput1 (vtkDataObject in)` - Set the two inputs to this filter
- `obj.SetInput2 (vtkDataObject in)`

35.33 vtkImageEllipsoidSource

35.33.1 Usage

vtkImageEllipsoidSource creates a binary image of a ellipsoid. It was created as an example of a simple source, and to test the mask filter. It is also used internally in vtkImageDilateErode3D.

To create an instance of class vtkImageEllipsoidSource, simply invoke its constructor as follows

```
obj = vtkImageEllipsoidSource
```

35.33.2 Methods

The class vtkImageEllipsoidSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageEllipsoidSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageEllipsoidSource = obj.NewInstance ()`
- `vtkImageEllipsoidSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetWholeExtent (int extent[6])` - Set/Get the extent of the whole output image.
- `obj.SetWholeExtent (int minX, int maxX, int minY, int maxY, int minZ, int maxZ)` - Set/Get the extent of the whole output image.
- `obj.GetWholeExtent (int extent[6])` - Set/Get the extent of the whole output image.
- `int = obj.GetWholeExtent ()` - Set/Get the center of the ellipsoid.
- `obj.SetCenter (double , double , double )` - Set/Get the center of the ellipsoid.
- `obj.SetCenter (double  a[3])` - Set/Get the center of the ellipsoid.
- `double = obj. GetCenter ()` - Set/Get the center of the ellipsoid.
- `obj.SetRadius (double , double , double )` - Set/Get the radius of the ellipsoid.
- `obj.SetRadius (double  a[3])` - Set/Get the radius of the ellipsoid.
- `double = obj. GetRadius ()` - Set/Get the radius of the ellipsoid.
- `obj.SetInValue (double )` - Set/Get the inside pixel values.
- `double = obj.GetInValue ()` - Set/Get the inside pixel values.
- `obj.SetOutValue (double )` - Set/Get the outside pixel values.
- `double = obj.GetOutValue ()` - Set/Get the outside pixel values.
- `obj.SetOutputScalarType (int )` - Set what type of scalar data this source should generate.
- `int = obj.GetOutputScalarType ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToFloat ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToDouble ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToLong ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedLong ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToInt ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedInt ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToShort ()` - Set what type of scalar data this source should generate.

- `obj.SetOutputScalarTypeToUnsignedShort ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToChar ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedChar ()`

35.34 vtkImageEuclideanDistance

35.34.1 Usage

vtkImageEuclideanDistance implements the Euclidean DT using Saito's algorithm. The distance map produced contains the square of the Euclidean distance values.

The algorithm has a o(n$\hat{}$(D+1)) complexity over nxnx...xn images in D dimensions. It is very efficient on relatively small images. Cuisenaire's algorithms should be used instead if n ¿¿ 500. These are not implemented yet.

For the special case of images where the slice-size is a multiple of 2N$\hat{}$ with a large N (typically for 256x256 slices), Saito's algorithm encounters a lot of cache conflicts during the 3rd iteration which can slow it very significantly. In that case, one should use ::SetAlgorithmToSaitoCached() instead for better performance.

References:

T. Saito and J.I. Toriwaki. New algorithms for Euclidean distance transformations of an n-dimensional digitised picture with applications. Pattern Recognition, 27(11). pp. 1551–1565, 1994.

O. Cuisenaire. Distance Transformation: fast algorithms and applications to medical image processing. PhD Thesis, Universite catholique de Louvain, October 1999. http://ltswww.epfl.ch/ cuisenai/papers/oc_thesis.pdf

To create an instance of class vtkImageEuclideanDistance, simply invoke its constructor as follows

```
obj = vtkImageEuclideanDistance
```

35.34.2 Methods

The class vtkImageEuclideanDistance has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageEuclideanDistance class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageEuclideanDistance = obj.NewInstance ()`
- `vtkImageEuclideanDistance = obj.SafeDownCast (vtkObject o)`
- `int = obj.SplitExtent (int splitExt[6], int startExt[6], int num, int total)` - Used internally for streaming and threads. Splits output update extent into num pieces. This method needs to be called num times. Results must not overlap for consistent starting extent. Subclass can override this method. This method returns the number of peices resulting from a successful split. This can be from 1 to "total". If 1 is returned, the extent cannot be split.
- `obj.SetInitialize (int )` - Used to set all non-zero voxels to MaximumDistance before starting the distance transformation. Setting Initialize off keeps the current value in the input image as starting point. This allows to superimpose several distance maps.
- `int = obj.GetInitialize ()` - Used to set all non-zero voxels to MaximumDistance before starting the distance transformation. Setting Initialize off keeps the current value in the input image as starting point. This allows to superimpose several distance maps.

- `obj.InitializeOn ()` - Used to set all non-zero voxels to MaximumDistance before starting the distance transformation. Setting Initialize off keeps the current value in the input image as starting point. This allows to superimpose several distance maps.
- `obj.InitializeOff ()` - Used to set all non-zero voxels to MaximumDistance before starting the distance transformation. Setting Initialize off keeps the current value in the input image as starting point. This allows to superimpose several distance maps.
- `obj.SetConsiderAnisotropy (int )` - Used to define whether Spacing should be used in the computation of the distances
- `int = obj.GetConsiderAnisotropy ()` - Used to define whether Spacing should be used in the computation of the distances
- `obj.ConsiderAnisotropyOn ()` - Used to define whether Spacing should be used in the computation of the distances
- `obj.ConsiderAnisotropyOff ()` - Used to define whether Spacing should be used in the computation of the distances
- `obj.SetMaximumDistance (double )` - Any distance bigger than this-¿MaximumDistance will not ne computed but set to this-¿MaximumDistance instead.
- `double = obj.GetMaximumDistance ()` - Any distance bigger than this-¿MaximumDistance will not ne computed but set to this-¿MaximumDistance instead.
- `obj.SetAlgorithm (int )` - Selects a Euclidean DT algorithm. 1. Saito 2. Saito-cached More algorithms will be added later on.
- `int = obj.GetAlgorithm ()` - Selects a Euclidean DT algorithm. 1. Saito 2. Saito-cached More algorithms will be added later on.
- `obj.SetAlgorithmToSaito ()` - Selects a Euclidean DT algorithm. 1. Saito 2. Saito-cached More algorithms will be added later on.
- `obj.SetAlgorithmToSaitoCached ()`

35.35 vtkImageEuclideanToPolar

35.35.1 Usage

For each pixel with vector components x,y, this filter outputs theta in component0, and radius in component1.
To create an instance of class vtkImageEuclideanToPolar, simply invoke its constructor as follows

```
obj = vtkImageEuclideanToPolar
```

35.35.2 Methods

The class vtkImageEuclideanToPolar has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageEuclideanToPolar class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageEuclideanToPolar = obj.NewInstance ()`
- `vtkImageEuclideanToPolar = obj.SafeDownCast (vtkObject o)`

- `obj.SetThetaMaximum (double )` - Theta is an angle. Maximum specifies when it maps back to 0. ThetaMaximum defaults to 255 instead of 2PI, because unsigned char is expected as input. The output type must be the same as input type.
- `double = obj.GetThetaMaximum ()` - Theta is an angle. Maximum specifies when it maps back to 0. ThetaMaximum defaults to 255 instead of 2PI, because unsigned char is expected as input. The output type must be the same as input type.

35.36 vtkImageExport

35.36.1 Usage

vtkImageExport provides a way of exporting image data at the end of a pipeline to a third-party system or to a simple C array. Applications can use this to get direct access to the image data in memory. A callback interface is provided to allow connection of the VTK pipeline to a third-party pipeline. This interface conforms to the interface of vtkImageImport. In Python it is possible to use this class to write the image data into a python string that has been pre-allocated to be the correct size.

To create an instance of class vtkImageExport, simply invoke its constructor as follows

```
obj = vtkImageExport
```

35.36.2 Methods

The class vtkImageExport has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageExport class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageExport = obj.NewInstance ()`
- `vtkImageExport = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDataMemorySize ()` - Get the number of bytes required for the output C array.
- `obj.GetDataDimensions (int ptr)` - Get the (x,y,z) index dimensions of the data. Please note that C arrays are indexed in decreasing order, i.e. array[z][y][x].
- `int = obj.GetDataDimensions ()` - Get the number of scalar components of the data. Please note that when you index into a C array, the scalar component index comes last, i.e. array[z][y][x][c].
- `int = obj.GetDataNumberOfScalarComponents ()` - Get the number of scalar components of the data. Please note that when you index into a C array, the scalar component index comes last, i.e. array[z][y][x][c].
- `int = obj.GetDataScalarType ()` - Get the scalar type of the data. The scalar type of the C array must match the scalar type of the data.
- `string = obj.GetDataScalarTypeAsString ()` - Get miscellaneous additional information about the data.
- `int = obj.GetDataExtent ()` - Get miscellaneous additional information about the data.
- `obj.GetDataExtent (int ptr)` - Get miscellaneous additional information about the data.
- `double = obj.GetDataSpacing ()` - Get miscellaneous additional information about the data.

- `obj.GetDataSpacing (double ptr)` - Get miscellaneous additional information about the data.
- `double = obj.GetDataOrigin ()` - Get miscellaneous additional information about the data.
- `obj.GetDataOrigin (double ptr)` - Get miscellaneous additional information about the data.
- `obj.ImageLowerLeftOn ()` - Set/Get whether the data goes to the exported memory starting in the lower left corner or upper left corner. Default: On. When this flag is Off, the image will be flipped vertically before it is exported. WARNING: this flag is used only with the Export() method, it is ignored by GetPointerToData().
- `obj.ImageLowerLeftOff ()` - Set/Get whether the data goes to the exported memory starting in the lower left corner or upper left corner. Default: On. When this flag is Off, the image will be flipped vertically before it is exported. WARNING: this flag is used only with the Export() method, it is ignored by GetPointerToData().
- `int = obj.GetImageLowerLeft ()` - Set/Get whether the data goes to the exported memory starting in the lower left corner or upper left corner. Default: On. When this flag is Off, the image will be flipped vertically before it is exported. WARNING: this flag is used only with the Export() method, it is ignored by GetPointerToData().
- `obj.SetImageLowerLeft (int )` - Set/Get whether the data goes to the exported memory starting in the lower left corner or upper left corner. Default: On. When this flag is Off, the image will be flipped vertically before it is exported. WARNING: this flag is used only with the Export() method, it is ignored by GetPointerToData().
- `obj.Export ()` - The main interface: update the pipeline and export the image to the memory pointed to by SetExportVoidPointer(). You can also specify a void pointer when you call Export().

35.37 vtkImageExtractComponents

35.37.1 Usage

vtkImageExtractComponents takes an input with any number of components and outputs some of them. It does involve a copy of the data.

To create an instance of class vtkImageExtractComponents, simply invoke its constructor as follows

```
obj = vtkImageExtractComponents
```

35.37.2 Methods

The class vtkImageExtractComponents has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageExtractComponents class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageExtractComponents = obj.NewInstance ()`
- `vtkImageExtractComponents = obj.SafeDownCast (vtkObject o)`
- `obj.SetComponents (int c1)` - Set/Get the components to extract.
- `obj.SetComponents (int c1, int c2)` - Set/Get the components to extract.
- `obj.SetComponents (int c1, int c2, int c3)` - Set/Get the components to extract.

- `int = obj. GetComponents ()` - Set/Get the components to extract.
- `int = obj.GetNumberOfComponents ()` - Get the number of components to extract. This is set implicitly by the SetComponents() method.

35.38 vtkImageFFT

35.38.1 Usage

vtkImageFFT implements a fast Fourier transform. The input can have real or complex data in any components and data types, but the output is always complex doubles with real values in component0, and imaginary values in component1. The filter is fastest for images that have power of two sizes. The filter uses a butterfly fitlers for each prime factor of the dimension. This makes images with prime number dimensions (i.e. 17x17) much slower to compute. Multi dimensional (i.e volumes) FFT's are decomposed so that each axis executes in series.

To create an instance of class vtkImageFFT, simply invoke its constructor as follows

```
obj = vtkImageFFT
```

35.38.2 Methods

The class vtkImageFFT has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageFFT class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageFFT = obj.NewInstance ()`
- `vtkImageFFT = obj.SafeDownCast (vtkObject o)`
- `int = obj.SplitExtent (int splitExt[6], int startExt[6], int num, int total)` - Used internally for streaming and threads. Splits output update extent into num pieces. This method needs to be called num times. Results must not overlap for consistent starting extent. Subclass can override this method. This method returns the number of pieces resulting from a successful split. This can be from 1 to "total". If 1 is returned, the extent cannot be split.

35.39 vtkImageFlip

35.39.1 Usage

vtkImageFlip will reflect the data along the filtered axis. This filter is actually a thin wrapper around vtkImageReslice.

To create an instance of class vtkImageFlip, simply invoke its constructor as follows

```
obj = vtkImageFlip
```

35.39.2 Methods

The class vtkImageFlip has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageFlip class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageFlip = obj.NewInstance ()`
- `vtkImageFlip = obj.SafeDownCast (vtkObject o)`
- `obj.SetFilteredAxis (int )` - Specify which axis will be flipped. This must be an integer between 0 (for x) and 2 (for z). Initial value is 0.
- `int = obj.GetFilteredAxis ()` - Specify which axis will be flipped. This must be an integer between 0 (for x) and 2 (for z). Initial value is 0.
- `obj.SetFlipAboutOrigin (int )` - By default the image will be flipped about its center, and the Origin, Spacing and Extent of the output will be identical to the input. However, if you have a coordinate system associated with the image and you want to use the flip to convert +ve values along one axis to -ve values (and vice versa) then you actually want to flip the image about coordinate (0,0,0) instead of about the center of the image. This method will adjust the Origin of the output such that the flip occurs about (0,0,0). Note that this method only changes the Origin (and hence the coordinate system) the output data: the actual pixel values are the same whether or not this method is used. Also note that the Origin in this method name refers to (0,0,0) in the coordinate system associated with the image, it does not refer to the Origin ivar that is associated with a vtkImageData.
- `int = obj.GetFlipAboutOrigin ()` - By default the image will be flipped about its center, and the Origin, Spacing and Extent of the output will be identical to the input. However, if you have a coordinate system associated with the image and you want to use the flip to convert +ve values along one axis to -ve values (and vice versa) then you actually want to flip the image about coordinate (0,0,0) instead of about the center of the image. This method will adjust the Origin of the output such that the flip occurs about (0,0,0). Note that this method only changes the Origin (and hence the coordinate system) the output data: the actual pixel values are the same whether or not this method is used. Also note that the Origin in this method name refers to (0,0,0) in the coordinate system associated with the image, it does not refer to the Origin ivar that is associated with a vtkImageData.
- `obj.FlipAboutOriginOn ()` - By default the image will be flipped about its center, and the Origin, Spacing and Extent of the output will be identical to the input. However, if you have a coordinate system associated with the image and you want to use the flip to convert +ve values along one axis to -ve values (and vice versa) then you actually want to flip the image about coordinate (0,0,0) instead of about the center of the image. This method will adjust the Origin of the output such that the flip occurs about (0,0,0). Note that this method only changes the Origin (and hence the coordinate system) the output data: the actual pixel values are the same whether or not this method is used. Also note that the Origin in this method name refers to (0,0,0) in the coordinate system associated with the image, it does not refer to the Origin ivar that is associated with a vtkImageData.
- `obj.FlipAboutOriginOff ()` - By default the image will be flipped about its center, and the Origin, Spacing and Extent of the output will be identical to the input. However, if you have a coordinate system associated with the image and you want to use the flip to convert +ve values along one axis to -ve values (and vice versa) then you actually want to flip the image about coordinate (0,0,0) instead of about the center of the image. This method will adjust the Origin of the output such that the flip occurs about (0,0,0). Note that this method only changes the Origin (and hence the coordinate system) the output data: the actual pixel values are the same whether or not this method is used. Also note that the Origin in this method name refers to (0,0,0) in the coordinate system associated with the image, it does not refer to the Origin ivar that is associated with a vtkImageData.
- `obj.SetFilteredAxes (int axis)` - Keep the mis-named Axes variations around for compatibility with old scripts. Axis is singular, not plural...

- `int = obj.GetFilteredAxes ()` - PreserveImageExtentOff wasn't covered by test scripts and its implementation was broken. It is deprecated now and it has no effect (i.e. the ImageExtent is always preserved).
- `obj.SetPreserveImageExtent (int )` - PreserveImageExtentOff wasn't covered by test scripts and its implementation was broken. It is deprecated now and it has no effect (i.e. the ImageExtent is always preserved).
- `int = obj.GetPreserveImageExtent ()` - PreserveImageExtentOff wasn't covered by test scripts and its implementation was broken. It is deprecated now and it has no effect (i.e. the ImageExtent is always preserved).
- `obj.PreserveImageExtentOn ()` - PreserveImageExtentOff wasn't covered by test scripts and its implementation was broken. It is deprecated now and it has no effect (i.e. the ImageExtent is always preserved).
- `obj.PreserveImageExtentOff ()` - PreserveImageExtentOff wasn't covered by test scripts and its implementation was broken. It is deprecated now and it has no effect (i.e. the ImageExtent is always preserved).

35.40 vtkImageFourierCenter

35.40.1 Usage

Is used for dispaying images in frequency space. FFT converts spatial images into frequency space, but puts the zero frequency at the origin. This filter shifts the zero frequency to the center of the image. Input and output are assumed to be doubles.

To create an instance of class vtkImageFourierCenter, simply invoke its constructor as follows

```
obj = vtkImageFourierCenter
```

35.40.2 Methods

The class vtkImageFourierCenter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageFourierCenter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageFourierCenter = obj.NewInstance ()`
- `vtkImageFourierCenter = obj.SafeDownCast (vtkObject o)`

35.41 vtkImageFourierFilter

35.41.1 Usage

vtkImageFourierFilter is a class of filters that use complex numbers this superclass is a container for methods that manipulate these structure including fast Fourier transforms. Complex numbers may become a class. This should really be a helper class.

To create an instance of class vtkImageFourierFilter, simply invoke its constructor as follows

```
obj = vtkImageFourierFilter
```

35.41.2 Methods

The class vtkImageFourierFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageFourierFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageFourierFilter = obj.NewInstance ()`
- `vtkImageFourierFilter = obj.SafeDownCast (vtkObject o)`

35.42 vtkImageGaussianSmooth

35.42.1 Usage

vtkImageGaussianSmooth implements a convolution of the input image with a gaussian. Supports from one to three dimensional convolutions.

To create an instance of class vtkImageGaussianSmooth, simply invoke its constructor as follows

```
obj = vtkImageGaussianSmooth
```

35.42.2 Methods

The class vtkImageGaussianSmooth has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageGaussianSmooth class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageGaussianSmooth = obj.NewInstance ()`
- `vtkImageGaussianSmooth = obj.SafeDownCast (vtkObject o)`
- `obj.SetStandardDeviations (double , double , double )` - Sets/Gets the Standard deviation of the gaussian in pixel units.
- `obj.SetStandardDeviations (double  a[3])` - Sets/Gets the Standard deviation of the gaussian in pixel units.
- `obj.SetStandardDeviation (double std)` - Sets/Gets the Standard deviation of the gaussian in pixel units.
- `obj.SetStandardDeviations (double a, double b)` - Sets/Gets the Standard deviation of the gaussian in pixel units.
- `double = obj. GetStandardDeviations ()` - Sets/Gets the Standard deviation of the gaussian in pixel units.
- `obj.SetStandardDeviation (double a, double b)` - Sets/Gets the Standard deviation of the gaussian in pixel units. These methods are provided for compatibility with old scripts
- `obj.SetStandardDeviation (double a, double b, double c)` - Sets/Gets the Radius Factors of the gaussian (no unit). The radius factors determine how far out the gaussian kernel will go before being clamped to zero.

- `obj.SetRadiusFactors (double , double , double )` - Sets/Gets the Radius Factors of the gaussian (no unit). The radius factors determine how far out the gaussian kernel will go before being clamped to zero.
- `obj.SetRadiusFactors (double  a[3])` - Sets/Gets the Radius Factors of the gaussian (no unit). The radius factors determine how far out the gaussian kernel will go before being clamped to zero.
- `obj.SetRadiusFactors (double f, double f2)` - Sets/Gets the Radius Factors of the gaussian (no unit). The radius factors determine how far out the gaussian kernel will go before being clamped to zero.
- `obj.SetRadiusFactor (double f)` - Sets/Gets the Radius Factors of the gaussian (no unit). The radius factors determine how far out the gaussian kernel will go before being clamped to zero.
- `double = obj. GetRadiusFactors ()` - Sets/Gets the Radius Factors of the gaussian (no unit). The radius factors determine how far out the gaussian kernel will go before being clamped to zero.
- `obj.SetDimensionality (int )` - Set/Get the dimensionality of this filter. This determines whether a one, two, or three dimensional gaussian is performed.
- `int = obj.GetDimensionality ()` - Set/Get the dimensionality of this filter. This determines whether a one, two, or three dimensional gaussian is performed.

35.43 vtkImageGaussianSource

35.43.1 Usage

vtkImageGaussianSource just produces images with pixel values determined by a Gaussian.

To create an instance of class vtkImageGaussianSource, simply invoke its constructor as follows

```
obj = vtkImageGaussianSource
```

35.43.2 Methods

The class vtkImageGaussianSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageGaussianSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageGaussianSource = obj.NewInstance ()`
- `vtkImageGaussianSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetWholeExtent (int xMinx, int xMax, int yMin, int yMax, int zMin, int zMax)` - Set/Get the extent of the whole output image.
- `obj.SetCenter (double , double , double )` - Set/Get the center of the Gaussian.
- `obj.SetCenter (double  a[3])` - Set/Get the center of the Gaussian.
- `double = obj. GetCenter ()` - Set/Get the center of the Gaussian.
- `obj.SetMaximum (double )` - Set/Get the Maximum value of the gaussian
- `double = obj.GetMaximum ()` - Set/Get the Maximum value of the gaussian
- `obj.SetStandardDeviation (double )` - Set/Get the standard deviation of the gaussian
- `double = obj.GetStandardDeviation ()` - Set/Get the standard deviation of the gaussian

35.44 vtkImageGradient

35.44.1 Usage

vtkImageGradient computes the gradient vector of an image. The vector results are stored as scalar components. The Dimensionality determines whether to perform a 2d or 3d gradient. The default is two dimensional XY gradient. OutputScalarType is always double. Gradient is computed using central differences.

To create an instance of class vtkImageGradient, simply invoke its constructor as follows

```
obj = vtkImageGradient
```

35.44.2 Methods

The class vtkImageGradient has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageGradient class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageGradient = obj.NewInstance ()`
- `vtkImageGradient = obj.SafeDownCast (vtkObject o)`
- `obj.SetDimensionality (int )` - Determines how the input is interpreted (set of 2d slices ...)
- `int = obj.GetDimensionalityMinValue ()` - Determines how the input is interpreted (set of 2d slices ...)
- `int = obj.GetDimensionalityMaxValue ()` - Determines how the input is interpreted (set of 2d slices ...)
- `int = obj.GetDimensionality ()` - Determines how the input is interpreted (set of 2d slices ...)
- `obj.SetHandleBoundaries (int )` - Get/Set whether to handle boundaries. If enabled, boundary pixels are treated as duplicated so that central differencing works for the boundary pixels. If disabled, the output whole extent of the image is reduced by one pixel.
- `int = obj.GetHandleBoundaries ()` - Get/Set whether to handle boundaries. If enabled, boundary pixels are treated as duplicated so that central differencing works for the boundary pixels. If disabled, the output whole extent of the image is reduced by one pixel.
- `obj.HandleBoundariesOn ()` - Get/Set whether to handle boundaries. If enabled, boundary pixels are treated as duplicated so that central differencing works for the boundary pixels. If disabled, the output whole extent of the image is reduced by one pixel.
- `obj.HandleBoundariesOff ()` - Get/Set whether to handle boundaries. If enabled, boundary pixels are treated as duplicated so that central differencing works for the boundary pixels. If disabled, the output whole extent of the image is reduced by one pixel.

35.45 vtkImageGradientMagnitude

35.45.1 Usage

vtkImageGradientMagnitude computes the gradient magnitude of an image. Setting the dimensionality determines whether the gradient is computed on 2D images, or 3D volumes. The default is two dimensional XY images.

To create an instance of class vtkImageGradientMagnitude, simply invoke its constructor as follows

```
obj = vtkImageGradientMagnitude
```

35.45.2 Methods

The class vtkImageGradientMagnitude has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageGradientMagnitude class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageGradientMagnitude = obj.NewInstance ()`
- `vtkImageGradientMagnitude = obj.SafeDownCast (vtkObject o)`
- `obj.SetHandleBoundaries (int )` - If "HandleBoundariesOn" then boundary pixels are duplicated So central differences can get values.
- `int = obj.GetHandleBoundaries ()` - If "HandleBoundariesOn" then boundary pixels are duplicated So central differences can get values.
- `obj.HandleBoundariesOn ()` - If "HandleBoundariesOn" then boundary pixels are duplicated So central differences can get values.
- `obj.HandleBoundariesOff ()` - If "HandleBoundariesOn" then boundary pixels are duplicated So central differences can get values.
- `obj.SetDimensionality (int )` - Determines how the input is interpreted (set of 2d slices ...)
- `int = obj.GetDimensionalityMinValue ()` - Determines how the input is interpreted (set of 2d slices ...)
- `int = obj.GetDimensionalityMaxValue ()` - Determines how the input is interpreted (set of 2d slices ...)
- `int = obj.GetDimensionality ()` - Determines how the input is interpreted (set of 2d slices ...)

35.46 vtkImageGridSource

35.46.1 Usage

vtkImageGridSource produces an image of a grid. The default output type is double.

To create an instance of class vtkImageGridSource, simply invoke its constructor as follows

```
obj = vtkImageGridSource
```

35.46.2 Methods

The class vtkImageGridSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageGridSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageGridSource = obj.NewInstance ()`
- `vtkImageGridSource = obj.SafeDownCast (vtkObject o)`

- `obj.SetGridSpacing (int , int , int )` - Set/Get the grid spacing in pixel units. Default (10,10,0). A value of zero means no grid.
- `obj.SetGridSpacing (int a[3])` - Set/Get the grid spacing in pixel units. Default (10,10,0). A value of zero means no grid.
- `int = obj. GetGridSpacing ()` - Set/Get the grid spacing in pixel units. Default (10,10,0). A value of zero means no grid.
- `obj.SetGridOrigin (int , int , int )` - Set/Get the grid origin, in ijk integer values. Default (0,0,0).
- `obj.SetGridOrigin (int a[3])` - Set/Get the grid origin, in ijk integer values. Default (0,0,0).
- `int = obj. GetGridOrigin ()` - Set/Get the grid origin, in ijk integer values. Default (0,0,0).
- `obj.SetLineValue (double )` - Set the grey level of the lines. Default 1.0.
- `double = obj.GetLineValue ()` - Set the grey level of the lines. Default 1.0.
- `obj.SetFillValue (double )` - Set the grey level of the fill. Default 0.0.
- `double = obj.GetFillValue ()` - Set the grey level of the fill. Default 0.0.
- `obj.SetDataScalarType (int )` - Set/Get the data type of pixels in the imported data. As a convenience, the OutputScalarType is set to the same value.
- `obj.SetDataScalarTypeToDouble ()` - Set/Get the data type of pixels in the imported data. As a convenience, the OutputScalarType is set to the same value.
- `obj.SetDataScalarTypeToInt ()` - Set/Get the data type of pixels in the imported data. As a convenience, the OutputScalarType is set to the same value.
- `obj.SetDataScalarTypeToShort ()` - Set/Get the data type of pixels in the imported data. As a convenience, the OutputScalarType is set to the same value.
- `obj.SetDataScalarTypeToUnsignedShort ()` - Set/Get the data type of pixels in the imported data. As a convenience, the OutputScalarType is set to the same value.
- `obj.SetDataScalarTypeToUnsignedChar ()` - Set/Get the data type of pixels in the imported data. As a convenience, the OutputScalarType is set to the same value.
- `int = obj.GetDataScalarType ()` - Set/Get the data type of pixels in the imported data. As a convenience, the OutputScalarType is set to the same value.
- `string = obj.GetDataScalarTypeAsString ()` - Set/Get the extent of the whole output image, Default: (0,255,0,255,0,0)
- `obj.SetDataExtent (int , int , int , int , int , int )` - Set/Get the extent of the whole output image, Default: (0,255,0,255,0,0)
- `obj.SetDataExtent (int a[6])` - Set/Get the extent of the whole output image, Default: (0,255,0,255,0,0)
- `int = obj. GetDataExtent ()` - Set/Get the extent of the whole output image, Default: (0,255,0,255,0,0)
- `obj.SetDataSpacing (double , double , double )` - Set/Get the pixel spacing.
- `obj.SetDataSpacing (double a[3])` - Set/Get the pixel spacing.
- `double = obj. GetDataSpacing ()` - Set/Get the pixel spacing.
- `obj.SetDataOrigin (double , double , double )` - Set/Get the origin of the data.
- `obj.SetDataOrigin (double a[3])` - Set/Get the origin of the data.
- `double = obj. GetDataOrigin ()` - Set/Get the origin of the data.

35.47 vtkImageHSIToRGB

35.47.1 Usage

For each pixel with hue, saturation and intensity components this filter outputs the color coded as red, green, blue. Output type must be the same as input type.

To create an instance of class vtkImageHSIToRGB, simply invoke its constructor as follows

```
obj = vtkImageHSIToRGB
```

35.47.2 Methods

The class vtkImageHSIToRGB has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageHSIToRGB class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageHSIToRGB = obj.NewInstance ()`
- `vtkImageHSIToRGB = obj.SafeDownCast (vtkObject o)`
- `obj.SetMaximum (double )` - Hue is an angle. Maximum specifies when it maps back to 0. Hue-Maximum defaults to 255 instead of 2PI, because unsigned char is expected as input. Maximum also specifies the maximum of the Saturation, and R, G, B.
- `double = obj.GetMaximum ()` - Hue is an angle. Maximum specifies when it maps back to 0. Hue-Maximum defaults to 255 instead of 2PI, because unsigned char is expected as input. Maximum also specifies the maximum of the Saturation, and R, G, B.

35.48 vtkImageHSVToRGB

35.48.1 Usage

For each pixel with hue, saturation and value components this filter outputs the color coded as red, green, blue. Output type must be the same as input type.

To create an instance of class vtkImageHSVToRGB, simply invoke its constructor as follows

```
obj = vtkImageHSVToRGB
```

35.48.2 Methods

The class vtkImageHSVToRGB has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageHSVToRGB class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageHSVToRGB = obj.NewInstance ()`
- `vtkImageHSVToRGB = obj.SafeDownCast (vtkObject o)`

- `obj.SetMaximum (double )` - Hue is an angle. Maximum specifies when it maps back to 0. Hue-Maximum defaults to 255 instead of 2PI, because unsigned char is expected as input. Maximum also specifies the maximum of the Saturation, and R, G, B.
- `double = obj.GetMaximum ()` - Hue is an angle. Maximum specifies when it maps back to 0. Hue-Maximum defaults to 255 instead of 2PI, because unsigned char is expected as input. Maximum also specifies the maximum of the Saturation, and R, G, B.

35.49 vtkImageHybridMedian2D

35.49.1 Usage

vtkImageHybridMedian2D is a median filter that preserves thin lines and corners. It operates on a 5x5 pixel neighborhood. It computes two values initially: the median of the + neighbors and the median of the x neighbors. It then computes the median of these two values plus the center pixel. This result of this second median is the output pixel value.

To create an instance of class vtkImageHybridMedian2D, simply invoke its constructor as follows

```
obj = vtkImageHybridMedian2D
```

35.49.2 Methods

The class vtkImageHybridMedian2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageHybridMedian2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageHybridMedian2D = obj.NewInstance ()`
- `vtkImageHybridMedian2D = obj.SafeDownCast (vtkObject o)`

35.50 vtkImageIdealHighPass

35.50.1 Usage

This filter only works on an image after it has been converted to frequency domain by a vtkImageFFT filter. A vtkImageRFFT filter can be used to convert the output back into the spatial domain. vtkImageIdealHighPass just sets a portion of the image to zero. The sharp cutoff in the frequence domain produces ringing in the spatial domain. Input and Output must be doubles. Dimensionality is set when the axes are set. Defaults to 2D on X and Y axes.

To create an instance of class vtkImageIdealHighPass, simply invoke its constructor as follows

```
obj = vtkImageIdealHighPass
```

35.50.2 Methods

The class vtkImageIdealHighPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageIdealHighPass class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkImageIdealHighPass = obj.NewInstance ()`
- `vtkImageIdealHighPass = obj.SafeDownCast (vtkObject o)`
- `obj.SetCutOff (double , double , double )` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetCutOff (double a[3])` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetXCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetYCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetZCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj. GetCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetXCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetYCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetZCutOff ()`

35.51 vtkImageIdealLowPass

35.51.1 Usage

This filter only works on an image after it has been converted to frequency domain by a vtkImageFFT filter. A vtkImageRFFT filter can be used to convert the output back into the spatial domain. vtkImageIdealLowPass just sets a portion of the image to zero. The result is an image with a lot of ringing. Input and Output must be doubles. Dimensionality is set when the axes are set. Defaults to 2D on X and Y axes.

To create an instance of class vtkImageIdealLowPass, simply invoke its constructor as follows

```
obj = vtkImageIdealLowPass
```

35.51.2 Methods

The class vtkImageIdealLowPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageIdealLowPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageIdealLowPass = obj.NewInstance ()`

- `vtkImageIdealLowPass = obj.SafeDownCast (vtkObject o)`
- `obj.SetCutOff (double , double , double )` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetCutOff (double a[3])` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetXCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetYCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `obj.SetZCutOff (double v)` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj. GetCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetXCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetYCutOff ()` - Set/Get the cutoff frequency for each axis. The values are specified in the order X, Y, Z, Time. Units: Cycles per world unit (as defined by the data spacing).
- `double = obj.GetZCutOff ()`

35.52 vtkImageImport

35.52.1 Usage

vtkImageImport provides methods needed to import image data from a source independent of VTK, such as a simple C array or a third-party pipeline. Note that the VTK convention is for the image voxel index (0,0,0) to be the lower-left corner of the image, while most 2D image formats use the upper-left corner. You can use vtkImageFlip to correct the orientation after the image has been loaded into VTK. Note that is also possible to import the raw data from a Python string instead of from a C array. The array applies on scalar point data only, not on cell data.

To create an instance of class vtkImageImport, simply invoke its constructor as follows

```
obj = vtkImageImport
```

35.52.2 Methods

The class vtkImageImport has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageImport class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageImport = obj.NewInstance ()`

- `vtkImageImport = obj.SafeDownCast (vtkObject o)`
- `obj.SetDataScalarType (int )` - Set/Get the data type of pixels in the imported data. This is used as the scalar type of the Output. Default: Short.
- `obj.SetDataScalarTypeToDouble ()` - Set/Get the data type of pixels in the imported data. This is used as the scalar type of the Output. Default: Short.
- `obj.SetDataScalarTypeToFloat ()` - Set/Get the data type of pixels in the imported data. This is used as the scalar type of the Output. Default: Short.
- `obj.SetDataScalarTypeToInt ()` - Set/Get the data type of pixels in the imported data. This is used as the scalar type of the Output. Default: Short.
- `obj.SetDataScalarTypeToShort ()` - Set/Get the data type of pixels in the imported data. This is used as the scalar type of the Output. Default: Short.
- `obj.SetDataScalarTypeToUnsignedShort ()` - Set/Get the data type of pixels in the imported data. This is used as the scalar type of the Output. Default: Short.
- `obj.SetDataScalarTypeToUnsignedChar ()` - Set/Get the data type of pixels in the imported data. This is used as the scalar type of the Output. Default: Short.
- `int = obj.GetDataScalarType ()` - Set/Get the data type of pixels in the imported data. This is used as the scalar type of the Output. Default: Short.
- `string = obj.GetDataScalarTypeAsString ()` - Set/Get the number of scalar components, for RGB images this must be 3. Default: 1.
- `obj.SetNumberOfScalarComponents (int )` - Set/Get the number of scalar components, for RGB images this must be 3. Default: 1.
- `int = obj.GetNumberOfScalarComponents ()` - Set/Get the number of scalar components, for RGB images this must be 3. Default: 1.
- `obj.SetDataExtent (int , int , int , int , int , int )` - Get/Set the extent of the data buffer. The dimensions of your data must be equal to (extent[1]-extent[0]+1) * (extent[3]-extent[2]+1) * (extent[5]-DataExtent[4]+1). For example, for a 2D image use (0,width-1, 0,height-1, 0,0).
- `obj.SetDataExtent (int  a[6])` - Get/Set the extent of the data buffer. The dimensions of your data must be equal to (extent[1]-extent[0]+1) * (extent[3]-extent[2]+1) * (extent[5]-DataExtent[4]+1). For example, for a 2D image use (0,width-1, 0,height-1, 0,0).
- `int = obj. GetDataExtent ()` - Get/Set the extent of the data buffer. The dimensions of your data must be equal to (extent[1]-extent[0]+1) * (extent[3]-extent[2]+1) * (extent[5]-DataExtent[4]+1). For example, for a 2D image use (0,width-1, 0,height-1, 0,0).
- `obj.SetDataExtentToWholeExtent ()` - Set/Get the spacing (typically in mm) between image voxels. Default: (1.0, 1.0, 1.0).
- `obj.SetDataSpacing (double , double , double )` - Set/Get the spacing (typically in mm) between image voxels. Default: (1.0, 1.0, 1.0).
- `obj.SetDataSpacing (double  a[3])` - Set/Get the spacing (typically in mm) between image voxels. Default: (1.0, 1.0, 1.0).
- `double = obj. GetDataSpacing ()` - Set/Get the spacing (typically in mm) between image voxels. Default: (1.0, 1.0, 1.0).
- `obj.SetDataOrigin (double , double , double )` - Set/Get the origin of the data, i.e. the coordinates (usually in mm) of voxel (0,0,0). Default: (0.0, 0.0, 0.0).

- `obj.SetDataOrigin (double a[3])` - Set/Get the origin of the data, i.e. the coordinates (usually in mm) of voxel (0,0,0). Default: (0.0, 0.0, 0.0).
- `double = obj. GetDataOrigin ()` - Set/Get the origin of the data, i.e. the coordinates (usually in mm) of voxel (0,0,0). Default: (0.0, 0.0, 0.0).
- `obj.SetWholeExtent (int , int , int , int , int , int )` - Get/Set the whole extent of the image. This is the largest possible extent. Set the DataExtent to the extent of the image in the buffer pointed to by the ImportVoidPointer.
- `obj.SetWholeExtent (int a[6])` - Get/Set the whole extent of the image. This is the largest possible extent. Set the DataExtent to the extent of the image in the buffer pointed to by the ImportVoidPointer.
- `int = obj. GetWholeExtent ()` - Get/Set the whole extent of the image. This is the largest possible extent. Set the DataExtent to the extent of the image in the buffer pointed to by the ImportVoidPointer.
- `obj.SetScalarArrayName (string )` - Set/get the scalar array name for this data set. Initial value is "scalars".
- `string = obj.GetScalarArrayName ()` - Set/get the scalar array name for this data set. Initial value is "scalars".
- `int = obj.InvokePipelineModifiedCallbacks ()` - Invoke the appropriate callbacks
- `obj.InvokeUpdateInformationCallbacks ()` - Invoke the appropriate callbacks
- `obj.InvokeExecuteInformationCallbacks ()` - Invoke the appropriate callbacks
- `obj.InvokeExecuteDataCallbacks ()` - Invoke the appropriate callbacks
- `obj.LegacyCheckWholeExtent ()` - Invoke the appropriate callbacks

35.53 vtkImageImportExecutive

35.53.1 Usage

vtkImageImportExecutive

To create an instance of class vtkImageImportExecutive, simply invoke its constructor as follows

```
obj = vtkImageImportExecutive
```

35.53.2 Methods

The class vtkImageImportExecutive has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageImportExecutive class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageImportExecutive = obj.NewInstance ()`
- `vtkImageImportExecutive = obj.SafeDownCast (vtkObject o)`

35.54 vtkImageIslandRemoval2D

35.54.1 Usage

vtkImageIslandRemoval2D computes the area of separate islands in a mask image. It removes any island that has less than AreaThreshold pixels. Output has the same ScalarType as input. It generates the whole 2D output image for any output request.

To create an instance of class vtkImageIslandRemoval2D, simply invoke its constructor as follows

```
obj = vtkImageIslandRemoval2D
```

35.54.2 Methods

The class vtkImageIslandRemoval2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageIslandRemoval2D class.

- `string = obj.GetClassName ()` - Constructor: Sets default filter to be identity.
- `int = obj.IsA (string name)` - Constructor: Sets default filter to be identity.
- `vtkImageIslandRemoval2D = obj.NewInstance ()` - Constructor: Sets default filter to be identity.
- `vtkImageIslandRemoval2D = obj.SafeDownCast (vtkObject o)` - Constructor: Sets default filter to be identity.
- `obj.SetAreaThreshold (int )` - Set/Get the cutoff area for removal
- `int = obj.GetAreaThreshold ()` - Set/Get the cutoff area for removal
- `obj.SetSquareNeighborhood (int )` - Set/Get whether to use 4 or 8 neighbors
- `int = obj.GetSquareNeighborhood ()` - Set/Get whether to use 4 or 8 neighbors
- `obj.SquareNeighborhoodOn ()` - Set/Get whether to use 4 or 8 neighbors
- `obj.SquareNeighborhoodOff ()` - Set/Get whether to use 4 or 8 neighbors
- `obj.SetIslandValue (double )` - Set/Get the value to remove.
- `double = obj.GetIslandValue ()` - Set/Get the value to remove.
- `obj.SetReplaceValue (double )` - Set/Get the value to put in the place of removed pixels.
- `double = obj.GetReplaceValue ()` - Set/Get the value to put in the place of removed pixels.

35.55 vtkImageIterateFilter

35.55.1 Usage

vtkImageIterateFilter is a filter superclass that supports calling execute multiple times per update. The largest hack/open issue is that the input and output caches are temporarily changed to "fool" the subclasses. I believe the correct solution is to pass the in and out cache to the subclasses methods as arguments. Now the data is passes. Can the caches be passed, and data retrieved from the cache?

To create an instance of class vtkImageIterateFilter, simply invoke its constructor as follows

```
obj = vtkImageIterateFilter
```

35.55.2 Methods

The class vtkImageIterateFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageIterateFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageIterateFilter = obj.NewInstance ()`
- `vtkImageIterateFilter = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetIteration ()` - Get which iteration is current being performed. Normally the user will not access this method.
- `int = obj.GetNumberOfIterations ()` - Get which iteration is current being performed. Normally the user will not access this method.

35.56 vtkImageLaplacian

35.56.1 Usage

vtkImageLaplacian computes the Laplacian (like a second derivative) of a scalar image. The operation is the same as taking the divergence after a gradient. Boundaries are handled, so the input is the same as the output. Dimensionality determines how the input regions are interpreted. (images, or volumes). The Dimensionality defaults to two.

To create an instance of class vtkImageLaplacian, simply invoke its constructor as follows

```
obj = vtkImageLaplacian
```

35.56.2 Methods

The class vtkImageLaplacian has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageLaplacian class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageLaplacian = obj.NewInstance ()`
- `vtkImageLaplacian = obj.SafeDownCast (vtkObject o)`
- `obj.SetDimensionality (int )` - Determines how the input is interpreted (set of 2d slices ...)
- `int = obj.GetDimensionalityMinValue ()` - Determines how the input is interpreted (set of 2d slices ...)
- `int = obj.GetDimensionalityMaxValue ()` - Determines how the input is interpreted (set of 2d slices ...)
- `int = obj.GetDimensionality ()` - Determines how the input is interpreted (set of 2d slices ...)

35.57 vtkImageLogarithmicScale

35.57.1 Usage

vtkImageLogarithmicScale passes each pixel through the function c*log(1+x). It also handles negative values with the function -c*log(1-x).

To create an instance of class vtkImageLogarithmicScale, simply invoke its constructor as follows

```
obj = vtkImageLogarithmicScale
```

35.57.2 Methods

The class vtkImageLogarithmicScale has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageLogarithmicScale class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageLogarithmicScale = obj.NewInstance ()`
- `vtkImageLogarithmicScale = obj.SafeDownCast (vtkObject o)`
- `obj.SetConstant (double )` - Set/Get the scale factor for the logarithmic function.
- `double = obj.GetConstant ()` - Set/Get the scale factor for the logarithmic function.

35.58 vtkImageLogic

35.58.1 Usage

vtkImageLogic implements basic logic operations. SetOperation is used to select the filter's behavior. The filter can take two or one input. Inputs must have the same type.

To create an instance of class vtkImageLogic, simply invoke its constructor as follows

```
obj = vtkImageLogic
```

35.58.2 Methods

The class vtkImageLogic has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageLogic class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageLogic = obj.NewInstance ()`
- `vtkImageLogic = obj.SafeDownCast (vtkObject o)`
- `obj.SetOperation (int )` - Set/Get the Operation to perform.
- `int = obj.GetOperation ()` - Set/Get the Operation to perform.
- `obj.SetOperationToAnd ()` - Set/Get the Operation to perform.

- `obj.SetOperationToOr ()` - Set/Get the Operation to perform.
- `obj.SetOperationToXor ()` - Set/Get the Operation to perform.
- `obj.SetOperationToNand ()` - Set/Get the Operation to perform.
- `obj.SetOperationToNor ()` - Set/Get the Operation to perform.
- `obj.SetOperationToNot ()` - Set/Get the Operation to perform.
- `obj.SetOutputTrueValue (double )` - Set the value to use for true in the output.
- `double = obj.GetOutputTrueValue ()` - Set the value to use for true in the output.
- `obj.SetInput1 (vtkDataObject input)` - Set the Input1 of this filter.
- `obj.SetInput2 (vtkDataObject input)` - Set the Input2 of this filter.

35.59 vtkImageLuminance

35.59.1 Usage

vtkImageLuminance calculates luminance from an rgb input.

To create an instance of class vtkImageLuminance, simply invoke its constructor as follows

```
obj = vtkImageLuminance
```

35.59.2 Methods

The class vtkImageLuminance has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageLuminance class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageLuminance = obj.NewInstance ()`
- `vtkImageLuminance = obj.SafeDownCast (vtkObject o)`

35.60 vtkImageMagnify

35.60.1 Usage

vtkImageMagnify maps each pixel of the input onto a nxmx... region of the output. Location (0,0,...) remains in the same place. The magnification occurs via pixel replication, or if Interpolate is on, by bilinear interpolation. Initially, interpolation is off and magnification factors are set to 1 in all directions.

To create an instance of class vtkImageMagnify, simply invoke its constructor as follows

```
obj = vtkImageMagnify
```

35.60.2 Methods

The class vtkImageMagnify has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMagnify class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMagnify = obj.NewInstance ()`
- `vtkImageMagnify = obj.SafeDownCast (vtkObject o)`
- `obj.SetMagnificationFactors (int , int , int )` - Set/Get the integer magnification factors in the i-j-k directions. Initially, factors are set to 1 in all directions.
- `obj.SetMagnificationFactors (int a[3])` - Set/Get the integer magnification factors in the i-j-k directions. Initially, factors are set to 1 in all directions.
- `int = obj. GetMagnificationFactors ()` - Set/Get the integer magnification factors in the i-j-k directions. Initially, factors are set to 1 in all directions.
- `obj.SetInterpolate (int )` - Turn interpolation on and off (pixel replication is used when off). Initially, interpolation is off.
- `int = obj.GetInterpolate ()` - Turn interpolation on and off (pixel replication is used when off). Initially, interpolation is off.
- `obj.InterpolateOn ()` - Turn interpolation on and off (pixel replication is used when off). Initially, interpolation is off.
- `obj.InterpolateOff ()` - Turn interpolation on and off (pixel replication is used when off). Initially, interpolation is off.

35.61 vtkImageMagnitude

35.61.1 Usage

vtkImageMagnitude takes the magnitude of the components.

To create an instance of class vtkImageMagnitude, simply invoke its constructor as follows

```
obj = vtkImageMagnitude
```

35.61.2 Methods

The class vtkImageMagnitude has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMagnitude class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMagnitude = obj.NewInstance ()`
- `vtkImageMagnitude = obj.SafeDownCast (vtkObject o)`

35.62 vtkImageMandelbrotSource

35.62.1 Usage

vtkImageMandelbrotSource creates an unsigned char image of the Mandelbrot set. The values in the image are the number of iterations it takes for the magnitude of the value to get over 2. The equation repeated is $z = z\hat{2} + C$ (z and C are complex). Initial value of z is zero, and the real value of C is mapped onto the x axis, and the imaginary value of C is mapped onto the Y Axis. I was thinking of extending this source to generate Julia Sets (initial value of Z varies). This would be 4 possible parameters to vary, but there are no more 4d images :(The third dimension (z axis) is the imaginary value of the initial value.

To create an instance of class vtkImageMandelbrotSource, simply invoke its constructor as follows

```
obj = vtkImageMandelbrotSource
```

35.62.2 Methods

The class vtkImageMandelbrotSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMandelbrotSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMandelbrotSource = obj.NewInstance ()`
- `vtkImageMandelbrotSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetWholeExtent (int extent[6])` - Set/Get the extent of the whole output Volume.
- `obj.SetWholeExtent (int minX, int maxX, int minY, int maxY, int minZ, int maxZ)` - Set/Get the extent of the whole output Volume.
- `int = obj. GetWholeExtent ()` - Set/Get the extent of the whole output Volume.
- `obj.SetConstantSize (int )` - This flag determines whether the Size or spacing of a data set remain constant (when extent is changed). By default, size remains constant.
- `int = obj.GetConstantSize ()` - This flag determines whether the Size or spacing of a data set remain constant (when extent is changed). By default, size remains constant.
- `obj.ConstantSizeOn ()` - This flag determines whether the Size or spacing of a data set remain constant (when extent is changed). By default, size remains constant.
- `obj.ConstantSizeOff ()` - This flag determines whether the Size or spacing of a data set remain constant (when extent is changed). By default, size remains constant.
- `obj.SetProjectionAxes (int x, int y, int z)` - Set the projection from the 4D space (4 parameters / 2 imaginary numbers) to the axes of the 3D Volume. 0=C_Real, 1=C_Imaginary, 2=X_Real, 4=X_Imaginary
- `obj.SetProjectionAxes (int a[3])` - Set the projection from the 4D space (4 parameters / 2 imaginary numbers) to the axes of the 3D Volume. 0=C_Real, 1=C_Imaginary, 2=X_Real, 4=X_Imaginary
- `int = obj. GetProjectionAxes ()` - Set the projection from the 4D space (4 parameters / 2 imaginary numbers) to the axes of the 3D Volume. 0=C_Real, 1=C_Imaginary, 2=X_Real, 4=X_Imaginary
- `obj.SetOriginCX (double , double , double , double )` - Imaginary and real value for C (constant in equation) and X (initial value).

- `obj.SetOriginCX (double a[4])` - Imaginary and real value for C (constant in equation) and X (initial value).
- `double = obj. GetOriginCX ()` - Imaginary and real value for C (constant in equation) and X (initial value). void SetOriginCX(double cReal, double cImag, double xReal, double xImag);
- `obj.SetSampleCX (double , double , double , double )` - Imaginary and real value for C (constant in equation) and X (initial value).
- `obj.SetSampleCX (double a[4])` - Imaginary and real value for C (constant in equation) and X (initial value).
- `double = obj. GetSampleCX ()` - Imaginary and real value for C (constant in equation) and X (initial value). void SetOriginCX(double cReal, double cImag, double xReal, double xImag);
- `obj.SetSizeCX (double cReal, double cImag, double xReal, double xImag)` - Just a different way of setting the sample. This sets the size of the 4D volume. SampleCX is computed from size and extent. Size is ignored when a dimension i 0 (collapsed).
- `double = obj.GetSizeCX ()` - Just a different way of setting the sample. This sets the size of the 4D volume. SampleCX is computed from size and extent. Size is ignored when a dimension i 0 (collapsed).
- `obj.GetSizeCX (double s[4])` - Just a different way of setting the sample. This sets the size of the 4D volume. SampleCX is computed from size and extent. Size is ignored when a dimension i 0 (collapsed).
- `obj.SetMaximumNumberOfIterations (short )` - The maximum number of cycles run to see if the value goes over 2
- `GetMaximumNumberOfIterationsMinValue = obj.()` - The maximum number of cycles run to see if the value goes over 2
- `GetMaximumNumberOfIterationsMaxValue = obj.()` - The maximum number of cycles run to see if the value goes over 2
- `short = obj.GetMaximumNumberOfIterations ()` - The maximum number of cycles run to see if the value goes over 2
- `obj.Zoom (double factor)` - Convienence for Viewer. Pan 3D volume relative to spacing. Zoom constant factor.
- `obj.Pan (double x, double y, double z)` - Convienence for Viewer. Pan 3D volume relative to spacing. Zoom constant factor.
- `obj.CopyOriginAndSample (vtkImageMandelbrotSource source)` - Convienence for Viewer. Copy the OriginCX and the SpacingCX. What about other parameters ???
- `obj.SetSubsampleRate (int )` - Set/Get a subsample rate.
- `int = obj.GetSubsampleRateMinValue ()` - Set/Get a subsample rate.
- `int = obj.GetSubsampleRateMaxValue ()` - Set/Get a subsample rate.
- `int = obj.GetSubsampleRate ()` - Set/Get a subsample rate.

35.63 vtkImageMapToColors

35.63.1 Usage

The vtkImageMapToColors filter will take an input image of any valid scalar type, and map the first component of the image through a lookup table. The result is an image of type VTK_UNSIGNED_CHAR. If the lookup table is not set, or is set to NULL, then the input data will be passed through if it is already of type VTK_UNSIGNED_CHAR.

To create an instance of class vtkImageMapToColors, simply invoke its constructor as follows

```
obj = vtkImageMapToColors
```

35.63.2 Methods

The class vtkImageMapToColors has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMapToColors class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMapToColors = obj.NewInstance ()`
- `vtkImageMapToColors = obj.SafeDownCast (vtkObject o)`
- `obj.SetLookupTable (vtkScalarsToColors )` - Set the lookup table.
- `vtkScalarsToColors = obj.GetLookupTable ()` - Set the lookup table.
- `obj.SetOutputFormat (int )` - Set the output format, the default is RGBA.
- `int = obj.GetOutputFormat ()` - Set the output format, the default is RGBA.
- `obj.SetOutputFormatToRGBA ()` - Set the output format, the default is RGBA.
- `obj.SetOutputFormatToRGB ()` - Set the output format, the default is RGBA.
- `obj.SetOutputFormatToLuminanceAlpha ()` - Set the output format, the default is RGBA.
- `obj.SetOutputFormatToLuminance ()` - Set the output format, the default is RGBA.
- `obj.SetActiveComponent (int )` - Set the component to map for multi-component images (default: 0)
- `int = obj.GetActiveComponent ()` - Set the component to map for multi-component images (default: 0)
- `obj.SetPassAlphaToOutput (int )` - Use the alpha component of the input when computing the alpha component of the output (useful when converting monochrome+alpha data to RGBA)
- `obj.PassAlphaToOutputOn ()` - Use the alpha component of the input when computing the alpha component of the output (useful when converting monochrome+alpha data to RGBA)
- `obj.PassAlphaToOutputOff ()` - Use the alpha component of the input when computing the alpha component of the output (useful when converting monochrome+alpha data to RGBA)
- `int = obj.GetPassAlphaToOutput ()` - Use the alpha component of the input when computing the alpha component of the output (useful when converting monochrome+alpha data to RGBA)
- `long = obj.GetMTime ()` - We need to check the modified time of the lookup table too.

35.64 vtkImageMapToRGBA

35.64.1 Usage

This filter has been replaced by vtkImageMapToColors, which provided additional features. Use vtkImageMapToColors instead.

To create an instance of class vtkImageMapToRGBA, simply invoke its constructor as follows

```
obj = vtkImageMapToRGBA
```

35.64.2 Methods

The class vtkImageMapToRGBA has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMapToRGBA class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMapToRGBA = obj.NewInstance ()`
- `vtkImageMapToRGBA = obj.SafeDownCast (vtkObject o)`

35.65 vtkImageMapToWindowLevelColors

35.65.1 Usage

The vtkImageMapToWindowLevelColors filter will take an input image of any valid scalar type, and map the first component of the image through a lookup table. This resulting color will be modulated with value obtained by a window / level operation. The result is an image of type VTK_UNSIGNED_CHAR. If the lookup table is not set, or is set to NULL, then the input data will be passed through if it is already of type UNSIGNED_CHAR.

To create an instance of class vtkImageMapToWindowLevelColors, simply invoke its constructor as follows

```
obj = vtkImageMapToWindowLevelColors
```

35.65.2 Methods

The class vtkImageMapToWindowLevelColors has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMapToWindowLevelColors class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMapToWindowLevelColors = obj.NewInstance ()`
- `vtkImageMapToWindowLevelColors = obj.SafeDownCast (vtkObject o)`
- `obj.SetWindow (double )` - Set / Get the Window to use -¿ modulation will be performed on the color based on (S - (L - W/2))/W where S is the scalar value, L is the level and W is the window.
- `double = obj.GetWindow ()` - Set / Get the Window to use -¿ modulation will be performed on the color based on (S - (L - W/2))/W where S is the scalar value, L is the level and W is the window.

- `obj.SetLevel (double )` - Set / Get the Level to use -¿ modulation will be performed on the color based on (S - (L - W/2))/W where S is the scalar value, L is the level and W is the window.
- `double = obj.GetLevel ()` - Set / Get the Level to use -¿ modulation will be performed on the color based on (S - (L - W/2))/W where S is the scalar value, L is the level and W is the window.

35.66 vtkImageMask

35.66.1 Usage

vtkImageMask combines a mask with an image. Non zero mask implies the output pixel will be the same as the image. If a mask pixel is zero, then the output pixel is set to "MaskedValue". The filter also has the option to pass the mask through a boolean not operation before processing the image. This reverses the passed and replaced pixels. The two inputs should have the same "WholeExtent". The mask input should be unsigned char, and the image scalar type is the same as the output scalar type.

To create an instance of class vtkImageMask, simply invoke its constructor as follows

```
obj = vtkImageMask
```

35.66.2 Methods

The class vtkImageMask has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMask class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMask = obj.NewInstance ()`
- `vtkImageMask = obj.SafeDownCast (vtkObject o)`
- `obj.SetMaskedOutputValue (int num, double v)` - SetGet the value of the output pixel replaced by mask.
- `obj.SetMaskedOutputValue (double v)` - SetGet the value of the output pixel replaced by mask.
- `obj.SetMaskedOutputValue (double v1, double v2)` - SetGet the value of the output pixel replaced by mask.
- `obj.SetMaskedOutputValue (double v1, double v2, double v3)` - SetGet the value of the output pixel replaced by mask.
- `int = obj.GetMaskedOutputValueLength ()` - Set/Get the alpha blending value for the mask The input image is assumed to be at alpha = 1.0 and the mask image uses this alpha to blend using an over operator.
- `obj.SetMaskAlpha (double )` - Set/Get the alpha blending value for the mask The input image is assumed to be at alpha = 1.0 and the mask image uses this alpha to blend using an over operator.
- `double = obj.GetMaskAlphaMinValue ()` - Set/Get the alpha blending value for the mask The input image is assumed to be at alpha = 1.0 and the mask image uses this alpha to blend using an over operator.
- `double = obj.GetMaskAlphaMaxValue ()` - Set/Get the alpha blending value for the mask The input image is assumed to be at alpha = 1.0 and the mask image uses this alpha to blend using an over operator.

- `double = obj.GetMaskAlpha ()` - Set/Get the alpha blending value for the mask The input image is assumed to be at alpha = 1.0 and the mask image uses this alpha to blend using an over operator.
- `obj.SetImageInput (vtkImageData in)` - Set the input to be masked.
- `obj.SetMaskInput (vtkImageData in)` - Set the mask to be used.
- `obj.SetNotMask (int )` - When Not Mask is on, the mask is passed through a boolean not before it is used to mask the image. The effect is to pass the pixels where the input mask is zero, and replace the pixels where the input value is non zero.
- `int = obj.GetNotMask ()` - When Not Mask is on, the mask is passed through a boolean not before it is used to mask the image. The effect is to pass the pixels where the input mask is zero, and replace the pixels where the input value is non zero.
- `obj.NotMaskOn ()` - When Not Mask is on, the mask is passed through a boolean not before it is used to mask the image. The effect is to pass the pixels where the input mask is zero, and replace the pixels where the input value is non zero.
- `obj.NotMaskOff ()` - When Not Mask is on, the mask is passed through a boolean not before it is used to mask the image. The effect is to pass the pixels where the input mask is zero, and replace the pixels where the input value is non zero.
- `obj.SetInput1 (vtkDataObject in)` - Set the two inputs to this filter
- `obj.SetInput2 (vtkDataObject in)`

35.67 vtkImageMaskBits

35.67.1 Usage

vtkImageMaskBits applies a bit-mask pattern to each component. The bit-mask can be applied using a variety of boolean bitwise operators.

To create an instance of class vtkImageMaskBits, simply invoke its constructor as follows

```
obj = vtkImageMaskBits
```

35.67.2 Methods

The class vtkImageMaskBits has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMaskBits class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMaskBits = obj.NewInstance ()`
- `vtkImageMaskBits = obj.SafeDownCast (vtkObject o)`
- `obj.SetMasks (int , int , int , int )` - Set/Get the bit-masks. Default is 0xffffffff.
- `obj.SetMasks (int  a[4])` - Set/Get the bit-masks. Default is 0xffffffff.
- `obj.SetMask (int mask)` - Set/Get the bit-masks. Default is 0xffffffff.
- `obj.SetMasks (int mask1, int mask2)` - Set/Get the bit-masks. Default is 0xffffffff.

- `obj.SetMasks (int mask1, int mask2, int mask3)` - Set/Get the bit-masks. Default is 0xffffffff.
- `int = obj. GetMasks ()` - Set/Get the bit-masks. Default is 0xffffffff.
- `obj.SetOperation (int )` - Set/Get the boolean operator. Default is AND.
- `int = obj.GetOperation ()` - Set/Get the boolean operator. Default is AND.
- `obj.SetOperationToAnd ()` - Set/Get the boolean operator. Default is AND.
- `obj.SetOperationToOr ()` - Set/Get the boolean operator. Default is AND.
- `obj.SetOperationToXor ()` - Set/Get the boolean operator. Default is AND.
- `obj.SetOperationToNand ()` - Set/Get the boolean operator. Default is AND.
- `obj.SetOperationToNor ()` - Set/Get the boolean operator. Default is AND.

35.68 vtkImageMathematics

35.68.1 Usage

vtkImageMathematics implements basic mathematic operations SetOperation is used to select the filters behavior. The filter can take two or one input.

To create an instance of class vtkImageMathematics, simply invoke its constructor as follows

```
obj = vtkImageMathematics
```

35.68.2 Methods

The class vtkImageMathematics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMathematics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMathematics = obj.NewInstance ()`
- `vtkImageMathematics = obj.SafeDownCast (vtkObject o)`
- `obj.SetOperation (int )` - Set/Get the Operation to perform.
- `int = obj.GetOperation ()` - Set/Get the Operation to perform.
- `obj.SetOperationToAdd ()` - Set each pixel in the output image to the sum of the corresponding pixels in Input1 and Input2.
- `obj.SetOperationToSubtract ()` - Set each pixel in the output image to the difference of the corresponding pixels in Input1 and Input2 (output = Input1 - Input2).
- `obj.SetOperationToMultiply ()` - Set each pixel in the output image to the product of the corresponding pixels in Input1 and Input2.
- `obj.SetOperationToDivide ()` - Set each pixel in the output image to the quotient of the corresponding pixels in Input1 and Input2 (Output = Input1 / Input2).
- `obj.SetOperationToConjugate ()`

- `obj.SetOperationToComplexMultiply ()`
- `obj.SetOperationToInvert ()` - Set each pixel in the output image to 1 over the corresponding pixel in Input1 and Input2 (output = 1 / Input1). Input2 is not used.
- `obj.SetOperationToSin ()` - Set each pixel in the output image to the sine of the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToCos ()` - Set each pixel in the output image to the cosine of the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToExp ()` - Set each pixel in the output image to the exponential of the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToLog ()` - Set each pixel in the output image to the log of the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToAbsoluteValue ()` - Set each pixel in the output image to the absolute value of the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToSquare ()` - Set each pixel in the output image to the square of the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToSquareRoot ()` - Set each pixel in the output image to the square root of the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToMin ()` - Set each pixel in the output image to the minimum of the corresponding pixels in Input1 and Input2. (Output = min(Input1, Input2))
- `obj.SetOperationToMax ()` - Set each pixel in the output image to the maximum of the corresponding pixels in Input1 and Input2. (Output = max(Input1, Input2))
- `obj.SetOperationToATAN ()` - Set each pixel in the output image to the arctangent of the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToATAN2 ()`
- `obj.SetOperationToMultiplyByK ()` - Set each pixel in the output image to the product of ConstantK with the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToAddConstant ()` - Set each pixel in the output image to the product of ConstantC with the corresponding pixel in Input1. Input2 is not used.
- `obj.SetOperationToReplaceCByK ()` - Find every pixel in Input1 that equals ConstantC and set the corresponding pixels in the Output to ConstantK. Input2 is not used.
- `obj.SetConstantK (double )` - A constant used by some operations (typically multiplicative). Default is 1.
- `double = obj.GetConstantK ()` - A constant used by some operations (typically multiplicative). Default is 1.
- `obj.SetConstantC (double )` - A constant used by some operations (typically additive). Default is 0.
- `double = obj.GetConstantC ()` - A constant used by some operations (typically additive). Default is 0.
- `obj.SetDivideByZeroToC (int )` - How to handle divide by zero. Default is 0.
- `int = obj.GetDivideByZeroToC ()` - How to handle divide by zero. Default is 0.

- `obj.DivideByZeroToCOn ()` - How to handle divide by zero. Default is 0.
- `obj.DivideByZeroToCOff ()` - How to handle divide by zero. Default is 0.
- `obj.SetInput1 (vtkDataObject in)` - Set the two inputs to this filter. For some operations, the second input is not used.
- `obj.SetInput2 (vtkDataObject in)`

35.69 vtkImageMedian3D

35.69.1 Usage

vtkImageMedian3D a Median filter that replaces each pixel with the median value from a rectangular neighborhood around that pixel. Neighborhoods can be no more than 3 dimensional. Setting one axis of the neighborhood kernelSize to 1 changes the filter into a 2D median.

To create an instance of class vtkImageMedian3D, simply invoke its constructor as follows

```
obj = vtkImageMedian3D
```

35.69.2 Methods

The class vtkImageMedian3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMedian3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMedian3D = obj.NewInstance ()`
- `vtkImageMedian3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetKernelSize (int size0, int size1, int size2)` - This method sets the size of the neighborhood. It also sets the default middle of the neighborhood
- `int = obj.GetNumberOfElements ()` - Return the number of elements in the median mask

35.70 vtkImageMirrorPad

35.70.1 Usage

vtkImageMirrorPad makes an image larger by filling extra pixels with a mirror image of the original image (mirror at image boundaries).

To create an instance of class vtkImageMirrorPad, simply invoke its constructor as follows

```
obj = vtkImageMirrorPad
```

35.70.2 Methods

The class vtkImageMirrorPad has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMirrorPad class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkImageMirrorPad = obj.NewInstance ()`
- `vtkImageMirrorPad = obj.SafeDownCast (vtkObject o)`

35.71 vtkImageNoiseSource

35.71.1 Usage

vtkImageNoiseSource just produces images filled with noise. The only option now is uniform noise specified by a min and a max. There is one major problem with this source. Every time it executes, it will output different pixel values. This has important implications when a stream requests overlapping regions. The same pixels will have different values on different updates.

To create an instance of class vtkImageNoiseSource, simply invoke its constructor as follows

```
obj = vtkImageNoiseSource
```

35.71.2 Methods

The class vtkImageNoiseSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageNoiseSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageNoiseSource = obj.NewInstance ()`
- `vtkImageNoiseSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetMinimum (double )` - Set/Get the minimum and maximum values for the generated noise.
- `double = obj.GetMinimum ()` - Set/Get the minimum and maximum values for the generated noise.
- `obj.SetMaximum (double )` - Set/Get the minimum and maximum values for the generated noise.
- `double = obj.GetMaximum ()` - Set/Get the minimum and maximum values for the generated noise.
- `obj.SetWholeExtent (int xMinx, int xMax, int yMin, int yMax, int zMin, int zMax)` - Set how large of an image to generate.
- `obj.SetWholeExtent (int ext[6])`

35.72 vtkImageNonMaximumSuppression

35.72.1 Usage

vtkImageNonMaximumSuppression Sets to zero any pixel that is not a peak. If a pixel has a neighbor along the vector that has larger magnitude, the smaller pixel is set to zero. The filter takes two inputs: a magnitude and a vector. Output is magnitude information and is always in doubles. Typically this filter is used with vtkImageGradient and vtkImageGradientMagnitude as inputs.

To create an instance of class vtkImageNonMaximumSuppression, simply invoke its constructor as follows

```
obj = vtkImageNonMaximumSuppression
```

35.72.2 Methods

The class vtkImageNonMaximumSuppression has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageNonMaximumSuppression class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageNonMaximumSuppression = obj.NewInstance ()`
- `vtkImageNonMaximumSuppression = obj.SafeDownCast (vtkObject o)`
- `obj.SetMagnitudeInput (vtkImageData input)` - Set the magnitude and vector inputs.
- `obj.SetVectorInput (vtkImageData input)` - Set the magnitude and vector inputs.
- `obj.SetHandleBoundaries (int )` - If "HandleBoundariesOn" then boundary pixels are duplicated So central differences can get values.
- `int = obj.GetHandleBoundaries ()` - If "HandleBoundariesOn" then boundary pixels are duplicated So central differences can get values.
- `obj.HandleBoundariesOn ()` - If "HandleBoundariesOn" then boundary pixels are duplicated So central differences can get values.
- `obj.HandleBoundariesOff ()` - If "HandleBoundariesOn" then boundary pixels are duplicated So central differences can get values.
- `obj.SetDimensionality (int )` - Determines how the input is interpreted (set of 2d slices or a 3D volume)
- `int = obj.GetDimensionalityMinValue ()` - Determines how the input is interpreted (set of 2d slices or a 3D volume)
- `int = obj.GetDimensionalityMaxValue ()` - Determines how the input is interpreted (set of 2d slices or a 3D volume)
- `int = obj.GetDimensionality ()` - Determines how the input is interpreted (set of 2d slices or a 3D volume)

35.73 vtkImageNormalize

35.73.1 Usage

For each point, vtkImageNormalize normalizes the vector defined by the scalar components. If the magnitude of this vector is zero, the output vector is zero also.

To create an instance of class vtkImageNormalize, simply invoke its constructor as follows

```
obj = vtkImageNormalize
```

35.73.2 Methods

The class vtkImageNormalize has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageNormalize class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageNormalize = obj.NewInstance ()`
- `vtkImageNormalize = obj.SafeDownCast (vtkObject o)`

35.74 vtkImageOpenClose3D

35.74.1 Usage

vtkImageOpenClose3D performs opening or closing by having two vtkImageErodeDilates in series. The size of operation is determined by the method SetKernelSize, and the operator is an ellipse. OpenValue and CloseValue determine how the filter behaves. For binary images Opening and closing behaves as expected. Close value is first dilated, and then eroded. Open value is first eroded, and then dilated. Degenerate two dimensional opening/closing can be achieved by setting the one axis the 3D KernelSize to 1. Values other than open value and close value are not touched. This enables the filter to processes segmented images containing more than two tags.

To create an instance of class vtkImageOpenClose3D, simply invoke its constructor as follows

```
obj = vtkImageOpenClose3D
```

35.74.2 Methods

The class vtkImageOpenClose3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageOpenClose3D class.

- `string = obj.GetClassName ()` - Default open value is 0, and default close value is 255.
- `int = obj.IsA (string name)` - Default open value is 0, and default close value is 255.
- `vtkImageOpenClose3D = obj.NewInstance ()` - Default open value is 0, and default close value is 255.
- `vtkImageOpenClose3D = obj.SafeDownCast (vtkObject o)` - Default open value is 0, and default close value is 255.
- `long = obj.GetMTime ()` - This method considers the sub filters MTimes when computing this objects modified time.
- `obj.DebugOn ()` - Turn debugging output on. (in sub filters also)
- `obj.DebugOff ()` - Turn debugging output on. (in sub filters also)
- `obj.Modified ()` - Pass modified message to sub filters.
- `obj.SetKernelSize (int size0, int size1, int size2)` - Selects the size of gaps or objects removed.

- `obj.SetOpenValue (double value)` - Determines the value that will opened. Open value is first eroded, and then dilated.
- `double = obj.GetOpenValue ()` - Determines the value that will opened. Open value is first eroded, and then dilated.
- `obj.SetCloseValue (double value)` - Determines the value that will closed. Close value is first dilated, and then eroded
- `double = obj.GetCloseValue ()` - Determines the value that will closed. Close value is first dilated, and then eroded
- `vtkImageDilateErode3D = obj.GetFilter0 ()` - Needed for Progress functions
- `vtkImageDilateErode3D = obj.GetFilter1 ()` - Needed for Progress functions

35.75 vtkImagePadFilter

35.75.1 Usage

vtkImagePadFilter Changes the image extent of an image. If the image extent is larger than the input image extent, the extra pixels are filled by an algorithm determined by the subclass. The image extent of the output has to be specified.

To create an instance of class vtkImagePadFilter, simply invoke its constructor as follows

```
obj = vtkImagePadFilter
```

35.75.2 Methods

The class vtkImagePadFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImagePadFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImagePadFilter = obj.NewInstance ()`
- `vtkImagePadFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutputWholeExtent (int extent[6])` - The image extent of the output has to be set explicitly.
- `obj.SetOutputWholeExtent (int minX, int maxX, int minY, int maxY, int minZ, int maxZ)` - The image extent of the output has to be set explicitly.
- `obj.GetOutputWholeExtent (int extent[6])` - The image extent of the output has to be set explicitly.
- `int = obj.GetOutputWholeExtent ()` - Set/Get the number of output scalar components.
- `obj.SetOutputNumberOfScalarComponents (int )` - Set/Get the number of output scalar components.
- `int = obj.GetOutputNumberOfScalarComponents ()` - Set/Get the number of output scalar components.

35.76 vtkImagePermute

35.76.1 Usage

vtkImagePermute reorders the axes of the input. Filtered axes specify the input axes which become X, Y, Z. The input has to have the same scalar type of the output. The filter does copy the data when it executes. This filter is actually a very thin wrapper around vtkImageReslice.

To create an instance of class vtkImagePermute, simply invoke its constructor as follows

```
obj = vtkImagePermute
```

35.76.2 Methods

The class vtkImagePermute has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImagePermute class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImagePermute = obj.NewInstance ()`
- `vtkImagePermute = obj.SafeDownCast (vtkObject o)`
- `obj.SetFilteredAxes (int x, int y, int z)` - The filtered axes are the input axes that get relabeled to X,Y,Z.
- `obj.SetFilteredAxes (int xyz[3])` - The filtered axes are the input axes that get relabeled to X,Y,Z.
- `int = obj. GetFilteredAxes ()` - The filtered axes are the input axes that get relabeled to X,Y,Z.

35.77 vtkImageQuantizeRGBToIndex

35.77.1 Usage

vtkImageQuantizeRGBToIndex takes a 3 component RGB image as input and produces a one component index image as output, along with a lookup table that contains the color definitions for the index values. This filter works on the entire input extent - it does not perform streaming, and it does not supported threaded execution (because it has to process the entire image).

To use this filter, you typically set the number of colors (between 2 and 65536), execute it, and then retrieve the lookup table. The colors can then be using the lookup table and the image index.

To create an instance of class vtkImageQuantizeRGBToIndex, simply invoke its constructor as follows

```
obj = vtkImageQuantizeRGBToIndex
```

35.77.2 Methods

The class vtkImageQuantizeRGBToIndex has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageQuantizeRGBToIndex class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkImageQuantizeRGBToIndex = obj.NewInstance ()`
- `vtkImageQuantizeRGBToIndex = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfColors (int )` - Set / Get the number of color index values to produce - must be a number between 2 and 65536.
- `int = obj.GetNumberOfColorsMinValue ()` - Set / Get the number of color index values to produce - must be a number between 2 and 65536.
- `int = obj.GetNumberOfColorsMaxValue ()` - Set / Get the number of color index values to produce - must be a number between 2 and 65536.
- `int = obj.GetNumberOfColors ()` - Set / Get the number of color index values to produce - must be a number between 2 and 65536.
- `vtkLookupTable = obj.GetLookupTable ()` - Get the resulting lookup table that contains the color definitions corresponding to the index values in the output image.
- `double = obj.GetInitializeExecuteTime ()`
- `double = obj.GetBuildTreeExecuteTime ()`
- `double = obj.GetLookupIndexExecuteTime ()`

35.78 vtkImageRange3D

35.78.1 Usage

vtkImageRange3D replaces a pixel with the maximum minus minimum over an ellipsoidal neighborhood. If KernelSize of an axis is 1, no processing is done on that axis.

To create an instance of class vtkImageRange3D, simply invoke its constructor as follows

```
obj = vtkImageRange3D
```

35.78.2 Methods

The class vtkImageRange3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageRange3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageRange3D = obj.NewInstance ()`
- `vtkImageRange3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetKernelSize (int size0, int size1, int size2)` - This method sets the size of the neighborhood. It also sets the default middle of the neighborhood and computes the elliptical foot print.

35.79 vtkImageRectilinearWipe

35.79.1 Usage

vtkImageRectilinearWipe makes a rectilinear combination of two images. The two input images must correspond in size, scalar type and number of components. The resulting image has four possible configurations called: Quad - alternate input 0 and input 1 horizontally and vertically. Select this with SetWipeModeToQuad. The Position specifies the location of the quad intersection. Corner - 3 of one input and 1 of the other. Select the location of input 0 with with SetWipeModeToLowerLeft, SetWipeModeToLowerRight, SetWipeModeToUpperLeft and SetWipeModeToUpperRight. The Position selects the location of the corner. Horizontal - alternate input 0 and input 1 with a vertical split. Select this with SetWipeModeToHorizontal. Position[0] specifies the location of the vertical transition between input 0 and input 1. Vertical - alternate input 0 and input 1 with a horizontal split. Only the y The intersection point of the rectilinear points is controlled with the Point ivar.

To create an instance of class vtkImageRectilinearWipe, simply invoke its constructor as follows

```
obj = vtkImageRectilinearWipe
```

35.79.2 Methods

The class vtkImageRectilinearWipe has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageRectilinearWipe class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageRectilinearWipe = obj.NewInstance ()`
- `vtkImageRectilinearWipe = obj.SafeDownCast (vtkObject o)`
- `obj.SetPosition (int , int )` - Set/Get the location of the image transition. Note that position is specified in pixels.
- `obj.SetPosition (int  a[2])` - Set/Get the location of the image transition. Note that position is specified in pixels.
- `int = obj. GetPosition ()` - Set/Get the location of the image transition. Note that position is specified in pixels.
- `obj.SetInput1 (vtkDataObject in)` - Set the two inputs to this filter.
- `obj.SetInput2 (vtkDataObject in)` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.
- `obj.SetWipe (int )` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1

of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `int = obj.GetWipeMinValue ()` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `int = obj.GetWipeMaxValue ()` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `int = obj.GetWipe ()` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `obj.SetWipeToQuad ()` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `obj.SetWipeToHorizontal ()` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `obj.SetWipeToVertical ()` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `obj.SetWipeToLowerLeft ()` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `obj.SetWipeToLowerRight ()` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `obj.SetWipeToUpperLeft ()` - Specify the wipe mode. This mode determnis how input 0 and input 1 are combined to produce the output. Each mode uses one or both of the values stored in Position. SetWipeToQuad - alternate input 0 and input 1 horizontally and vertically. The Position specifies the location of the quad intersection. SetWipeToLowerLeftLowerRight,UpperLeft.UpperRight - 3 of one input and 1 of the other. Select the location of input 0 to the LowerLeftLowerRight,UpperLeft,UpperRight. Position selects the location of the corner. SetWipeToHorizontal - alternate input 0 and input 1 with a vertical split. Position[0] specifies the location of the vertical transition between input 0 and input 1. SetWipeToVertical - alternate input 0 and input 1 with a horizontal split. Position[1] specifies the location of the horizonal transition between input 0 and input 1.

- `obj.SetWipeToUpperRight ()`

35.80 vtkImageResample

35.80.1 Usage

This filter produces an output with different spacing (and extent) than the input. Linear interpolation can be used to resample the data. The Output spacing can be set explicitly or relative to input spacing with the SetAxisMagnificationFactor method.

To create an instance of class vtkImageResample, simply invoke its constructor as follows

```
obj = vtkImageResample
```

35.80.2 Methods

The class vtkImageResample has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageResample class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageResample = obj.NewInstance ()`
- `vtkImageResample = obj.SafeDownCast (vtkObject o)`
- `obj.SetAxisOutputSpacing (int axis, double spacing)` - Set desired spacing. Zero is a reserved value indicating spacing has not been set.
- `obj.SetAxisMagnificationFactor (int axis, double factor)` - Set/Get Magnification factors. Zero is a reserved value indicating values have not been computed.
- `double = obj.GetAxisMagnificationFactor (int axis, vtkInformation inInfo)` - Set/Get Magnification factors. Zero is a reserved value indicating values have not been computed.
- `obj.SetDimensionality (int )` - Dimensionality is the number of axes which are considered during execution. To process images dimensionality would be set to 2. This has the same effect as setting the magnification of the third axis to 1.0
- `int = obj.GetDimensionality ()` - Dimensionality is the number of axes which are considered during execution. To process images dimensionality would be set to 2. This has the same effect as setting the magnification of the third axis to 1.0

35.81 vtkImageReslice

35.81.1 Usage

vtkImageReslice is the swiss-army-knife of image geometry filters: It can permute, rotate, flip, scale, resample, deform, and pad image data in any combination with reasonably high efficiency. Simple operations such as permutation, resampling and padding are done with similar efficiently to the specialized vtkImagePermute, vtkImageResample, and vtkImagePad filters. There are a number of tasks that vtkImageReslice is well suited for: ¡p¿1) Application of simple rotations, scales, and translations to an image. It is often a good idea to use vtkImageChangeInformation to center the image first, so that scales and rotations occur around the center rather than around the lower-left corner of the image. ¡p¿2) Resampling of one data set to match the voxel sampling of a second data set via the SetInformationInput() method, e.g. for the purpose of comparing two images or combining two images. A transformation, either linear or nonlinear, can be applied at the same time via the SetResliceTransform method if the two images are not in the same coordinate space. ¡p¿3) Extraction of slices from an image volume. The most convenient way to do this is to use SetResliceAxesDirectionCosines() to specify the orientation of the slice. The direction cosines give the x, y, and z axes for the output volume. The method SetOutputDimensionality(2) is used to specify that want to output a slice rather than a volume. The SetResliceAxesOrigin() command is used to provide an (x,y,z) point that the slice will pass through. You can use both the ResliceAxes and the ResliceTransform at the same time, in order to extract slices from a volume that you have applied a transformation to.

To create an instance of class vtkImageReslice, simply invoke its constructor as follows

```
obj = vtkImageReslice
```

35.81.2 Methods

The class vtkImageReslice has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageReslice class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageReslice = obj.NewInstance ()`
- `vtkImageReslice = obj.SafeDownCast (vtkObject o)`
- `obj.SetResliceAxes (vtkMatrix4x4 )` - This method is used to set up the axes for the output voxels. The output Spacing, Origin, and Extent specify the locations of the voxels within the coordinate system defined by the axes. The ResliceAxes are used most often to permute the data, e.g. to extract ZY or XZ slices of a volume as 2D XY images. ¡p¿The first column of the matrix specifies the x-axis vector (the fourth element must be set to zero), the second column specifies the y-axis, and the third column the z-axis. The fourth column is the origin of the axes (the fourth element must be set to one). ¡p¿An alternative to SetResliceAxes() is to use SetResliceAxesDirectionCosines() to set the directions of the axes and SetResliceAxesOrigin() to set the origin of the axes.
- `vtkMatrix4x4 = obj.GetResliceAxes ()` - This method is used to set up the axes for the output voxels. The output Spacing, Origin, and Extent specify the locations of the voxels within the coordinate system defined by the axes. The ResliceAxes are used most often to permute the data, e.g. to extract ZY or XZ slices of a volume as 2D XY images. ¡p¿The first column of the matrix specifies the x-axis vector (the fourth element must be set to zero), the second column specifies the y-axis, and the third column the z-axis. The fourth column is the origin of the axes (the fourth element must be set to one). ¡p¿An alternative to SetResliceAxes() is to use SetResliceAxesDirectionCosines() to set the directions of the axes and SetResliceAxesOrigin() to set the origin of the axes.
- `obj.SetResliceAxesDirectionCosines (double x0, double x1, double x2, double y0, double y1, double y` - Specify the direction cosines for the ResliceAxes (i.e. the first three elements of each of the first three columns of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create a new matrix if none exists.
- `obj.SetResliceAxesDirectionCosines (double x[3], double y[3], double z[3])` - Specify the direction cosines for the ResliceAxes (i.e. the first three elements of each of the first three columns of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create a new matrix if none exists.
- `obj.SetResliceAxesDirectionCosines (double xyz[9])` - Specify the direction cosines for the ResliceAxes (i.e. the first three elements of each of the first three columns of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create a new matrix if none exists.
- `obj.GetResliceAxesDirectionCosines (double x[3], double y[3], double z[3])` - Specify the direction cosines for the ResliceAxes (i.e. the first three elements of each of the first three columns of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create a new matrix if none exists.
- `obj.GetResliceAxesDirectionCosines (double xyz[9])` - Specify the direction cosines for the ResliceAxes (i.e. the first three elements of each of the first three columns of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create a new matrix if none exists.
- `double = obj.GetResliceAxesDirectionCosines ()` - Specify the direction cosines for the ResliceAxes (i.e. the first three elements of each of the first three columns of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create a new matrix if none exists.

- `obj.SetResliceAxesOrigin (double x, double y, double z)` - Specify the origin for the ResliceAxes (i.e. the first three elements of the final column of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create new matrix if none exists.

- `obj.SetResliceAxesOrigin (double xyz[3])` - Specify the origin for the ResliceAxes (i.e. the first three elements of the final column of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create new matrix if none exists.

- `obj.GetResliceAxesOrigin (double xyz[3])` - Specify the origin for the ResliceAxes (i.e. the first three elements of the final column of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create new matrix if none exists.

- `double = obj.GetResliceAxesOrigin ()` - Specify the origin for the ResliceAxes (i.e. the first three elements of the final column of the ResliceAxes matrix). This will modify the current ResliceAxes matrix, or create new matrix if none exists.

- `obj.SetResliceTransform (vtkAbstractTransform )` - Set a transform to be applied to the resampling grid that has been defined via the ResliceAxes and the output Origin, Spacing and Extent. Note that applying a transform to the resampling grid (which lies in the output coordinate system) is equivalent to applying the inverse of that transform to the input volume. Nonlinear transforms such as vtkGridTransform and vtkThinPlateSplineTransform can be used here.

- `vtkAbstractTransform = obj.GetResliceTransform ()` - Set a transform to be applied to the resampling grid that has been defined via the ResliceAxes and the output Origin, Spacing and Extent. Note that applying a transform to the resampling grid (which lies in the output coordinate system) is equivalent to applying the inverse of that transform to the input volume. Nonlinear transforms such as vtkGridTransform and vtkThinPlateSplineTransform can be used here.

- `obj.SetInformationInput (vtkImageData )` - Set a vtkImageData from which the default Spacing, Origin, and WholeExtent of the output will be copied. The spacing, origin, and extent will be permuted according to the ResliceAxes. Any values set via SetOutputSpacing, SetOutputOrigin, and SetOutputExtent will override these values. By default, the Spacing, Origin, and WholeExtent of the Input are used.

- `vtkImageData = obj.GetInformationInput ()` - Set a vtkImageData from which the default Spacing, Origin, and WholeExtent of the output will be copied. The spacing, origin, and extent will be permuted according to the ResliceAxes. Any values set via SetOutputSpacing, SetOutputOrigin, and SetOutputExtent will override these values. By default, the Spacing, Origin, and WholeExtent of the Input are used.

- `obj.SetTransformInputSampling (int )` - Specify whether to transform the spacing, origin and extent of the Input (or the InformationInput) according to the direction cosines and origin of the ResliceAxes before applying them as the default output spacing, origin and extent (default: On).

- `obj.TransformInputSamplingOn ()` - Specify whether to transform the spacing, origin and extent of the Input (or the InformationInput) according to the direction cosines and origin of the ResliceAxes before applying them as the default output spacing, origin and extent (default: On).

- `obj.TransformInputSamplingOff ()` - Specify whether to transform the spacing, origin and extent of the Input (or the InformationInput) according to the direction cosines and origin of the ResliceAxes before applying them as the default output spacing, origin and extent (default: On).

- `int = obj.GetTransformInputSampling ()` - Specify whether to transform the spacing, origin and extent of the Input (or the InformationInput) according to the direction cosines and origin of the ResliceAxes before applying them as the default output spacing, origin and extent (default: On).

- `obj.SetAutoCropOutput (int )` - Turn this on if you want to guarantee that the extent of the output will be large enough to ensure that none of the data will be cropped (default: Off).

- `obj.AutoCropOutputOn ()` - Turn this on if you want to guarantee that the extent of the output will be large enough to ensure that none of the data will be cropped (default: Off).
- `obj.AutoCropOutputOff ()` - Turn this on if you want to guarantee that the extent of the output will be large enough to ensure that none of the data will be cropped (default: Off).
- `int = obj.GetAutoCropOutput ()` - Turn this on if you want to guarantee that the extent of the output will be large enough to ensure that none of the data will be cropped (default: Off).
- `obj.SetWrap (int )` - Turn on wrap-pad feature (default: Off).
- `int = obj.GetWrap ()` - Turn on wrap-pad feature (default: Off).
- `obj.WrapOn ()` - Turn on wrap-pad feature (default: Off).
- `obj.WrapOff ()` - Turn on wrap-pad feature (default: Off).
- `obj.SetMirror (int )` - Turn on mirror-pad feature (default: Off). This will override the wrap-pad.
- `int = obj.GetMirror ()` - Turn on mirror-pad feature (default: Off). This will override the wrap-pad.
- `obj.MirrorOn ()` - Turn on mirror-pad feature (default: Off). This will override the wrap-pad.
- `obj.MirrorOff ()` - Turn on mirror-pad feature (default: Off). This will override the wrap-pad.
- `obj.SetBorder (int )` - Extend the apparent input border by a half voxel (default: On). This changes how interpolation is handled at the borders of the input image: if the center of an output voxel is beyond the edge of the input image, but is within a half voxel width of the edge (using the input voxel width), then the value of the output voxel is calculated as if the input's edge voxels were duplicated past the edges of the input. This has no effect if Mirror or Wrap are on.
- `int = obj.GetBorder ()` - Extend the apparent input border by a half voxel (default: On). This changes how interpolation is handled at the borders of the input image: if the center of an output voxel is beyond the edge of the input image, but is within a half voxel width of the edge (using the input voxel width), then the value of the output voxel is calculated as if the input's edge voxels were duplicated past the edges of the input. This has no effect if Mirror or Wrap are on.
- `obj.BorderOn ()` - Extend the apparent input border by a half voxel (default: On). This changes how interpolation is handled at the borders of the input image: if the center of an output voxel is beyond the edge of the input image, but is within a half voxel width of the edge (using the input voxel width), then the value of the output voxel is calculated as if the input's edge voxels were duplicated past the edges of the input. This has no effect if Mirror or Wrap are on.
- `obj.BorderOff ()` - Extend the apparent input border by a half voxel (default: On). This changes how interpolation is handled at the borders of the input image: if the center of an output voxel is beyond the edge of the input image, but is within a half voxel width of the edge (using the input voxel width), then the value of the output voxel is calculated as if the input's edge voxels were duplicated past the edges of the input. This has no effect if Mirror or Wrap are on.
- `obj.SetInterpolationMode (int )` - Set interpolation mode (default: nearest neighbor).
- `int = obj.GetInterpolationModeMinValue ()` - Set interpolation mode (default: nearest neighbor).
- `int = obj.GetInterpolationModeMaxValue ()` - Set interpolation mode (default: nearest neighbor).
- `int = obj.GetInterpolationMode ()` - Set interpolation mode (default: nearest neighbor).
- `obj.SetInterpolationModeToNearestNeighbor ()` - Set interpolation mode (default: nearest neighbor).

- `obj.SetInterpolationModeToLinear ()` - Set interpolation mode (default: nearest neighbor).
- `obj.SetInterpolationModeToCubic ()` - Set interpolation mode (default: nearest neighbor).
- `string = obj.GetInterpolationModeAsString ()` - Set interpolation mode (default: nearest neighbor).
- `obj.SetOptimization (int )` - Turn on and off optimizations (default on, they should only be turned off for testing purposes).
- `int = obj.GetOptimization ()` - Turn on and off optimizations (default on, they should only be turned off for testing purposes).
- `obj.OptimizationOn ()` - Turn on and off optimizations (default on, they should only be turned off for testing purposes).
- `obj.OptimizationOff ()` - Turn on and off optimizations (default on, they should only be turned off for testing purposes).
- `obj.SetBackgroundColor (double , double , double , double )` - Set the background color (for multi-component images).
- `obj.SetBackgroundColor (double a[4])` - Set the background color (for multi-component images).
- `double = obj. GetBackgroundColor ()` - Set the background color (for multi-component images).
- `obj.SetBackgroundLevel (double v)` - Set background grey level (for single-component images).
- `double = obj.GetBackgroundLevel ()` - Set background grey level (for single-component images).
- `obj.SetOutputSpacing (double , double , double )` - Set the voxel spacing for the output data. The default output spacing is the input spacing permuted through the ResliceAxes.
- `obj.SetOutputSpacing (double a[3])` - Set the voxel spacing for the output data. The default output spacing is the input spacing permuted through the ResliceAxes.
- `double = obj. GetOutputSpacing ()` - Set the voxel spacing for the output data. The default output spacing is the input spacing permuted through the ResliceAxes.
- `obj.SetOutputSpacingToDefault ()` - Set the voxel spacing for the output data. The default output spacing is the input spacing permuted through the ResliceAxes.
- `obj.SetOutputOrigin (double , double , double )` - Set the origin for the output data. The default output origin is the input origin permuted through the ResliceAxes.
- `obj.SetOutputOrigin (double a[3])` - Set the origin for the output data. The default output origin is the input origin permuted through the ResliceAxes.
- `double = obj. GetOutputOrigin ()` - Set the origin for the output data. The default output origin is the input origin permuted through the ResliceAxes.
- `obj.SetOutputOriginToDefault ()` - Set the origin for the output data. The default output origin is the input origin permuted through the ResliceAxes.
- `obj.SetOutputExtent (int , int , int , int , int , int )` - Set the extent for the output data. The default output extent is the input extent permuted through the ResliceAxes.
- `obj.SetOutputExtent (int a[6])` - Set the extent for the output data. The default output extent is the input extent permuted through the ResliceAxes.
- `int = obj. GetOutputExtent ()` - Set the extent for the output data. The default output extent is the input extent permuted through the ResliceAxes.

- `obj.SetOutputExtentToDefault ()` - Set the extent for the output data. The default output extent is the input extent permuted through the ResliceAxes.
- `obj.SetOutputDimensionality (int )` - Force the dimensionality of the output to either 1, 2, 3 or 0 (default: 3). If the dimensionality is 2D, then the Z extent of the output is forced to (0,0) and the Z origin of the output is forced to 0.0 (i.e. the output extent is confined to the xy plane). If the dimensionality is 1D, the output extent is confined to the x axis. For 0D, the output extent consists of a single voxel at (0,0,0).
- `int = obj.GetOutputDimensionality ()` - Force the dimensionality of the output to either 1, 2, 3 or 0 (default: 3). If the dimensionality is 2D, then the Z extent of the output is forced to (0,0) and the Z origin of the output is forced to 0.0 (i.e. the output extent is confined to the xy plane). If the dimensionality is 1D, the output extent is confined to the x axis. For 0D, the output extent consists of a single voxel at (0,0,0).
- `long = obj.GetMTime ()` - When determining the modified time of the filter, this check the modified time of the transform and matrix.
- `obj.ReportReferences (vtkGarbageCollector )` - Report object referenced by instances of this class.
- `obj.SetInterpolate (int t)` - Convenient methods for switching between nearest-neighbor and linear interpolation. InterpolateOn() is equivalent to SetInterpolationModeToLinear() and InterpolateOff() is equivalent to SetInterpolationModeToNearestNeighbor(). You should not use these methods if you use the SetInterpolationMode methods.
- `obj.InterpolateOn ()` - Convenient methods for switching between nearest-neighbor and linear interpolation. InterpolateOn() is equivalent to SetInterpolationModeToLinear() and InterpolateOff() is equivalent to SetInterpolationModeToNearestNeighbor(). You should not use these methods if you use the SetInterpolationMode methods.
- `obj.InterpolateOff ()` - Convenient methods for switching between nearest-neighbor and linear interpolation. InterpolateOn() is equivalent to SetInterpolationModeToLinear() and InterpolateOff() is equivalent to SetInterpolationModeToNearestNeighbor(). You should not use these methods if you use the SetInterpolationMode methods.
- `int = obj.GetInterpolate ()` - Convenient methods for switching between nearest-neighbor and linear interpolation. InterpolateOn() is equivalent to SetInterpolationModeToLinear() and InterpolateOff() is equivalent to SetInterpolationModeToNearestNeighbor(). You should not use these methods if you use the SetInterpolationMode methods.
- `obj.SetStencil (vtkImageStencilData stencil)` - Use a stencil to limit the calculations to a specific region of the output. Portions of the output that are 'outside' the stencil will be cleared to the background color.
- `vtkImageStencilData = obj.GetStencil ()` - Use a stencil to limit the calculations to a specific region of the output. Portions of the output that are 'outside' the stencil will be cleared to the background color.

35.82 vtkImageRFFT

35.82.1 Usage

vtkImageRFFT implements the reverse fast Fourier transform. The input can have real or complex data in any components and data types, but the output is always complex doubles with real values in component0, and imaginary values in component1. The filter is fastest for images that have power of two sizes. The filter uses a butterfly fitlers for each prime factor of the dimension. This makes images with prime number

dimensions (i.e. 17x17) much slower to compute. Multi dimensional (i.e volumes) FFT's are decomposed so that each axis executes in series. In most cases the RFFT will produce an image whose imaginary values are all zero's. In this case vtkImageExtractComponents can be used to remove this imaginary components leaving only the real image.

To create an instance of class vtkImageRFFT, simply invoke its constructor as follows

```
obj = vtkImageRFFT
```

35.82.2 Methods

The class vtkImageRFFT has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageRFFT class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageRFFT = obj.NewInstance ()`
- `vtkImageRFFT = obj.SafeDownCast (vtkObject o)`
- `int = obj.SplitExtent (int splitExt[6], int startExt[6], int num, int total)` - For streaming and threads. Splits output update extent into num pieces. This method needs to be called num times. Results must not overlap for consistent starting extent. Subclass can override this method. This method returns the number of pieces resulting from a successful split. This can be from 1 to "total". If 1 is returned, the extent cannot be split.

35.83 vtkImageRGBToHSI

35.83.1 Usage

For each pixel with red, blue, and green components this filter output the color coded as hue, saturation and intensity. Output type must be the same as input type.

To create an instance of class vtkImageRGBToHSI, simply invoke its constructor as follows

```
obj = vtkImageRGBToHSI
```

35.83.2 Methods

The class vtkImageRGBToHSI has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageRGBToHSI class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageRGBToHSI = obj.NewInstance ()`
- `vtkImageRGBToHSI = obj.SafeDownCast (vtkObject o)`
- `obj.SetMaximum (double )` - Hue is an angle. Maximum specifies when it maps back to 0. Hue-Maximum defaults to 255 instead of 2PI, because unsigned char is expected as input. Maximum also specifies the maximum of the Saturation.
- `double = obj.GetMaximum ()` - Hue is an angle. Maximum specifies when it maps back to 0. Hue-Maximum defaults to 255 instead of 2PI, because unsigned char is expected as input. Maximum also specifies the maximum of the Saturation.

35.84 vtkImageRGBToHSV

35.84.1 Usage

For each pixel with red, blue, and green components this filter output the color coded as hue, saturation and value. Output type must be the same as input type.

To create an instance of class vtkImageRGBToHSV, simply invoke its constructor as follows

```
obj = vtkImageRGBToHSV
```

35.84.2 Methods

The class vtkImageRGBToHSV has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageRGBToHSV class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageRGBToHSV = obj.NewInstance ()`
- `vtkImageRGBToHSV = obj.SafeDownCast (vtkObject o)`
- `obj.SetMaximum (double )`
- `double = obj.GetMaximum ()`

35.85 vtkImageSeedConnectivity

35.85.1 Usage

vtkImageSeedConnectivity marks pixels connected to user supplied seeds. The input must be unsigned char, and the output is also unsigned char. If a seed supplied by the user does not have pixel value "InputTrueValue", then the image is scanned +x, +y, +z until a pixel is encountered with value "InputTrueValue". This new pixel is used as the seed . Any pixel with out value "InputTrueValue" is consider off. The output pixels values are 0 for any off pixel in input, "OutputTrueValue" for any pixels connected to seeds, and "OutputUnconnectedValue" for any on pixels not connected to seeds. The same seeds are used for all images in the image set.

To create an instance of class vtkImageSeedConnectivity, simply invoke its constructor as follows

```
obj = vtkImageSeedConnectivity
```

35.85.2 Methods

The class vtkImageSeedConnectivity has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageSeedConnectivity class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageSeedConnectivity = obj.NewInstance ()`
- `vtkImageSeedConnectivity = obj.SafeDownCast (vtkObject o)`

- `obj.RemoveAllSeeds ()` - Methods for manipulating the seed pixels.
- `obj.AddSeed (int num, int index)` - Methods for manipulating the seed pixels.
- `obj.AddSeed (int i0, int i1, int i2)` - Methods for manipulating the seed pixels.
- `obj.AddSeed (int i0, int i1)` - Methods for manipulating the seed pixels.
- `obj.SetInputConnectValue (int )` - Set/Get what value is considered as connecting pixels.
- `int = obj.GetInputConnectValue ()` - Set/Get what value is considered as connecting pixels.
- `obj.SetOutputConnectedValue (int )` - Set/Get the value to set connected pixels to.
- `int = obj.GetOutputConnectedValue ()` - Set/Get the value to set connected pixels to.
- `obj.SetOutputUnconnectedValue (int )` - Set/Get the value to set unconnected pixels to.
- `int = obj.GetOutputUnconnectedValue ()` - Set/Get the value to set unconnected pixels to.
- `vtkImageConnector = obj.GetConnector ()` - Get the vtkImageCOnnector used by this filter.
- `obj.SetDimensionality (int )` - Set the number of axes to use in connectivity.
- `int = obj.GetDimensionality ()` - Set the number of axes to use in connectivity.

35.86 vtkImageSeparableConvolution

35.86.1 Usage

vtkImageSeparableConvolution performs a convolution along the X, Y, and Z axes of an image, based on the three different 1D convolution kernels. The kernels must be of odd size, and are considered to be centered at (int)((kernelsize - 1) / 2.0). If a kernel is NULL, that dimension is skipped. This filter is designed to efficiently convolve separable filters that can be decomposed into 1 or more 1D convolutions. It also handles arbitrarly large kernel sizes, and uses edge replication to handle boundaries.

To create an instance of class vtkImageSeparableConvolution, simply invoke its constructor as follows

```
obj = vtkImageSeparableConvolution
```

35.86.2 Methods

The class vtkImageSeparableConvolution has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageSeparableConvolution class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageSeparableConvolution = obj.NewInstance ()`
- `vtkImageSeparableConvolution = obj.SafeDownCast (vtkObject o)`
- `obj.SetXKernel (vtkFloatArray )`
- `vtkFloatArray = obj.GetXKernel ()`
- `obj.SetYKernel (vtkFloatArray )`
- `vtkFloatArray = obj.GetYKernel ()`

- `obj.SetZKernel (vtkFloatArray )`
- `vtkFloatArray = obj.GetZKernel ()`
- `long = obj.GetMTime ()` - Overload standard modified time function. If kernel arrays are modified, then this object is modified as well.

35.87 vtkImageShiftScale

35.87.1 Usage

With vtkImageShiftScale Pixels are shifted (a constant value added) and then scaled (multiplied by a scalar. As a convenience, this class allows you to set the output scalar type similar to vtkImageCast. This is because shift scale operations frequently convert data types.

To create an instance of class vtkImageShiftScale, simply invoke its constructor as follows

```
obj = vtkImageShiftScale
```

35.87.2 Methods

The class vtkImageShiftScale has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageShiftScale class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageShiftScale = obj.NewInstance ()`
- `vtkImageShiftScale = obj.SafeDownCast (vtkObject o)`
- `obj.SetShift (double )` - Set/Get the shift value. This value is added to each pixel
- `double = obj.GetShift ()` - Set/Get the shift value. This value is added to each pixel
- `obj.SetScale (double )` - Set/Get the scale value. Each pixel is multiplied by this value.
- `double = obj.GetScale ()` - Set/Get the scale value. Each pixel is multiplied by this value.
- `obj.SetOutputScalarType (int )` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `int = obj.GetOutputScalarType ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `obj.SetOutputScalarTypeToDouble ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `obj.SetOutputScalarTypeToFloat ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `obj.SetOutputScalarTypeToLong ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `obj.SetOutputScalarTypeToUnsignedLong ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `obj.SetOutputScalarTypeToInt ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.

- `obj.SetOutputScalarTypeToUnsignedInt ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `obj.SetOutputScalarTypeToShort ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `obj.SetOutputScalarTypeToUnsignedShort ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `obj.SetOutputScalarTypeToChar ()` - Set the desired output scalar type. The result of the shift and scale operations is cast to the type specified.
- `obj.SetOutputScalarTypeToUnsignedChar ()` - When the ClampOverflow flag is on, the data is thresholded so that the output value does not exceed the max or min of the data type. By default, ClampOverflow is off.
- `obj.SetClampOverflow (int )` - When the ClampOverflow flag is on, the data is thresholded so that the output value does not exceed the max or min of the data type. By default, ClampOverflow is off.
- `int = obj.GetClampOverflow ()` - When the ClampOverflow flag is on, the data is thresholded so that the output value does not exceed the max or min of the data type. By default, ClampOverflow is off.
- `obj.ClampOverflowOn ()` - When the ClampOverflow flag is on, the data is thresholded so that the output value does not exceed the max or min of the data type. By default, ClampOverflow is off.
- `obj.ClampOverflowOff ()` - When the ClampOverflow flag is on, the data is thresholded so that the output value does not exceed the max or min of the data type. By default, ClampOverflow is off.

35.88 vtkImageShrink3D

35.88.1 Usage

vtkImageShrink3D shrinks an image by sub sampling on a uniform grid (integer multiples).

To create an instance of class vtkImageShrink3D, simply invoke its constructor as follows

```
obj = vtkImageShrink3D
```

35.88.2 Methods

The class vtkImageShrink3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageShrink3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageShrink3D = obj.NewInstance ()`
- `vtkImageShrink3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetShrinkFactors (int , int , int )` - Set/Get the shrink factors
- `obj.SetShrinkFactors (int  a[3])` - Set/Get the shrink factors
- `int = obj. GetShrinkFactors ()` - Set/Get the shrink factors
- `obj.SetShift (int , int , int )` - Set/Get the pixel to use as origin.

- `obj.SetShift (int a[3])` - Set/Get the pixel to use as origin.
- `int = obj. GetShift ()` - Set/Get the pixel to use as origin.
- `obj.SetAveraging (int )` - Choose Mean, Minimum, Maximum, Median or sub sampling. The neighborhood operations are not centered on the sampled pixel. This may cause a half pixel shift in your output image. You can changed "Shift" to get around this. vtkImageGaussianSmooth or vtkImageMean with strides.
- `int = obj.GetAveraging ()` - Choose Mean, Minimum, Maximum, Median or sub sampling. The neighborhood operations are not centered on the sampled pixel. This may cause a half pixel shift in your output image. You can changed "Shift" to get around this. vtkImageGaussianSmooth or vtkImageMean with strides.
- `obj.AveragingOn ()` - Choose Mean, Minimum, Maximum, Median or sub sampling. The neighborhood operations are not centered on the sampled pixel. This may cause a half pixel shift in your output image. You can changed "Shift" to get around this. vtkImageGaussianSmooth or vtkImageMean with strides.
- `obj.AveragingOff ()` - Choose Mean, Minimum, Maximum, Median or sub sampling. The neighborhood operations are not centered on the sampled pixel. This may cause a half pixel shift in your output image. You can changed "Shift" to get around this. vtkImageGaussianSmooth or vtkImageMean with strides.
- `obj.SetMean (int )`
- `int = obj.GetMean ()`
- `obj.MeanOn ()`
- `obj.MeanOff ()`
- `obj.SetMinimum (int )`
- `int = obj.GetMinimum ()`
- `obj.MinimumOn ()`
- `obj.MinimumOff ()`
- `obj.SetMaximum (int )`
- `int = obj.GetMaximum ()`
- `obj.MaximumOn ()`
- `obj.MaximumOff ()`
- `obj.SetMedian (int )`
- `int = obj.GetMedian ()`
- `obj.MedianOn ()`
- `obj.MedianOff ()`

35.89 vtkImageSinusoidSource

35.89.1 Usage

vtkImageSinusoidSource just produces images with pixel values determined by a sinusoid.

To create an instance of class vtkImageSinusoidSource, simply invoke its constructor as follows

```
obj = vtkImageSinusoidSource
```

35.89.2 Methods

The class vtkImageSinusoidSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageSinusoidSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageSinusoidSource = obj.NewInstance ()`
- `vtkImageSinusoidSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetWholeExtent (int xMinx, int xMax, int yMin, int yMax, int zMin, int zMax)` - Set/Get the extent of the whole output image.
- `obj.SetDirection (double , double , double )` - Set/Get the direction vector which determines the sinusoidal orientation. The magnitude is ignored.
- `obj.SetDirection (double dir[3])` - Set/Get the direction vector which determines the sinusoidal orientation. The magnitude is ignored.
- `double = obj. GetDirection ()` - Set/Get the direction vector which determines the sinusoidal orientation. The magnitude is ignored.
- `obj.SetPeriod (double )` - Set/Get the period of the sinusoid in pixels.
- `double = obj.GetPeriod ()` - Set/Get the period of the sinusoid in pixels.
- `obj.SetPhase (double )` - Set/Get the phase: 0-¿2Pi. 0 =¿ Cosine, pi/2 =¿ Sine.
- `double = obj.GetPhase ()` - Set/Get the phase: 0-¿2Pi. 0 =¿ Cosine, pi/2 =¿ Sine.
- `obj.SetAmplitude (double )` - Set/Get the magnitude of the sinusoid.
- `double = obj.GetAmplitude ()` - Set/Get the magnitude of the sinusoid.

35.90 vtkImageSkeleton2D

35.90.1 Usage

vtkImageSkeleton2D should leave only single pixel width lines of non-zero-valued pixels (values of 1 are not allowed). It works by erosion on a 3x3 neighborhood with special rules. The number of iterations determines how far the filter can erode. There are three pruning levels: prune == 0 will leave traces on all angles... prune == 1 will not leave traces on 135 degree angles, but will on 90. prune == 2 does not leave traces on any angles leaving only closed loops. Prune defaults to zero. The output scalar type is the same as the input.

To create an instance of class vtkImageSkeleton2D, simply invoke its constructor as follows

```
obj = vtkImageSkeleton2D
```

35.90.2 Methods

The class vtkImageSkeleton2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageSkeleton2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageSkeleton2D = obj.NewInstance ()`
- `vtkImageSkeleton2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetPrune (int )` - When prune is on, only closed loops are left unchanged.
- `int = obj.GetPrune ()` - When prune is on, only closed loops are left unchanged.
- `obj.PruneOn ()` - When prune is on, only closed loops are left unchanged.
- `obj.PruneOff ()` - When prune is on, only closed loops are left unchanged.
- `obj.SetNumberOfIterations (int num)` - Sets the number of cycles in the erosion.

35.91 vtkImageSobel2D

35.91.1 Usage

vtkImageSobel2D computes a vector field from a scalar field by using Sobel functions. The number of vector components is 2 because the input is an image. Output is always doubles.

To create an instance of class vtkImageSobel2D, simply invoke its constructor as follows

```
obj = vtkImageSobel2D
```

35.91.2 Methods

The class vtkImageSobel2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageSobel2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageSobel2D = obj.NewInstance ()`
- `vtkImageSobel2D = obj.SafeDownCast (vtkObject o)`

35.92 vtkImageSobel3D

35.92.1 Usage

vtkImageSobel3D computes a vector field from a scalar field by using Sobel functions. The number of vector components is 3 because the input is a volume. Output is always doubles. A little creative liberty was used to extend the 2D sobel kernels into 3D.

To create an instance of class vtkImageSobel3D, simply invoke its constructor as follows

```
obj = vtkImageSobel3D
```

35.92.2 Methods

The class vtkImageSobel3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageSobel3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageSobel3D = obj.NewInstance ()`
- `vtkImageSobel3D = obj.SafeDownCast (vtkObject o)`

35.93 vtkImageSpatialAlgorithm

35.93.1 Usage

vtkImageSpatialAlgorithm is a super class for filters that operate on an input neighborhood for each output pixel. It handles even sized neighborhoods, but their can be a half pixel shift associated with processing. This superclass has some logic for handling boundaries. It can split regions into boundary and non-boundary pieces and call different execute methods.

To create an instance of class vtkImageSpatialAlgorithm, simply invoke its constructor as follows

```
obj = vtkImageSpatialAlgorithm
```

35.93.2 Methods

The class vtkImageSpatialAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageSpatialAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageSpatialAlgorithm = obj.NewInstance ()`
- `vtkImageSpatialAlgorithm = obj.SafeDownCast (vtkObject o)`
- `int = obj. GetKernelSize ()` - Get the Kernel size.
- `int = obj. GetKernelMiddle ()` - Get the Kernel middle.

35.94 vtkImageSpatialFilter

35.94.1 Usage

vtkImageSpatialFilter is a super class for filters that operate on an input neighborhood for each output pixel. It handles even sized neighborhoods, but their can be a half pixel shift associated with processing. This superclass has some logic for handling boundaries. It can split regions into boundary and non-boundary pieces and call different execute methods. .SECTION Warning This used to be the parent class for most imaging filter in VTK4.x, now this role has been replaced by vtkImageSpatialAlgorithm. You should consider using vtkImageSpatialAlgorithm instead, when writing filter for VTK5 and above. This class was kept to ensure full backward compatibility. .SECTION See also vtkSimpleImageToImageFilter vtkImageToImageFilter vtkImageSpatialAlgorithm

To create an instance of class vtkImageSpatialFilter, simply invoke its constructor as follows

```
obj = vtkImageSpatialFilter
```

35.94.2 Methods

The class vtkImageSpatialFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageSpatialFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageSpatialFilter = obj.NewInstance ()`
- `vtkImageSpatialFilter = obj.SafeDownCast (vtkObject o)`
- `int = obj. GetKernelSize ()` - Get the Kernel size.
- `int = obj. GetKernelMiddle ()` - Get the Kernel middle.

35.95 vtkImageStencil

35.95.1 Usage

vtkImageStencil will combine two images together using a stencil. The stencil should be provided in the form of a vtkImageStencilData,

To create an instance of class vtkImageStencil, simply invoke its constructor as follows

```
obj = vtkImageStencil
```

35.95.2 Methods

The class vtkImageStencil has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageStencil class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageStencil = obj.NewInstance ()`
- `vtkImageStencil = obj.SafeDownCast (vtkObject o)`
- `obj.SetStencil (vtkImageStencilData stencil)` - Specify the stencil to use. The stencil can be created from a vtkImplicitFunction or a vtkPolyData.
- `vtkImageStencilData = obj.GetStencil ()` - Specify the stencil to use. The stencil can be created from a vtkImplicitFunction or a vtkPolyData.
- `obj.SetReverseStencil (int )` - Reverse the stencil.
- `obj.ReverseStencilOn ()` - Reverse the stencil.
- `obj.ReverseStencilOff ()` - Reverse the stencil.
- `int = obj.GetReverseStencil ()` - Reverse the stencil.
- `obj.SetBackgroundInput (vtkImageData input)` - NOTE: Not yet implemented, use SetBackgroundValue instead. Set the second input. This image will be used for the 'outside' of the stencil. If not set, the output voxels will be filled with BackgroundValue instead.

- `vtkImageData = obj.GetBackgroundInput ()` - NOTE: Not yet implemented, use SetBackgroundValue instead. Set the second input. This image will be used for the 'outside' of the stencil. If not set, the output voxels will be filled with BackgroundValue instead.
- `obj.SetBackgroundValue (double val)` - Set the default output value to use when the second input is not set.
- `double = obj.GetBackgroundValue ()` - Set the default output value to use when the second input is not set.
- `obj.SetBackgroundColor (double , double , double , double )` - Set the default color to use when the second input is not set. This is like SetBackgroundValue, but for multi-component images.
- `obj.SetBackgroundColor (double a[4])` - Set the default color to use when the second input is not set. This is like SetBackgroundValue, but for multi-component images.
- `double = obj. GetBackgroundColor ()` - Set the default color to use when the second input is not set. This is like SetBackgroundValue, but for multi-component images.

35.96 vtkImageStencilData

35.96.1 Usage

vtkImageStencilData describes an image stencil in a manner which is efficient both in terms of speed and storage space. The stencil extents are stored for each x-row across the image (multiple extents per row if necessary) and can be retrieved via the GetNextExtent() method.

To create an instance of class vtkImageStencilData, simply invoke its constructor as follows

```
obj = vtkImageStencilData
```

35.96.2 Methods

The class vtkImageStencilData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageStencilData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageStencilData = obj.NewInstance ()`
- `vtkImageStencilData = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()`
- `obj.DeepCopy (vtkDataObject o)`
- `obj.ShallowCopy (vtkDataObject f)`
- `obj.InternalImageStencilDataCopy (vtkImageStencilData s)`
- `int = obj.GetDataObjectType ()` - The extent type is 3D, just like vtkImageData.
- `int = obj.GetExtentType ()` - The extent type is 3D, just like vtkImageData.
- `obj.InsertNextExtent (int r1, int r2, int yIdx, int zIdx)` - This method is used by vtkImageStencilDataSource to add an x sub extent [r1,r2] for the x row (yIdx,zIdx). The specified sub extent must not intersect any other sub extents along the same x row. As well, r1 and r2 must both be within the total x extent [Extent[0],Extent[1]].

- `obj.InsertAndMergeExtent (int r1, int r2, int yIdx, int zIdx)` - Similar to InsertNextExtent, except that the extent (r1,r2) at yIdx, zIdx is merged with other extents, (if any) on that row. So a unique extent may not necessarily be added. For instance, if an extent [5,11] already exists adding an extent, [7,9] will not affect the stencil. Likewise adding [10, 13] will replace the existing extent with [5,13].

- `obj.RemoveExtent (int r1, int r2, int yIdx, int zIdx)` - Remove the extent from (r1,r2) at yIdx, zIdx

- `obj.SetSpacing (double , double , double )` - Set the desired spacing for the stencil. This must be called before the stencil is Updated, ideally in the ExecuteInformation method of the imaging filter that is using the stencil.

- `obj.SetSpacing (double a[3])` - Set the desired spacing for the stencil. This must be called before the stencil is Updated, ideally in the ExecuteInformation method of the imaging filter that is using the stencil.

- `double = obj. GetSpacing ()` - Set the desired spacing for the stencil. This must be called before the stencil is Updated, ideally in the ExecuteInformation method of the imaging filter that is using the stencil.

- `obj.SetOrigin (double , double , double )` - Set the desired origin for the stencil. This must be called before the stencil is Updated, ideally in the ExecuteInformation method of the imaging filter that is using the stencil.

- `obj.SetOrigin (double a[3])` - Set the desired origin for the stencil. This must be called before the stencil is Updated, ideally in the ExecuteInformation method of the imaging filter that is using the stencil.

- `double = obj. GetOrigin ()` - Set the desired origin for the stencil. This must be called before the stencil is Updated, ideally in the ExecuteInformation method of the imaging filter that is using the stencil.

- `obj.SetExtent (int extent[6])` - Set the extent of the data. This is should be called only by vtkImageStencilSource, as it is part of the basic pipeline functionality.

- `obj.SetExtent (int x1, int x2, int y1, int y2, int z1, int z2)` - Set the extent of the data. This is should be called only by vtkImageStencilSource, as it is part of the basic pipeline functionality.

- `int = obj. GetExtent ()` - Set the extent of the data. This is should be called only by vtkImageStencilSource, as it is part of the basic pipeline functionality.

- `obj.AllocateExtents ()` - Allocate space for the sub-extents. This is called by vtkImageStencilSource.

- `obj.Fill ()` - Fill the sub-extents.

- `obj.CopyInformationToPipeline (vtkInformation request, vtkInformation input, vtkInformation output,` - Override these to handle origin, spacing, scalar type, and scalar number of components. See vtkDataObject for details.

- `obj.CopyInformationFromPipeline (vtkInformation request)` - Override these to handle origin, spacing, scalar type, and scalar number of components. See vtkDataObject for details.

- `obj.Add (vtkImageStencilData )` - Add merges the stencil supplied as argument into Self.

- `obj.Subtract (vtkImageStencilData )` - Subtract removes the portion of the stencil, supplied as argument, that lies within Self from Self.

- `obj.Replace (vtkImageStencilData )` - Replaces the portion of the stencil, supplied as argument, that lies within Self from Self.
- `int = obj.Clip (int extent[6])` - Clip the stencil with the supplied extents. In other words, discard data outside the specified extents. Return 1 if something changed.

35.97 vtkImageStencilSource

35.97.1 Usage

vtkImageStencilSource is a superclass for filters that generate image stencils. Given a clipping object such as a vtkImplicitFunction, it will set up a list of clipping extents for each x-row through the image data. The extents for each x-row can be retrieved via the GetNextExtent() method after the extent lists have been built with the BuildExtents() method. For large images, using clipping extents is much more memory efficient (and slightly more time-efficient) than building a mask. This class can be subclassed to allow clipping with objects other than vtkImplicitFunction.

To create an instance of class vtkImageStencilSource, simply invoke its constructor as follows

```
obj = vtkImageStencilSource
```

35.97.2 Methods

The class vtkImageStencilSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageStencilSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageStencilSource = obj.NewInstance ()`
- `vtkImageStencilSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutput (vtkImageStencilData output)` - Get or set the output for this source.
- `vtkImageStencilData = obj.GetOutput ()` - Get or set the output for this source.

35.98 vtkImageThreshold

35.98.1 Usage

vtkImageThreshold can do binary or continuous thresholding for lower, upper or a range of data. The output data type may be different than the output, but defaults to the same type.

To create an instance of class vtkImageThreshold, simply invoke its constructor as follows

```
obj = vtkImageThreshold
```

35.98.2 Methods

The class vtkImageThreshold has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageThreshold class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkImageThreshold = obj.NewInstance ()`
- `vtkImageThreshold = obj.SafeDownCast (vtkObject o)`
- `obj.ThresholdByUpper (double thresh)` - The values greater than or equal to the value match.
- `obj.ThresholdByLower (double thresh)` - The values less than or equal to the value match.
- `obj.ThresholdBetween (double lower, double upper)` - The values in a range (inclusive) match
- `obj.SetReplaceIn (int )` - Determines whether to replace the pixel in range with InValue
- `int = obj.GetReplaceIn ()` - Determines whether to replace the pixel in range with InValue
- `obj.ReplaceInOn ()` - Determines whether to replace the pixel in range with InValue
- `obj.ReplaceInOff ()` - Determines whether to replace the pixel in range with InValue
- `obj.SetInValue (double val)` - Replace the in range pixels with this value.
- `double = obj.GetInValue ()` - Replace the in range pixels with this value.
- `obj.SetReplaceOut (int )` - Determines whether to replace the pixel out of range with OutValue
- `int = obj.GetReplaceOut ()` - Determines whether to replace the pixel out of range with OutValue
- `obj.ReplaceOutOn ()` - Determines whether to replace the pixel out of range with OutValue
- `obj.ReplaceOutOff ()` - Determines whether to replace the pixel out of range with OutValue
- `obj.SetOutValue (double val)` - Replace the in range pixels with this value.
- `double = obj.GetOutValue ()` - Replace the in range pixels with this value.
- `double = obj.GetUpperThreshold ()` - Get the Upper and Lower thresholds.
- `double = obj.GetLowerThreshold ()` - Get the Upper and Lower thresholds.
- `obj.SetOutputScalarType (int )` - Set the desired output scalar type to cast to
- `int = obj.GetOutputScalarType ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToDouble ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToFloat ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToLong ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToUnsignedLong ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToInt ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToUnsignedInt ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToShort ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToUnsignedShort ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToChar ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToSignedChar ()` - Set the desired output scalar type to cast to
- `obj.SetOutputScalarTypeToUnsignedChar ()`

35.99 vtkImageToImageStencil

35.99.1 Usage

vtkImageToImageStencil will convert a vtkImageData into an stencil that can be used with vtkImageStecil or other vtk classes that apply a stencil to an image.

To create an instance of class vtkImageToImageStencil, simply invoke its constructor as follows

```
obj = vtkImageToImageStencil
```

35.99.2 Methods

The class vtkImageToImageStencil has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageToImageStencil class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageToImageStencil = obj.NewInstance ()`
- `vtkImageToImageStencil = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImageData input)` - Specify the image data to convert into a stencil.
- `vtkImageData = obj.GetInput ()` - Specify the image data to convert into a stencil.
- `obj.ThresholdByUpper (double thresh)` - The values greater than or equal to the value match.
- `obj.ThresholdByLower (double thresh)` - The values less than or equal to the value match.
- `obj.ThresholdBetween (double lower, double upper)` - The values in a range (inclusive) match
- `obj.SetUpperThreshold (double )` - Get the Upper and Lower thresholds.
- `double = obj.GetUpperThreshold ()` - Get the Upper and Lower thresholds.
- `obj.SetLowerThreshold (double )` - Get the Upper and Lower thresholds.
- `double = obj.GetLowerThreshold ()` - Get the Upper and Lower thresholds.

35.100 vtkImageTranslateExtent

35.100.1 Usage

vtkImageTranslateExtent shift the whole extent, but does not change the data.

To create an instance of class vtkImageTranslateExtent, simply invoke its constructor as follows

```
obj = vtkImageTranslateExtent
```

35.100.2 Methods

The class vtkImageTranslateExtent has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageTranslateExtent class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageTranslateExtent = obj.NewInstance ()`
- `vtkImageTranslateExtent = obj.SafeDownCast (vtkObject o)`
- `obj.SetTranslation (int , int , int )` - Delta to change "WholeExtent". -1 changes 0-¿10 to -1-¿9.
- `obj.SetTranslation (int  a[3])` - Delta to change "WholeExtent". -1 changes 0-¿10 to -1-¿9.
- `int = obj. GetTranslation ()` - Delta to change "WholeExtent". -1 changes 0-¿10 to -1-¿9.

35.101 vtkImageVariance3D

35.101.1 Usage

vtkImageVariance3D replaces each pixel with a measurement of pixel variance in a elliptical neighborhood centered on that pixel. The value computed is not exactly the variance. The difference between the neighbor values and center value is computed and squared for each neighbor. These values are summed and divided by the total number of neighbors to produce the output value.

To create an instance of class vtkImageVariance3D, simply invoke its constructor as follows

```
obj = vtkImageVariance3D
```

35.101.2 Methods

The class vtkImageVariance3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageVariance3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageVariance3D = obj.NewInstance ()`
- `vtkImageVariance3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetKernelSize (int size0, int size1, int size2)` - This method sets the size of the neighborhood. It also sets the default middle of the neighborhood and computes the Elliptical foot print.

35.102 vtkImageWeightedSum

35.102.1 Usage

All weights are normalized so they will sum to 1. Images must have the same extents. Output is

.SECTION Thanks The original author of this class is Lauren O'Donnell (MIT) for Slicer

To create an instance of class vtkImageWeightedSum, simply invoke its constructor as follows

```
obj = vtkImageWeightedSum
```

35.102.2 Methods

The class vtkImageWeightedSum has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageWeightedSum class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageWeightedSum = obj.NewInstance ()`
- `vtkImageWeightedSum = obj.SafeDownCast (vtkObject o)`
- `obj.SetWeights (vtkDoubleArray )` - The weights control the contribution of each input to the sum. They will be normalized to sum to 1 before filter execution.
- `vtkDoubleArray = obj.GetWeights ()` - The weights control the contribution of each input to the sum. They will be normalized to sum to 1 before filter execution.
- `obj.SetWeight (vtkIdType id, double weight)` - Change a specific weight. Reallocation is done
- `int = obj.GetNormalizeByWeight ()` - Setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the weighted sum By default, NormalizeByWeight is on.
- `obj.SetNormalizeByWeight (int )` - Setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the weighted sum By default, NormalizeByWeight is on.
- `int = obj.GetNormalizeByWeightMinValue ()` - Setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the weighted sum By default, NormalizeByWeight is on.
- `int = obj.GetNormalizeByWeightMaxValue ()` - Setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the weighted sum By default, NormalizeByWeight is on.
- `obj.NormalizeByWeightOn ()` - Setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the weighted sum By default, NormalizeByWeight is on.
- `obj.NormalizeByWeightOff ()` - Setting NormalizeByWeight on will divide the final result by the total weight of the component functions. This process does not otherwise normalize the weighted sum By default, NormalizeByWeight is on.
- `double = obj.CalculateTotalWeight ()` - Compute the total value of all the weight

35.103 vtkImageWrapPad

35.103.1 Usage

vtkImageWrapPad performs a modulo operation on the output pixel index to determine the source input index. The new image extent of the output has to be specified. Input has to be the same scalar type as output.

To create an instance of class vtkImageWrapPad, simply invoke its constructor as follows

```
obj = vtkImageWrapPad
```

35.103.2 Methods

The class vtkImageWrapPad has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageWrapPad class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageWrapPad = obj.NewInstance ()`
- `vtkImageWrapPad = obj.SafeDownCast (vtkObject o)`

35.104 vtkImplicitFunctionToImageStencil

35.104.1 Usage

vtkImplicitFunctionToImageStencil will convert a vtkImplicitFunction into a stencil that can be used with vtkImageStencil or with other classes that apply a stencil to an image.

To create an instance of class vtkImplicitFunctionToImageStencil, simply invoke its constructor as follows

```
obj = vtkImplicitFunctionToImageStencil
```

35.104.2 Methods

The class vtkImplicitFunctionToImageStencil has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitFunctionToImageStencil class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitFunctionToImageStencil = obj.NewInstance ()`
- `vtkImplicitFunctionToImageStencil = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImplicitFunction )` - Specify the implicit function to convert into a stencil.
- `vtkImplicitFunction = obj.GetInput ()` - Specify the implicit function to convert into a stencil.
- `obj.SetInformationInput (vtkImageData )` - Set a vtkImageData that has the Spacing, Origin, and WholeExtent that will be used for the stencil. This input should be set to the image that you wish to apply the stencil to. If you use this method, then any values set with the SetOutputSpacing, SetOutputOrigin, and SetOutputWholeExtent methods will be ignored.
- `vtkImageData = obj.GetInformationInput ()` - Set a vtkImageData that has the Spacing, Origin, and WholeExtent that will be used for the stencil. This input should be set to the image that you wish to apply the stencil to. If you use this method, then any values set with the SetOutputSpacing, SetOutputOrigin, and SetOutputWholeExtent methods will be ignored.
- `obj.SetOutputOrigin (double , double , double )` - Set the Origin to be used for the stencil. It should be set to the Origin of the image you intend to apply the stencil to. The default value is (0,0,0).
- `obj.SetOutputOrigin (double  a[3])` - Set the Origin to be used for the stencil. It should be set to the Origin of the image you intend to apply the stencil to. The default value is (0,0,0).

- `double = obj. GetOutputOrigin ()` - Set the Origin to be used for the stencil. It should be set to the Origin of the image you intend to apply the stencil to. The default value is (0,0,0).
- `obj.SetOutputSpacing (double , double , double )` - Set the Spacing to be used for the stencil. It should be set to the Spacing of the image you intend to apply the stencil to. The default value is (1,1,1)
- `obj.SetOutputSpacing (double a[3])` - Set the Spacing to be used for the stencil. It should be set to the Spacing of the image you intend to apply the stencil to. The default value is (1,1,1)
- `double = obj. GetOutputSpacing ()` - Set the Spacing to be used for the stencil. It should be set to the Spacing of the image you intend to apply the stencil to. The default value is (1,1,1)
- `obj.SetOutputWholeExtent (int , int , int , int , int , int )` - Set the whole extent for the stencil (anything outside this extent will be considered to be "outside" the stencil). If this is not set, then the stencil will always use the requested UpdateExtent as the stencil extent.
- `obj.SetOutputWholeExtent (int a[6])` - Set the whole extent for the stencil (anything outside this extent will be considered to be "outside" the stencil). If this is not set, then the stencil will always use the requested UpdateExtent as the stencil extent.
- `int = obj. GetOutputWholeExtent ()` - Set the whole extent for the stencil (anything outside this extent will be considered to be "outside" the stencil). If this is not set, then the stencil will always use the requested UpdateExtent as the stencil extent.
- `obj.SetThreshold (double )` - Set the threshold value for the implicit function.
- `double = obj.GetThreshold ()` - Set the threshold value for the implicit function.

35.105 vtkPointLoad

35.105.1 Usage

vtkPointLoad is a source object that computes stress tensors on a volume. The tensors are computed from the application of a point load on a semi-infinite domain. (The analytical results are adapted from Saada - see text.) It also is possible to compute effective stress scalars if desired. This object serves as a specialized data generator for some of the examples in the text.

To create an instance of class vtkPointLoad, simply invoke its constructor as follows

```
obj = vtkPointLoad
```

35.105.2 Methods

The class vtkPointLoad has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointLoad class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointLoad = obj.NewInstance ()`
- `vtkPointLoad = obj.SafeDownCast (vtkObject o)`
- `obj.SetLoadValue (double )` - Set/Get value of applied load.
- `double = obj.GetLoadValue ()` - Set/Get value of applied load.

- `obj.SetSampleDimensions (int i, int j, int k)` - Specify the dimensions of the volume. A stress tensor will be computed for each point in the volume.
- `obj.SetSampleDimensions (int dim[3])` - Specify the dimensions of the volume. A stress tensor will be computed for each point in the volume.
- `int = obj. GetSampleDimensions ()` - Specify the dimensions of the volume. A stress tensor will be computed for each point in the volume.
- `obj.SetModelBounds (double , double , double , double , double , double )` - Specify the region in space over which the tensors are computed. The point load is assumed to be applied at top center of the volume.
- `obj.SetModelBounds (double a[6])` - Specify the region in space over which the tensors are computed. The point load is assumed to be applied at top center of the volume.
- `double = obj. GetModelBounds ()` - Specify the region in space over which the tensors are computed. The point load is assumed to be applied at top center of the volume.
- `obj.SetPoissonsRatio (double )` - Set/Get Poisson's ratio.
- `double = obj.GetPoissonsRatio ()` - Set/Get Poisson's ratio.
- `obj.SetComputeEffectiveStress (int )` - Turn on/off computation of effective stress scalar. These methods do nothing. The effective stress is always computed.
- `int = obj.GetComputeEffectiveStress ()` - Turn on/off computation of effective stress scalar. These methods do nothing. The effective stress is always computed.
- `obj.ComputeEffectiveStressOn ()` - Turn on/off computation of effective stress scalar. These methods do nothing. The effective stress is always computed.
- `obj.ComputeEffectiveStressOff ()` - Turn on/off computation of effective stress scalar. These methods do nothing. The effective stress is always computed.

35.106 vtkRTAnalyticSource

35.106.1 Usage

vtkRTAnalyticSource just produces images with pixel values determined by a Maximum*Gaussian*XMag*sin(XFreq*x)*sin(Y
Values are float scalars on point data with name "RTData".

To create an instance of class vtkRTAnalyticSource, simply invoke its constructor as follows

```
obj = vtkRTAnalyticSource
```

35.106.2 Methods

The class vtkRTAnalyticSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRTAnalyticSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRTAnalyticSource = obj.NewInstance ()`
- `vtkRTAnalyticSource = obj.SafeDownCast (vtkObject o)`

- `obj.SetWholeExtent (int xMinx, int xMax, int yMin, int yMax, int zMin, int zMax)` - Set/Get the extent of the whole output image. Initial value is -10,10,-10,10,-10,10
- `int = obj. GetWholeExtent ()` - Set/Get the extent of the whole output image. Initial value is -10,10,-10,10,-10,10
- `obj.SetCenter (double , double , double )` - Set/Get the center of function. Initial value is 0.0,0.0,0.0
- `obj.SetCenter (double  a[3])` - Set/Get the center of function. Initial value is 0.0,0.0,0.0
- `double = obj. GetCenter ()` - Set/Get the center of function. Initial value is 0.0,0.0,0.0
- `obj.SetMaximum (double )` - Set/Get the Maximum value of the function. Initial value is 255.0.
- `double = obj.GetMaximum ()` - Set/Get the Maximum value of the function. Initial value is 255.0.
- `obj.SetStandardDeviation (double )` - Set/Get the standard deviation of the function. Initial value is 0.5.
- `double = obj.GetStandardDeviation ()` - Set/Get the standard deviation of the function. Initial value is 0.5.
- `obj.SetXFreq (double )` - Set/Get the natural frequency in x. Initial value is 60.
- `double = obj.GetXFreq ()` - Set/Get the natural frequency in x. Initial value is 60.
- `obj.SetYFreq (double )` - Set/Get the natural frequency in y. Initial value is 30.
- `double = obj.GetYFreq ()` - Set/Get the natural frequency in y. Initial value is 30.
- `obj.SetZFreq (double )` - Set/Get the natural frequency in z. Initial value is 40.
- `double = obj.GetZFreq ()` - Set/Get the natural frequency in z. Initial value is 40.
- `obj.SetXMag (double )` - Set/Get the magnitude in x. Initial value is 10.
- `double = obj.GetXMag ()` - Set/Get the magnitude in x. Initial value is 10.
- `obj.SetYMag (double )` - Set/Get the magnitude in y. Initial value is 18.
- `double = obj.GetYMag ()` - Set/Get the magnitude in y. Initial value is 18.
- `obj.SetZMag (double )` - Set/Get the magnitude in z. Initial value is 5.
- `double = obj.GetZMag ()` - Set/Get the magnitude in z. Initial value is 5.
- `obj.SetSubsampleRate (int )` - Set/Get the sub-sample rate. Initial value is 1.
- `int = obj.GetSubsampleRate ()` - Set/Get the sub-sample rate. Initial value is 1.

35.107 vtkSampleFunction

35.107.1 Usage

vtkSampleFunction is a source object that evaluates an implicit function and normals at each point in a vtkStructuredPoints. The user can specify the sample dimensions and location in space to perform the sampling. To create closed surfaces (in conjunction with the vtkContourFilter), capping can be turned on to set a particular value on the boundaries of the sample space.

To create an instance of class vtkSampleFunction, simply invoke its constructor as follows

```
obj = vtkSampleFunction
```

35.107.2 Methods

The class vtkSampleFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSampleFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSampleFunction = obj.NewInstance ()`
- `vtkSampleFunction = obj.SafeDownCast (vtkObject o)`
- `obj.SetImplicitFunction (vtkImplicitFunction )` - Specify the implicit function to use to generate data.
- `vtkImplicitFunction = obj.GetImplicitFunction ()` - Specify the implicit function to use to generate data.
- `obj.SetOutputScalarType (int )` - Set what type of scalar data this source should generate.
- `int = obj.GetOutputScalarType ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToDouble ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToFloat ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToLong ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedLong ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToInt ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedInt ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToShort ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedShort ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToChar ()` - Set what type of scalar data this source should generate.
- `obj.SetOutputScalarTypeToUnsignedChar ()` - Control the type of the scalars object by explicitly providing a scalar object. THIS IS DEPRECATED, although it still works!!! Please use SetOutputScalarType instead.
- `obj.SetScalars (vtkDataArray da)` - Control the type of the scalars object by explicitly providing a scalar object. THIS IS DEPRECATED, although it still works!!! Please use SetOutputScalarType instead.
- `obj.SetSampleDimensions (int i, int j, int k)` - Specify the dimensions of the data on which to sample.
- `obj.SetSampleDimensions (int dim[3])` - Specify the dimensions of the data on which to sample.
- `int = obj. GetSampleDimensions ()` - Specify the dimensions of the data on which to sample.

- `obj.SetModelBounds (double , double , double , double , double , double )` - Specify the region in space over which the sampling occurs. The bounds is specified as (xMin,xMax, yMin,yMax, zMin,zMax).
- `obj.SetModelBounds (double a[6])` - Specify the region in space over which the sampling occurs. The bounds is specified as (xMin,xMax, yMin,yMax, zMin,zMax).
- `double = obj. GetModelBounds ()` - Specify the region in space over which the sampling occurs. The bounds is specified as (xMin,xMax, yMin,yMax, zMin,zMax).
- `obj.SetCapping (int )` - Turn on/off capping. If capping is on, then the outer boundaries of the structured point set are set to cap value. This can be used to insure surfaces are closed.
- `int = obj.GetCapping ()` - Turn on/off capping. If capping is on, then the outer boundaries of the structured point set are set to cap value. This can be used to insure surfaces are closed.
- `obj.CappingOn ()` - Turn on/off capping. If capping is on, then the outer boundaries of the structured point set are set to cap value. This can be used to insure surfaces are closed.
- `obj.CappingOff ()` - Turn on/off capping. If capping is on, then the outer boundaries of the structured point set are set to cap value. This can be used to insure surfaces are closed.
- `obj.SetCapValue (double )` - Set the cap value.
- `double = obj.GetCapValue ()` - Set the cap value.
- `obj.SetComputeNormals (int )` - Turn on/off the computation of normals (normals are float values).
- `int = obj.GetComputeNormals ()` - Turn on/off the computation of normals (normals are float values).
- `obj.ComputeNormalsOn ()` - Turn on/off the computation of normals (normals are float values).
- `obj.ComputeNormalsOff ()` - Turn on/off the computation of normals (normals are float values).
- `obj.SetScalarArrayName (string )` - Set/get the scalar array name for this data set. Initial value is "scalars".
- `string = obj.GetScalarArrayName ()` - Set/get the scalar array name for this data set. Initial value is "scalars".
- `obj.SetNormalArrayName (string )` - Set/get the normal array name for this data set. Initial value is "normals".
- `string = obj.GetNormalArrayName ()` - Set/get the normal array name for this data set. Initial value is "normals".
- `long = obj.GetMTime ()` - Return the MTime also considering the implicit function.

35.108 vtkShepardMethod

35.108.1 Usage

vtkShepardMethod is a filter used to visualize unstructured point data using Shepard's method. The method works by resampling the unstructured points onto a structured points set. The influence functions are described as "inverse distance weighted". Once the structured points are computed, the usual visualization techniques (e.g., iso-contouring or volume rendering) can be used visualize the structured points.

To create an instance of class vtkShepardMethod, simply invoke its constructor as follows

```
obj = vtkShepardMethod
```

35.108.2 Methods

The class vtkShepardMethod has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkShepardMethod class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkShepardMethod = obj.NewInstance ()`
- `vtkShepardMethod = obj.SafeDownCast (vtkObject o)`
- `double = obj.ComputeModelBounds (double origin[3], double ar[3])` - Compute ModelBounds from input geometry.
- `int = obj. GetSampleDimensions ()` - Specify i-j-k dimensions on which to sample input points.
- `obj.SetSampleDimensions (int i, int j, int k)` - Set the i-j-k dimensions on which to sample the distance function.
- `obj.SetSampleDimensions (int dim[3])` - Set the i-j-k dimensions on which to sample the distance function.
- `obj.SetMaximumDistance (double )` - Specify influence distance of each input point. This distance is a fraction of the length of the diagonal of the sample space. Thus, values of 1.0 will cause each input point to influence all points in the structured point dataset. Values less than 1.0 can improve performance significantly.
- `double = obj.GetMaximumDistanceMinValue ()` - Specify influence distance of each input point. This distance is a fraction of the length of the diagonal of the sample space. Thus, values of 1.0 will cause each input point to influence all points in the structured point dataset. Values less than 1.0 can improve performance significantly.
- `double = obj.GetMaximumDistanceMaxValue ()` - Specify influence distance of each input point. This distance is a fraction of the length of the diagonal of the sample space. Thus, values of 1.0 will cause each input point to influence all points in the structured point dataset. Values less than 1.0 can improve performance significantly.
- `double = obj.GetMaximumDistance ()` - Specify influence distance of each input point. This distance is a fraction of the length of the diagonal of the sample space. Thus, values of 1.0 will cause each input point to influence all points in the structured point dataset. Values less than 1.0 can improve performance significantly.
- `obj.SetModelBounds (double , double , double , double , double , double )` - Specify the position in space to perform the sampling.
- `obj.SetModelBounds (double  a[6])` - Specify the position in space to perform the sampling.
- `double = obj. GetModelBounds ()` - Specify the position in space to perform the sampling.
- `obj.SetNullValue (double )` - Set the Null value for output points not receiving a contribution from the input points.
- `double = obj.GetNullValue ()` - Set the Null value for output points not receiving a contribution from the input points.

35.109 vtkSimpleImageFilterExample

35.109.1 Usage

This is an example of a simple image-image filter. It copies it's input to it's output (point by point). It shows how templates can be used to support various data types. .SECTION See also vtkSimpleImageToImageFilter

To create an instance of class vtkSimpleImageFilterExample, simply invoke its constructor as follows

```
obj = vtkSimpleImageFilterExample
```

35.109.2 Methods

The class vtkSimpleImageFilterExample has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSimpleImageFilterExample class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSimpleImageFilterExample = obj.NewInstance ()`
- `vtkSimpleImageFilterExample = obj.SafeDownCast (vtkObject o)`

35.110 vtkSurfaceReconstructionFilter

35.110.1 Usage

vtkSurfaceReconstructionFilter takes a list of points assumed to lie on the surface of a solid 3D object. A signed measure of the distance to the surface is computed and sampled on a regular grid. The grid can then be contoured at zero to extract the surface. The default values for neighborhood size and sample spacing should give reasonable results for most uses but can be set if desired. This procedure is based on the PhD work of Hugues Hoppe: http://www.research.microsoft.com/ hoppe

To create an instance of class vtkSurfaceReconstructionFilter, simply invoke its constructor as follows

```
obj = vtkSurfaceReconstructionFilter
```

35.110.2 Methods

The class vtkSurfaceReconstructionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSurfaceReconstructionFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSurfaceReconstructionFilter = obj.NewInstance ()`
- `vtkSurfaceReconstructionFilter = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNeighborhoodSize ()` - Specify the number of neighbors each point has, used for estimating the local surface orientation. The default value of 20 should be OK for most applications, higher values can be specified if the spread of points is uneven. Values as low as 10 may yield adequate results for some surfaces. Higher values cause the algorithm to take longer. Higher values will cause errors on sharp boundaries.

- `obj.SetNeighborhoodSize (int )` - Specify the number of neighbors each point has, used for estimating the local surface orientation. The default value of 20 should be OK for most applications, higher values can be specified if the spread of points is uneven. Values as low as 10 may yield adequate results for some surfaces. Higher values cause the algorithm to take longer. Higher values will cause errors on sharp boundaries.
- `double = obj.GetSampleSpacing ()` - Specify the spacing of the 3D sampling grid. If not set, a reasonable guess will be made.
- `obj.SetSampleSpacing (double )` - Specify the spacing of the 3D sampling grid. If not set, a reasonable guess will be made.

35.111 vtkTriangularTexture

35.111.1 Usage

vtkTriangularTexture is a filter that generates a 2D texture map based on the paper "Opacity-modulating Triangular Textures for Irregular Surfaces," by Penny Rheingans, IEEE Visualization '96, pp. 219-225. The textures assume texture coordinates of (0,0), (1.0) and (.5, sqrt(3)/2). The sequence of texture values is the same along each edge of the triangular texture map. So, the assignment order of texture coordinates is arbitrary.

To create an instance of class vtkTriangularTexture, simply invoke its constructor as follows

```
obj = vtkTriangularTexture
```

35.111.2 Methods

The class vtkTriangularTexture has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTriangularTexture class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTriangularTexture = obj.NewInstance ()`
- `vtkTriangularTexture = obj.SafeDownCast (vtkObject o)`
- `obj.SetScaleFactor (double )` - Set a Scale Factor.
- `double = obj.GetScaleFactor ()` - Set a Scale Factor.
- `obj.SetXSize (int )` - Set the X texture map dimension. Default is 64.
- `int = obj.GetXSize ()` - Set the X texture map dimension. Default is 64.
- `obj.SetYSize (int )` - Set the Y texture map dimension. Default is 64.
- `int = obj.GetYSize ()` - Set the Y texture map dimension. Default is 64.
- `obj.SetTexturePattern (int )` - Set the texture pattern. 1 = opaque at centroid (default) 2 = opaque at vertices 3 = opaque in rings around vertices
- `int = obj.GetTexturePatternMinValue ()` - Set the texture pattern. 1 = opaque at centroid (default) 2 = opaque at vertices 3 = opaque in rings around vertices
- `int = obj.GetTexturePatternMaxValue ()` - Set the texture pattern. 1 = opaque at centroid (default) 2 = opaque at vertices 3 = opaque in rings around vertices
- `int = obj.GetTexturePattern ()` - Set the texture pattern. 1 = opaque at centroid (default) 2 = opaque at vertices 3 = opaque in rings around vertices

35.112 vtkVoxelModeller

35.112.1 Usage

vtkVoxelModeller is a filter that converts an arbitrary data set to a structured point (i.e., voxel) representation. It is very similar to vtkImplicitModeller, except that it doesn't record distance; instead it records occupancy. By default it supports a compact output of 0/1 VTK_BIT. Other vtk scalar types can be specified. The Foreground and Background values of the output can also be specified. NOTE: Not all vtk filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.

To create an instance of class vtkVoxelModeller, simply invoke its constructor as follows

```
obj = vtkVoxelModeller
```

35.112.2 Methods

The class vtkVoxelModeller has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVoxelModeller class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVoxelModeller = obj.NewInstance ()`
- `vtkVoxelModeller = obj.SafeDownCast (vtkObject o)`
- `double = obj.ComputeModelBounds (double origin[3], double ar[3])` - Compute the ModelBounds based on the input geometry.
- `obj.SetSampleDimensions (int i, int j, int k)` - Set the i-j-k dimensions on which to sample the distance function. Default is (50, 50, 50)
- `obj.SetSampleDimensions (int dim[3])` - Set the i-j-k dimensions on which to sample the distance function. Default is (50, 50, 50)
- `int = obj. GetSampleDimensions ()` - Set the i-j-k dimensions on which to sample the distance function. Default is (50, 50, 50)
- `obj.SetMaximumDistance (double )` - Specify distance away from surface of input geometry to sample. Smaller values make large increases in performance. Default is 1.0.
- `double = obj.GetMaximumDistanceMinValue ()` - Specify distance away from surface of input geometry to sample. Smaller values make large increases in performance. Default is 1.0.
- `double = obj.GetMaximumDistanceMaxValue ()` - Specify distance away from surface of input geometry to sample. Smaller values make large increases in performance. Default is 1.0.
- `double = obj.GetMaximumDistance ()` - Specify distance away from surface of input geometry to sample. Smaller values make large increases in performance. Default is 1.0.
- `obj.SetModelBounds (double bounds[6])` - Specify the position in space to perform the voxelization. Default is (0, 0, 0, 0, 0, 0)
- `obj.SetModelBounds (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Specify the position in space to perform the voxelization. Default is (0, 0, 0, 0, 0, 0)
- `double = obj. GetModelBounds ()` - Specify the position in space to perform the voxelization. Default is (0, 0, 0, 0, 0, 0)

- `obj.SetScalarType (int )` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToFloat ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToDouble ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToInt ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToUnsignedInt ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToLong ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToUnsignedLong ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToShort ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToUnsignedShort ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToUnsignedChar ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToChar ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetScalarTypeToBit ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `int = obj.GetScalarType ()` - Control the scalar type of the output image. The default is VTK_BIT. NOTE: Not all filters/readers/writers support the VTK_BIT scalar type. You may want to use VTK_CHAR as an alternative.
- `obj.SetForegroundValue (double )` - Set the Foreground/Background values of the output. The Foreground value is set when a voxel is occupied. The Background value is set when a voxel is not occupied. The default ForegroundValue is 1. The default BackgroundValue is 0.
- `double = obj.GetForegroundValue ()` - Set the Foreground/Background values of the output. The Foreground value is set when a voxel is occupied. The Background value is set when a voxel is not occupied. The default ForegroundValue is 1. The default BackgroundValue is 0.

- `obj.SetBackgroundValue (double )` - Set the Foreground/Background values of the output. The Foreground value is set when a voxel is occupied. The Background value is set when a voxel is not occupied. The default ForegroundValue is 1. The default BackgroundValue is 0.
- `double = obj.GetBackgroundValue ()` - Set the Foreground/Background values of the output. The Foreground value is set when a voxel is occupied. The Background value is set when a voxel is not occupied. The default ForegroundValue is 1. The default BackgroundValue is 0.

Chapter 36

Visualization Toolkit Infovis Classes

36.1 vtkAddMembershipArray

36.1.1 Usage

This filter takes an input selection, vtkDataSetAttribute information, and data object and adds a bit array to the output vtkDataSetAttributes indicating whether each index was selected or not.

To create an instance of class vtkAddMembershipArray, simply invoke its constructor as follows

```
obj = vtkAddMembershipArray
```

36.1.2 Methods

The class vtkAddMembershipArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAddMembershipArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAddMembershipArray = obj.NewInstance ()`
- `vtkAddMembershipArray = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetFieldType ()` - The field type to add the membership array to.
- `obj.SetFieldType (int )` - The field type to add the membership array to.
- `int = obj.GetFieldTypeMinValue ()` - The field type to add the membership array to.
- `int = obj.GetFieldTypeMaxValue ()` - The field type to add the membership array to.
- `obj.SetOutputArrayName (string )` - The name of the array added to the output vtkDataSetAttributes indicating membership. Defaults to "membership".
- `string = obj.GetOutputArrayName ()` - The name of the array added to the output vtkDataSetAttributes indicating membership. Defaults to "membership".
- `obj.SetInputArrayName (string )`
- `string = obj.GetInputArrayName ()`
- `obj.SetInputValues (vtkAbstractArray )`
- `vtkAbstractArray = obj.GetInputValues ()`

36.2 vtkAdjacencyMatrixToEdgeTable

36.2.1 Usage

Treats a dense 2-way array of doubles as an adacency matrix and converts it into a vtkTable suitable for use as an edge table with vtkTableToGraph.

To create an instance of class vtkAdjacencyMatrixToEdgeTable, simply invoke its constructor as follows

```
obj = vtkAdjacencyMatrixToEdgeTable
```

36.2.2 Methods

The class vtkAdjacencyMatrixToEdgeTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAdjacencyMatrixToEdgeTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAdjacencyMatrixToEdgeTable = obj.NewInstance ()`
- `vtkAdjacencyMatrixToEdgeTable = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetSourceDimension ()` - Specifies whether rows or columns become the "source" in the output edge table. 0 = rows, 1 = columns. Default: 0
- `obj.SetSourceDimension (vtkIdType )` - Specifies whether rows or columns become the "source" in the output edge table. 0 = rows, 1 = columns. Default: 0
- `string = obj.GetValueArrayName ()` - Controls the name of the output table column that contains edge weights. Default: "value"
- `obj.SetValueArrayName (string )` - Controls the name of the output table column that contains edge weights. Default: "value"
- `vtkIdType = obj.GetMinimumCount ()` - Specifies the minimum number of adjacent edges to include for each source vertex. Default: 0
- `obj.SetMinimumCount (vtkIdType )` - Specifies the minimum number of adjacent edges to include for each source vertex. Default: 0
- `double = obj.GetMinimumThreshold ()` - Specifies a minimum threshold that an edge weight must exceed to be included in the output. Default: 0.5
- `obj.SetMinimumThreshold (double )` - Specifies a minimum threshold that an edge weight must exceed to be included in the output. Default: 0.5

36.3 vtkAppendPoints

36.3.1 Usage

vtkAppendPoints is a filter that appends the points and assoicated data of one or more polygonal (vtkPolyData) datasets. This filter can optionally add a new array marking the input index that the point came from.

To create an instance of class vtkAppendPoints, simply invoke its constructor as follows

```
obj = vtkAppendPoints
```

36.3.2 Methods

The class vtkAppendPoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAppendPoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAppendPoints = obj.NewInstance ()`
- `vtkAppendPoints = obj.SafeDownCast (vtkObject o)`
- `obj.SetInputIdArrayName (string )` - Sets the output array name to fill with the input connection index for each point. This provides a way to trace a point back to a particular input. If this is NULL (the default), the array is not generated.
- `string = obj.GetInputIdArrayName ()` - Sets the output array name to fill with the input connection index for each point. This provides a way to trace a point back to a particular input. If this is NULL (the default), the array is not generated.

36.4 vtkApplyColors

36.4.1 Usage

vtkApplyColors performs a coloring of the dataset using default colors, lookup tables, annotations, and/or a selection. The output is a four-component vtkUnsignedCharArray containing RGBA tuples for each element in the dataset. The first input is the dataset to be colored, which may be a vtkTable, vtkGraph subclass, or vtkDataSet subclass. The API of this algorithm refers to "points" and "cells". For vtkGraph, the "points" refer to the graph vertices and "cells" refer to graph edges. For vtkTable, "points" refer to table rows. For vtkDataSet subclasses, the meaning is obvious.

The second (optional) input is a vtkAnnotationLayers object, which stores a list of annotation layers, with each layer holding a list of vtkAnnotation objects. The annotation specifies a subset of data along with other properties, including color. For annotations with color properties, this algorithm will use the color to color elements, using a "top one wins" strategy.

The third (optional) input is a vtkSelection object, meant for specifying the current selection. You can control the color of the selection.

The algorithm takes two input arrays, specified with SetInputArrayToProcess(0, 0, 0, vtkDataObject::FIELD_ASSOCIATION_POINTS, name) and SetInputArrayToProcess(1, 0, 0, vtkDataObject::FIELD_ASSOCIATION_ name). These set the point and cell data arrays to use to color the data with the associated lookup table. For vtkGraph, vtkTable inputs, you would use FIELD_ASSOCIATION_VERTICES, FIELD_ASSOCIATION_EDGES, or FIELD_ASSOCIATION_ROWS as appropriate.

To use the color array generated here, you should do the following:

mapper-¿SetScalarModeToUseCellFieldData(); mapper-¿SelectColorArray("vtkApplyColors color"); mapper-¿SetScalarVisibility(true);

Colors are assigned with the following priorities: ¡ol¿ ¡li¿ If an item is part of the selection, it is colored with that color. ¡li¿ Otherwise, if the item is part of an annotation, it is colored with the color of the final (top) annotation in the set of layers. ¡li¿ Otherwise, if the lookup table is used, it is colored using the lookup table color for the data value of the element. ¡li¿ Otherwise it will be colored with the default color. ¡/ol¿

Note: The opacity of an unselected item is defined by the multiplication of default opacity, lookup table opacity, and annotation opacity, where opacity is taken as a number from 0 to 1. So items will never be more opaque than any of these three opacities. Selected items are always given the selection opacity directly.

To create an instance of class vtkApplyColors, simply invoke its constructor as follows

```
obj = vtkApplyColors
```

36.4.2 Methods

The class vtkApplyColors has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkApplyColors class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkApplyColors = obj.NewInstance ()`
- `vtkApplyColors = obj.SafeDownCast (vtkObject o)`
- `obj.SetPointLookupTable (vtkScalarsToColors lut)` - The lookup table to use for point colors. This is only used if input array 0 is set and UsePointLookupTable is on.
- `vtkScalarsToColors = obj.GetPointLookupTable ()` - The lookup table to use for point colors. This is only used if input array 0 is set and UsePointLookupTable is on.
- `obj.SetUsePointLookupTable (bool )` - If on, uses the point lookup table to set the colors of unannotated, unselected elements of the data.
- `bool = obj.GetUsePointLookupTable ()` - If on, uses the point lookup table to set the colors of unannotated, unselected elements of the data.
- `obj.UsePointLookupTableOn ()` - If on, uses the point lookup table to set the colors of unannotated, unselected elements of the data.
- `obj.UsePointLookupTableOff ()` - If on, uses the point lookup table to set the colors of unannotated, unselected elements of the data.
- `obj.SetScalePointLookupTable (bool )` - If on, uses the range of the data to scale the lookup table range. Otherwise, uses the range defined in the lookup table.
- `bool = obj.GetScalePointLookupTable ()` - If on, uses the range of the data to scale the lookup table range. Otherwise, uses the range defined in the lookup table.
- `obj.ScalePointLookupTableOn ()` - If on, uses the range of the data to scale the lookup table range. Otherwise, uses the range defined in the lookup table.
- `obj.ScalePointLookupTableOff ()` - If on, uses the range of the data to scale the lookup table range. Otherwise, uses the range defined in the lookup table.
- `obj.SetDefaultPointColor (double , double , double )` - The default point color for all unannotated, unselected elements of the data. This is used if UsePointLookupTable is off.
- `obj.SetDefaultPointColor (double  a[3])` - The default point color for all unannotated, unselected elements of the data. This is used if UsePointLookupTable is off.
- `double = obj. GetDefaultPointColor ()` - The default point color for all unannotated, unselected elements of the data. This is used if UsePointLookupTable is off.
- `obj.SetDefaultPointOpacity (double )` - The default point opacity for all unannotated, unselected elements of the data. This is used if UsePointLookupTable is off.
- `double = obj.GetDefaultPointOpacity ()` - The default point opacity for all unannotated, unselected elements of the data. This is used if UsePointLookupTable is off.
- `obj.SetSelectedPointColor (double , double , double )` - The point color for all selected elements of the data. This is used if the selection input is available.

- `obj.SetSelectedPointColor (double a[3])` - The point color for all selected elements of the data. This is used if the selection input is available.
- `double = obj. GetSelectedPointColor ()` - The point color for all selected elements of the data. This is used if the selection input is available.
- `obj.SetSelectedPointOpacity (double )` - The point opacity for all selected elements of the data. This is used if the selection input is available.
- `double = obj.GetSelectedPointOpacity ()` - The point opacity for all selected elements of the data. This is used if the selection input is available.
- `obj.SetPointColorOutputArrayName (string )` - The output array name for the point color RGBA array. Default is "vtkApplyColors color".
- `string = obj.GetPointColorOutputArrayName ()` - The output array name for the point color RGBA array. Default is "vtkApplyColors color".
- `obj.SetCellLookupTable (vtkScalarsToColors lut)` - The lookup table to use for cell colors. This is only used if input array 1 is set and UseCellLookupTable is on.
- `vtkScalarsToColors = obj.GetCellLookupTable ()` - The lookup table to use for cell colors. This is only used if input array 1 is set and UseCellLookupTable is on.
- `obj.SetUseCellLookupTable (bool )` - If on, uses the cell lookup table to set the colors of unannotated, unselected elements of the data.
- `bool = obj.GetUseCellLookupTable ()` - If on, uses the cell lookup table to set the colors of unannotated, unselected elements of the data.
- `obj.UseCellLookupTableOn ()` - If on, uses the cell lookup table to set the colors of unannotated, unselected elements of the data.
- `obj.UseCellLookupTableOff ()` - If on, uses the cell lookup table to set the colors of unannotated, unselected elements of the data.
- `obj.SetScaleCellLookupTable (bool )` - If on, uses the range of the data to scale the lookup table range. Otherwise, uses the range defined in the lookup table.
- `bool = obj.GetScaleCellLookupTable ()` - If on, uses the range of the data to scale the lookup table range. Otherwise, uses the range defined in the lookup table.
- `obj.ScaleCellLookupTableOn ()` - If on, uses the range of the data to scale the lookup table range. Otherwise, uses the range defined in the lookup table.
- `obj.ScaleCellLookupTableOff ()` - If on, uses the range of the data to scale the lookup table range. Otherwise, uses the range defined in the lookup table.
- `obj.SetDefaultCellColor (double , double , double )` - The default cell color for all unannotated, unselected elements of the data. This is used if UseCellLookupTable is off.
- `obj.SetDefaultCellColor (double a[3])` - The default cell color for all unannotated, unselected elements of the data. This is used if UseCellLookupTable is off.
- `double = obj. GetDefaultCellColor ()` - The default cell color for all unannotated, unselected elements of the data. This is used if UseCellLookupTable is off.
- `obj.SetDefaultCellOpacity (double )` - The default cell opacity for all unannotated, unselected elements of the data. This is used if UseCellLookupTable is off.
- `double = obj.GetDefaultCellOpacity ()` - The default cell opacity for all unannotated, unselected elements of the data. This is used if UseCellLookupTable is off.

- `obj.SetSelectedCellColor (double , double , double )` - The cell color for all selected elements of the data. This is used if the selection input is available.
- `obj.SetSelectedCellColor (double a[3])` - The cell color for all selected elements of the data. This is used if the selection input is available.
- `double = obj. GetSelectedCellColor ()` - The cell color for all selected elements of the data. This is used if the selection input is available.
- `obj.SetSelectedCellOpacity (double )` - The cell opacity for all selected elements of the data. This is used if the selection input is available.
- `double = obj.GetSelectedCellOpacity ()` - The cell opacity for all selected elements of the data. This is used if the selection input is available.
- `obj.SetCellColorOutputArrayName (string )` - The output array name for the cell color RGBA array. Default is "vtkApplyColors color".
- `string = obj.GetCellColorOutputArrayName ()` - The output array name for the cell color RGBA array. Default is "vtkApplyColors color".
- `obj.SetUseCurrentAnnotationColor (bool )` - Use the annotation to color the current annotation (i.e. the current selection). Otherwise use the selection color attributes of this filter.
- `bool = obj.GetUseCurrentAnnotationColor ()` - Use the annotation to color the current annotation (i.e. the current selection). Otherwise use the selection color attributes of this filter.
- `obj.UseCurrentAnnotationColorOn ()` - Use the annotation to color the current annotation (i.e. the current selection). Otherwise use the selection color attributes of this filter.
- `obj.UseCurrentAnnotationColorOff ()` - Use the annotation to color the current annotation (i.e. the current selection). Otherwise use the selection color attributes of this filter.

36.5 vtkApplyIcons

36.5.1 Usage

vtkApplyIcons performs a iconing of the dataset using default icons, lookup tables, annotations, and/or a selection. The output is a vtkIntArray containing the icon index for each element in the dataset. The first input is the dataset to be iconed, which may be a vtkTable, vtkGraph subclass, or vtkDataSet subclass.

The second (optional) input is a vtkAnnotationLayers object, which stores a list of annotation layers, with each layer holding a list of vtkAnnotation objects. The annotation specifies a subset of data along with other properties, including icon. For annotations with icon properties, this algorithm will use the icon index of annotated elements, using a "top one wins" strategy.

The third (optional) input is a vtkSelection object, meant for specifying the current selection. You can control the icon of the selection, or whether there is a set of selected icons at a particular offset in the icon sheet.

The algorithm takes an input array, specified with SetInputArrayToProcess(0, 0, 0, vtkDataObject::FIELD_ASSOCIATIO name) This sets data arrays to use to icon the data with the associated lookup table. For vtkGraph and vtkTable inputs, you would use FIELD_ASSOCIATION_VERTICES, FIELD_ASSOCIATION_EDGES, or FIELD_ASSOCIATION_ROWS as appropriate. The icon array will be added to the same set of attributes that the input array came from. If there is no input array, the icon array will be applied to the attributes associated with the AttributeType parameter.

Icons are assigned with the following priorities: ¡ol¿ ¡li¿ If an item is part of the selection, it is glyphed with that icon. ¡li¿ Otherwise, if the item is part of an annotation, it is glyphed with the icon of the final (top) annotation in the set of layers. ¡li¿ Otherwise, if a lookup table is used, it is glyphed using the lookup table icon for the data value of the element. ¡li¿ Otherwise it will be glyphed with the default icon. ¡/ol¿

To create an instance of class vtkApplyIcons, simply invoke its constructor as follows

```
obj = vtkApplyIcons
```

36.5.2 Methods

The class vtkApplyIcons has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkApplyIcons class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkApplyIcons = obj.NewInstance ()`
- `vtkApplyIcons = obj.SafeDownCast (vtkObject o)`
- `obj.SetIconType (double v, int icon)` - Edits the lookup table to use for point icons. This is only used if input array 0 is set and UsePointLookupTable is on.
- `obj.SetIconType (string v, int icon)` - Edits the lookup table to use for point icons. This is only used if input array 0 is set and UsePointLookupTable is on.
- `obj.ClearAllIconTypes ()` - Edits the lookup table to use for point icons. This is only used if input array 0 is set and UsePointLookupTable is on.
- `obj.SetUseLookupTable (bool )` - If on, uses the point lookup table to set the colors of unannotated, unselected elements of the data.
- `bool = obj.GetUseLookupTable ()` - If on, uses the point lookup table to set the colors of unannotated, unselected elements of the data.
- `obj.UseLookupTableOn ()` - If on, uses the point lookup table to set the colors of unannotated, unselected elements of the data.
- `obj.UseLookupTableOff ()` - If on, uses the point lookup table to set the colors of unannotated, unselected elements of the data.
- `obj.SetDefaultIcon (int )` - The default point icon for all unannotated, unselected elements of the data. This is used if UsePointLookupTable is off.
- `int = obj.GetDefaultIcon ()` - The default point icon for all unannotated, unselected elements of the data. This is used if UsePointLookupTable is off.
- `obj.SetSelectedIcon (int )` - The point icon for all selected elements of the data. This is used if the annotation input has a current selection.
- `int = obj.GetSelectedIcon ()` - The point icon for all selected elements of the data. This is used if the annotation input has a current selection.
- `obj.SetIconOutputArrayName (string )` - The output array name for the point icon index array. Default is "vtkApplyIcons icon".
- `string = obj.GetIconOutputArrayName ()` - The output array name for the point icon index array. Default is "vtkApplyIcons icon".
- `obj.SetSelectionMode (int )` - Changes the behavior of the icon to use for selected items. ¡ul¿ ¡li¿SELECTED_ICON uses SelectedIcon as the icon for all selected elements. ¡li¿SELECTED_OFFSET uses SelectedIcon as an offset to add to all selected elements. ¡li¿ANNOTATION_ICON uses the ICON_INDEX() property of the current annotation. ¡li¿IGNORE_SELECTION does not change the icon based on the current selection. ¡/ul¿ The default is IGNORE_SELECTION.

- `int = obj.GetSelectionMode ()` - Changes the behavior of the icon to use for selected items. ¡ul¿ ¡li¿SELECTED_ICON uses SelectedIcon as the icon for all selected elements. ¡li¿SELECTED_OFFSET uses SelectedIcon as an offset to add to all selected elements. ¡li¿ANNOTATION_ICON uses the ICON_INDEX() property of the current annotation. ¡li¿IGNORE_SELECTION does not change the icon based on the current selection. ¡/ul¿ The default is IGNORE_SELECTION.
- `obj.SetSelectionModeToSelectedIcon ()` - Changes the behavior of the icon to use for selected items. ¡ul¿ ¡li¿SELECTED_ICON uses SelectedIcon as the icon for all selected elements. ¡li¿SELECTED_OFFSET uses SelectedIcon as an offset to add to all selected elements. ¡li¿ANNOTATION_ICON uses the ICON_INDEX() property of the current annotation. ¡li¿IGNORE_SELECTION does not change the icon based on the current selection. ¡/ul¿ The default is IGNORE_SELECTION.
- `obj.SetSelectionModeToSelectedOffset ()` - Changes the behavior of the icon to use for selected items. ¡ul¿ ¡li¿SELECTED_ICON uses SelectedIcon as the icon for all selected elements. ¡li¿SELECTED_OFFSET uses SelectedIcon as an offset to add to all selected elements. ¡li¿ANNOTATION_ICON uses the ICON_INDEX() property of the current annotation. ¡li¿IGNORE_SELECTION does not change the icon based on the current selection. ¡/ul¿ The default is IGNORE_SELECTION.
- `obj.SetSelectionModeToAnnotationIcon ()` - Changes the behavior of the icon to use for selected items. ¡ul¿ ¡li¿SELECTED_ICON uses SelectedIcon as the icon for all selected elements. ¡li¿SELECTED_OFFSET uses SelectedIcon as an offset to add to all selected elements. ¡li¿ANNOTATION_ICON uses the ICON_INDEX() property of the current annotation. ¡li¿IGNORE_SELECTION does not change the icon based on the current selection. ¡/ul¿ The default is IGNORE_SELECTION.
- `obj.SetSelectionModeToIgnoreSelection ()` - The attribute type to append the icon array to, used only if the input array is not specified or does not exist. This is set to one of the AttributeTypes enum in vtkDataObject (e.g. POINT, CELL, VERTEX EDGE, FIELD).
- `obj.SetAttributeType (int )` - The attribute type to append the icon array to, used only if the input array is not specified or does not exist. This is set to one of the AttributeTypes enum in vtkDataObject (e.g. POINT, CELL, VERTEX EDGE, FIELD).
- `int = obj.GetAttributeType ()` - The attribute type to append the icon array to, used only if the input array is not specified or does not exist. This is set to one of the AttributeTypes enum in vtkDataObject (e.g. POINT, CELL, VERTEX EDGE, FIELD).

36.6 vtkArcParallelEdgeStrategy

36.6.1 Usage

Parallel edges are drawn as arcs, and self-loops are drawn as ovals. When only one edge connects two vertices it is drawn as a straight line.

To create an instance of class vtkArcParallelEdgeStrategy, simply invoke its constructor as follows

```
obj = vtkArcParallelEdgeStrategy
```

36.6.2 Methods

The class vtkArcParallelEdgeStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArcParallelEdgeStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArcParallelEdgeStrategy = obj.NewInstance ()`

- `vtkArcParallelEdgeStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out.
- `int = obj.GetNumberOfSubdivisions ()` - Get/Set the number of subdivisions on each edge.
- `obj.SetNumberOfSubdivisions (int )` - Get/Set the number of subdivisions on each edge.

36.7 vtkAreaLayout

36.7.1 Usage

vtkAreaLayout assigns sector regions to each vertex in the tree, creating a tree ring. The data is added as a data array with four components per tuple representing the location and size of the sector using the format (StartAngle, EndAngle, innerRadius, outerRadius).

This algorithm relies on a helper class to perform the actual layout. This helper class is a subclass of vtkAreaLayoutStrategy.

.SECTION Thanks Thanks to Jason Shepherd from Sandia National Laboratories for help developing this class.

To create an instance of class vtkAreaLayout, simply invoke its constructor as follows

```
obj = vtkAreaLayout
```

36.7.2 Methods

The class vtkAreaLayout has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAreaLayout class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAreaLayout = obj.NewInstance ()`
- `vtkAreaLayout = obj.SafeDownCast (vtkObject o)`
- `obj.SetSizeArrayName (string name)` - The name for the array created for the area for each vertex. The rectangles are stored in a quadruple float array (startAngle, endAngle, innerRadius, outerRadius). For rectangular layouts, this is (minx, maxx, miny, maxy).
- `string = obj.GetAreaArrayName ()` - The name for the array created for the area for each vertex. The rectangles are stored in a quadruple float array (startAngle, endAngle, innerRadius, outerRadius). For rectangular layouts, this is (minx, maxx, miny, maxy).
- `obj.SetAreaArrayName (string )` - The name for the array created for the area for each vertex. The rectangles are stored in a quadruple float array (startAngle, endAngle, innerRadius, outerRadius). For rectangular layouts, this is (minx, maxx, miny, maxy).
- `bool = obj.GetEdgeRoutingPoints ()` - Whether to output a second output tree with vertex locations appropriate for routing bundled edges. Default is on.
- `obj.SetEdgeRoutingPoints (bool )` - Whether to output a second output tree with vertex locations appropriate for routing bundled edges. Default is on.
- `obj.EdgeRoutingPointsOn ()` - Whether to output a second output tree with vertex locations appropriate for routing bundled edges. Default is on.

- `obj.EdgeRoutingPointsOff ()` - Whether to output a second output tree with vertex locations appropriate for routing bundled edges. Default is on.
- `vtkAreaLayoutStrategy = obj.GetLayoutStrategy ()` - The strategy to use when laying out the tree map.
- `obj.SetLayoutStrategy (vtkAreaLayoutStrategy strategy)` - The strategy to use when laying out the tree map.
- `long = obj.GetMTime ()` - Get the modification time of the layout algorithm.
- `vtkIdType = obj.FindVertex (float pnt[2])` - Get the vertex whose area contains the point, or return -1 if no vertex area covers the point.
- `obj.GetBoundingArea (vtkIdType id, float sinfo)` - The bounding area information for a certain vertex id.

36.8 vtkAreaLayoutStrategy

36.8.1 Usage

All subclasses of this class perform a area layout on a tree. This involves assigning a region to each vertex in the tree, and placing that information in a data array with four components per tuple representing (innerRadius, outerRadius, startAngle, endAngle).

Instances of subclasses of this class may be assigned as the layout strategy to vtkAreaLayout

.SECTION Thanks Thanks to Jason Shepherd from Sandia National Laboratories for help developing this class.

To create an instance of class vtkAreaLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkAreaLayoutStrategy
```

36.8.2 Methods

The class vtkAreaLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAreaLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAreaLayoutStrategy = obj.NewInstance ()`
- `vtkAreaLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout (vtkTree inputTree, vtkDataArray areaArray, vtkDataArray sizeArray)` - Perform the layout of the input tree, and store the sector bounds of each vertex as a tuple in a data array. For radial layout, this is (innerRadius, outerRadius, startAngle, endAngle). For rectangular layout, this is (xmin, xmax, ymin, ymax).

  The sizeArray may be NULL, or may contain the desired size of each vertex in the tree.
- `obj.LayoutEdgePoints (vtkTree inputTree, vtkDataArray areaArray, vtkDataArray sizeArray, vtkTree ed`
- `vtkIdType = obj.FindVertex (vtkTree tree, vtkDataArray array, float pnt[2])` - Returns the vertex id that contains pnt (or -1 if no one contains it)
- `obj.SetShrinkPercentage (double )`

- `double = obj.GetShrinkPercentageMinValue ()`
- `double = obj.GetShrinkPercentageMaxValue ()`
- `double = obj.GetShrinkPercentage ()`

36.9 vtkArrayNorm

36.9.1 Usage

Given an input matrix (vtkTypedArray¡double¿), computes the L-norm for each vector along either dimension, storing the results in a dense output vector (1D vtkDenseArray¡double¿). The caller may optionally request the inverse norm as output (useful for subsequent normalization), and may limit the computation to a "window" of vector elements, to avoid data copying.

.SECTION Thanks Developed by Timothy M. Shead (tshead@sandia.gov) at Sandia National Laboratories.

To create an instance of class vtkArrayNorm, simply invoke its constructor as follows

```
obj = vtkArrayNorm
```

36.9.2 Methods

The class vtkArrayNorm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArrayNorm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArrayNorm = obj.NewInstance ()`
- `vtkArrayNorm = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDimension ()` - Controls the dimension along which norms will be computed. For input matrices, For input matrices, use "0" (rows) or "1" (columns). Default: 0
- `obj.SetDimension (int )` - Controls the dimension along which norms will be computed. For input matrices, For input matrices, use "0" (rows) or "1" (columns). Default: 0
- `int = obj.GetL ()` - Controls the L-value. Default: 2
- `obj.SetL (int value)` - Controls the L-value. Default: 2
- `obj.SetInvert (int )` - Controls whether to invert output values. Default: false
- `int = obj.GetInvert ()` - Controls whether to invert output values. Default: false

36.10 vtkAssignCoordinates

36.10.1 Usage

Given two(or three) arrays take the values in those arrays and simply assign them to the coordinates of the vertices. Yes you could do this with the array calculator, but your mom wears army boots so we're not going to.

To create an instance of class vtkAssignCoordinates, simply invoke its constructor as follows

```
obj = vtkAssignCoordinates
```

36.10.2 Methods

The class vtkAssignCoordinates has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAssignCoordinates class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAssignCoordinates = obj.NewInstance ()`
- `vtkAssignCoordinates = obj.SafeDownCast (vtkObject o)`
- `obj.SetXCoordArrayName (string )` - Set the x coordinate array name.
- `string = obj.GetXCoordArrayName ()` - Set the x coordinate array name.
- `obj.SetYCoordArrayName (string )` - Set the y coordinate array name.
- `string = obj.GetYCoordArrayName ()` - Set the y coordinate array name.
- `obj.SetZCoordArrayName (string )` - Set the z coordinate array name.
- `string = obj.GetZCoordArrayName ()` - Set the z coordinate array name.
- `obj.SetJitter (bool )` - Set if you want a random jitter

36.11 vtkAssignCoordinatesLayoutStrategy

36.11.1 Usage

Uses vtkAssignCoordinates to use values from arrays as the x, y, and z coordinates.

To create an instance of class vtkAssignCoordinatesLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkAssignCoordinatesLayoutStrategy
```

36.11.2 Methods

The class vtkAssignCoordinatesLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAssignCoordinatesLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAssignCoordinatesLayoutStrategy = obj.NewInstance ()`
- `vtkAssignCoordinatesLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetXCoordArrayName (string name)` - The array to use for the x coordinate values.
- `string = obj.GetXCoordArrayName ()` - The array to use for the x coordinate values.
- `obj.SetYCoordArrayName (string name)` - The array to use for the y coordinate values.
- `string = obj.GetYCoordArrayName ()` - The array to use for the y coordinate values.

- `obj.SetZCoordArrayName (string name)` - The array to use for the z coordinate values.
- `string = obj.GetZCoordArrayName ()` - The array to use for the z coordinate values.
- `obj.Layout ()` - Perform the random layout.

36.12 vtkAttributeClustering2DLayoutStrategy

36.12.1 Usage

This class is a density grid based force directed layout strategy. Also please note that 'fast' is relative to quite slow. :) The layout running time is O(V+E) with an extremely high constant. .SECTION Thanks Thanks to Godzilla for not eating my computer so that this class could be written.

To create an instance of class vtkAttributeClustering2DLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkAttributeClustering2DLayoutStrategy
```

36.12.2 Methods

The class vtkAttributeClustering2DLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAttributeClustering2DLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAttributeClustering2DLayoutStrategy = obj.NewInstance ()`
- `vtkAttributeClustering2DLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetVertexAttribute ()` - The name of the array on the vertices, whose values will be used for determining clusters.
- `obj.SetVertexAttribute (string )` - The name of the array on the vertices, whose values will be used for determining clusters.
- `obj.SetRandomSeed (int )` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMinValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMaxValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeed ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `obj.SetMaxNumberOfIterations (int )` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `int = obj.GetMaxNumberOfIterationsMinValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterationsMaxValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterations ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `obj.SetIterationsPerLayout (int )` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayoutMinValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayoutMaxValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayout ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `obj.SetInitialTemperature (float )` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperatureMinValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperatureMaxValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperature ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `obj.SetCoolDownRate (double )` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)

- `double = obj.GetCoolDownRateMinValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)

- `double = obj.GetCoolDownRateMaxValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)

- `double = obj.GetCoolDownRate ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `obj.SetRestDistance (float )` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `float = obj.GetRestDistance ()` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `obj.Initialize ()` - This strategy sets up some data structures for faster processing of each Layout() call
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out. The method can either entirely layout the graph or iteratively lay out the graph. If you have an iterative layout please implement the IsLayoutComplete() method.
- `int = obj.IsLayoutComplete ()`

36.13 vtkBivariateLinearTableThreshold

36.13.1 Usage

Class for filtering the rows of a two numeric columns of a vtkTable. The columns are treated as the two variables of a line. This filter will then iterate through the rows of the table determining if X,Y values pairs are above/below/between/near one or more lines.

The "between" mode checks to see if a row is contained within the convex hull of all of the specified lines. The "near" mode checks if a row is within a distance threshold two one of the specified lines. This class is used in conjunction with various plotting classes, so it is useful to rescale the X,Y axes to a particular range of values. Distance comparisons can be performed in the scaled space by setting the CustomRanges ivar and enabling UseNormalizedDistance.

To create an instance of class vtkBivariateLinearTableThreshold, simply invoke its constructor as follows

```
obj = vtkBivariateLinearTableThreshold
```

36.13.2 Methods

The class vtkBivariateLinearTableThreshold has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBivariateLinearTableThreshold class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBivariateLinearTableThreshold = obj.NewInstance ()`
- `vtkBivariateLinearTableThreshold = obj.SafeDownCast (vtkObject o)`
- `obj.SetInclusive (int )` - Include the line in the threshold. Essentially whether the threshold operation uses ¿ versus ¿=.
- `int = obj.GetInclusive ()` - Include the line in the threshold. Essentially whether the threshold operation uses ¿ versus ¿=.
- `obj.AddColumnToThreshold (vtkIdType column, vtkIdType component)` - Add a numeric column to the pair of columns to be thresholded. Call twice.

- `int = obj.GetNumberOfColumnsToThreshold ()` - Return how many columns have been added. Hopefully 2.
- `obj.ClearColumnsToThreshold ()` - Reset the columns to be thresholded.
- `vtkIdTypeArray = obj.GetSelectedRowIds (int selection)` - Get the output as a table of row ids.
- `obj.Initialize ()` - Reset the columns to threshold, column ranges, etc.
- `obj.AddLineEquation (double p1, double p2)` - Add a line for thresholding from two x,y points.
- `obj.AddLineEquation (double p, double slope)` - Add a line for thresholding in point-slope form.
- `obj.AddLineEquation (double a, double b, double c)` - Add a line for thresholding in implicit form (ax + by + c = 0)
- `obj.ClearLineEquations ()` - Reset the list of line equations.
- `int = obj.GetLinearThresholdType ()` - Set the threshold type. Above: find all rows that are above the specified lines. Below: find all rows that are below the specified lines. Near: find all rows that are near the specified lines. Between: find all rows that are between the specified lines.
- `obj.SetLinearThresholdType (int )` - Set the threshold type. Above: find all rows that are above the specified lines. Below: find all rows that are below the specified lines. Near: find all rows that are near the specified lines. Between: find all rows that are between the specified lines.
- `obj.SetLinearThresholdTypeToAbove ()` - Set the threshold type. Above: find all rows that are above the specified lines. Below: find all rows that are below the specified lines. Near: find all rows that are near the specified lines. Between: find all rows that are between the specified lines.
- `obj.SetLinearThresholdTypeToBelow ()` - Set the threshold type. Above: find all rows that are above the specified lines. Below: find all rows that are below the specified lines. Near: find all rows that are near the specified lines. Between: find all rows that are between the specified lines.
- `obj.SetLinearThresholdTypeToNear ()` - Set the threshold type. Above: find all rows that are above the specified lines. Below: find all rows that are below the specified lines. Near: find all rows that are near the specified lines. Between: find all rows that are between the specified lines.
- `obj.SetLinearThresholdTypeToBetween ()` - Manually access the maximum/minimum x,y values. This is used in conjunction with UseNormalizedDistance when determining if a row passes the threshold.
- `obj.SetColumnRanges (double , double )` - Manually access the maximum/minimum x,y values. This is used in conjunction with UseNormalizedDistance when determining if a row passes the threshold.
- `obj.SetColumnRanges (double a[2])` - Manually access the maximum/minimum x,y values. This is used in conjunction with UseNormalizedDistance when determining if a row passes the threshold.
- `double = obj. GetColumnRanges ()` - Manually access the maximum/minimum x,y values. This is used in conjunction with UseNormalizedDistance when determining if a row passes the threshold.
- `obj.SetDistanceThreshold (double )` - The Cartesian distance within which a point will pass the near threshold.
- `double = obj.GetDistanceThreshold ()` - The Cartesian distance within which a point will pass the near threshold.
- `obj.SetUseNormalizedDistance (int )` - Renormalize the space of the data such that the X and Y axes are "square" over the specified ColumnRanges. This essentially scales the data space so that ColumnRanges[1]-ColumnRanges[0] = 1.0 and ColumnRanges[3]-ColumnRanges[2] = 1.0. Used for scatter plot distance calculations. Be sure to set DistanceThreshold accordingly, when used.

- `int = obj.GetUseNormalizedDistance ()` - Renormalize the space of the data such that the X and Y axes are "square" over the specified ColumnRanges. This essentially scales the data space so that ColumnRanges[1]-ColumnRanges[0] = 1.0 and ColumnRanges[3]-ColumnRanges[2] = 1.0. Used for scatter plot distance calculations. Be sure to set DistanceThreshold accordingly, when used.
- `obj.UseNormalizedDistanceOn ()` - Renormalize the space of the data such that the X and Y axes are "square" over the specified ColumnRanges. This essentially scales the data space so that ColumnRanges[1]-ColumnRanges[0] = 1.0 and ColumnRanges[3]-ColumnRanges[2] = 1.0. Used for scatter plot distance calculations. Be sure to set DistanceThreshold accordingly, when used.
- `obj.UseNormalizedDistanceOff ()` - Renormalize the space of the data such that the X and Y axes are "square" over the specified ColumnRanges. This essentially scales the data space so that ColumnRanges[1]-ColumnRanges[0] = 1.0 and ColumnRanges[3]-ColumnRanges[2] = 1.0. Used for scatter plot distance calculations. Be sure to set DistanceThreshold accordingly, when used.

36.14 vtkBivariateStatisticsAlgorithm

36.14.1 Usage

This class specializes statistics algorithms to the bivariate case, where a number of pairs of columns of interest can be selected in the input data set. This is done by the means of the following functions:

ResetColumns() - reset the list of columns of interest. Add/RemoveColum(namColX, namColY) - try to add/remove column pair (namColX, namColy) to/from the list. SetColumnStatus (namCol, status) - mostly for UI wrapping purposes, try to add/remove (depending on status) namCol from a list of buffered columns, from which all possible pairs are generated. The verb "try" is used in the sense that neither attempting to repeat an existing entry nor to remove a non-existent entry will work.

.SECTION Thanks Thanks to Philippe Pebay and David Thompson from Sandia National Laboratories for implementing this class.

To create an instance of class vtkBivariateStatisticsAlgorithm, simply invoke its constructor as follows

```
obj = vtkBivariateStatisticsAlgorithm
```

36.14.2 Methods

The class vtkBivariateStatisticsAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBivariateStatisticsAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBivariateStatisticsAlgorithm = obj.NewInstance ()`
- `vtkBivariateStatisticsAlgorithm = obj.SafeDownCast (vtkObject o)`
- `obj.AddColumnPair (string namColX, string namColY)` - Convenience method to create a request with a single column name pair (namColX, namColY) in a single call; this is the preferred method to select columns pairs, ensuring selection consistency (a pair of columns per request).

  Unlike SetColumnStatus(), you need not call RequestSelectedColumns() after AddColumnPair().

  Warning: namColX and namColY are only checked for their validity as strings; no check is made that either are valid column names.
- `int = obj.RequestSelectedColumns ()` - Use the current column status values to produce a new request for statistics to be produced when RequestData() is called. Unlike the superclass implementation, this version adds a new request for every possible pairing of the selected columns instead of a single request containing all the columns.

36.15 vtkBoxLayoutStrategy

36.15.1 Usage

vtkBoxLayoutStrategy recursively partitions the space for children vertices in a tree-map into square regions (or regions very close to a square).

.SECTION Thanks Thanks to Brian Wylie from Sandia National Laboratories for creating this class.

To create an instance of class vtkBoxLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkBoxLayoutStrategy
```

36.15.2 Methods

The class vtkBoxLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBoxLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBoxLayoutStrategy = obj.NewInstance ()`
- `vtkBoxLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout (vtkTree inputTree, vtkDataArray coordsArray, vtkDataArray sizeArray)` - Perform the layout of a tree and place the results as 4-tuples in coordsArray (Xmin, Xmax, Ymin, Ymax).

36.16 vtkChacoGraphReader

36.16.1 Usage

vtkChacoGraphReader reads in files in the Chaco format into a vtkGraph. An example is the following ¡code¿ 10 13 2 6 10 1 3 2 4 8 3 5 4 6 10 1 5 7 6 8 3 7 9 8 10 1 5 9 ¡/code¿ The first line specifies the number of vertices and edges in the graph. Each additional line contains the vertices adjacent to a particular vertex. In this example, vertex 1 is adjacent to 2, 6 and 10, vertex 2 is adjacent to 1 and 3, etc. Since Chaco ids start at 1 and VTK ids start at 0, the vertex ids in the vtkGraph will be 1 less than the Chaco ids.

To create an instance of class vtkChacoGraphReader, simply invoke its constructor as follows

```
obj = vtkChacoGraphReader
```

36.16.2 Methods

The class vtkChacoGraphReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkChacoGraphReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkChacoGraphReader = obj.NewInstance ()`
- `vtkChacoGraphReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileName ()` - The Chaco file name.
- `obj.SetFileName (string )` - The Chaco file name.

36.17 vtkCircularLayoutStrategy

36.17.1 Usage

Assigns points to the vertices around a circle with unit radius.

To create an instance of class vtkCircularLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkCircularLayoutStrategy
```

36.17.2 Methods

The class vtkCircularLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCircularLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCircularLayoutStrategy = obj.NewInstance ()`
- `vtkCircularLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout ()` - Perform the layout.

36.18 vtkClustering2DLayoutStrategy

36.18.1 Usage

This class is a density grid based force directed layout strategy. Also please note that 'fast' is relative to quite slow. :) The layout running time is O(V+E) with an extremely high constant. .SECTION Thanks Thanks to Godzilla for not eating my computer so that this class could be written.

To create an instance of class vtkClustering2DLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkClustering2DLayoutStrategy
```

36.18.2 Methods

The class vtkClustering2DLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkClustering2DLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkClustering2DLayoutStrategy = obj.NewInstance ()`
- `vtkClustering2DLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetRandomSeed (int )` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMinValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMaxValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.

- `int = obj.GetRandomSeed ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.

- `obj.SetMaxNumberOfIterations (int )` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterationsMinValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterationsMaxValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterations ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `obj.SetIterationsPerLayout (int )` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayoutMinValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayoutMaxValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayout ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `obj.SetInitialTemperature (float )` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperatureMinValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperatureMaxValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperature ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `obj.SetCoolDownRate (double )` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRateMinValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRateMaxValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRate ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `obj.SetRestDistance (float )` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `float = obj.GetRestDistance ()` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `obj.Initialize ()` - This strategy sets up some data structures for faster processing of each Layout() call
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out. The method can either entirely layout the graph or iteratively lay out the graph. If you have an iterative layout please implement the IsLayoutComplete() method.
- `int = obj.IsLayoutComplete ()`

36.19 vtkCollapseGraph

36.19.1 Usage

vtkCollapseGraph "collapses" vertices onto their neighbors, while maintaining connectivity. Two inputs are required - a graph (directed or undirected), and a vertex selection that can be converted to indices.

Conceptually, each of the vertices specified in the input selection expands, "swallowing" adacent vertices. Edges to-or-from the "swallowed" vertices become edges to-or-from the expanding vertices, maintaining the overall graph connectivity.

In the case of directed graphs, expanding vertices only swallow vertices that are connected via out edges. This rule provides intuitive behavior when working with trees, so that "child" vertices collapse into their parents when the parents are part of the input selection.

Input port 0: graph Input port 1: selection

To create an instance of class vtkCollapseGraph, simply invoke its constructor as follows

```
obj = vtkCollapseGraph
```

36.19.2 Methods

The class vtkCollapseGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCollapseGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCollapseGraph = obj.NewInstance ()`
- `vtkCollapseGraph = obj.SafeDownCast (vtkObject o)`
- `obj.SetGraphConnection (vtkAlgorithmOutput )`
- `obj.SetSelectionConnection (vtkAlgorithmOutput )`

36.20 vtkCollapseVerticesByArray

36.20.1 Usage

vtkCollapseVerticesByArray is a class which collapses the graph using a vertex array as the key. So if the graph has vertices sharing common traits then this class combines all these vertices into one. This class does not perform aggregation on vertex data but allow to do so for edge data. Users can choose one or more edge data arrays for aggregation using AddAggregateEdgeArray function.

.SECTION Thanks

To create an instance of class vtkCollapseVerticesByArray, simply invoke its constructor as follows

```
obj = vtkCollapseVerticesByArray
```

36.20.2 Methods

The class vtkCollapseVerticesByArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCollapseVerticesByArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCollapseVerticesByArray = obj.NewInstance ()`
- `vtkCollapseVerticesByArray = obj.SafeDownCast (vtkObject o)`
- `bool = obj.GetAllowSelfLoops ()` - Boolean to allow self loops during collapse.
- `obj.SetAllowSelfLoops (bool )` - Boolean to allow self loops during collapse.
- `obj.AllowSelfLoopsOn ()` - Boolean to allow self loops during collapse.
- `obj.AllowSelfLoopsOff ()` - Boolean to allow self loops during collapse.
- `obj.AddAggregateEdgeArray (string arrName)` - Add arrays on which aggregation of data is allowed. Default if replaced by the last value.
- `obj.ClearAggregateEdgeArray ()` - Clear the list of arrays on which aggregation was set to allow.
- `string = obj.GetVertexArray ()` - Set the array using which perform the collapse.
- `obj.SetVertexArray (string )` - Set the array using which perform the collapse.

36.21 vtkCommunity2DLayoutStrategy

36.21.1 Usage

This class is a density grid based force directed layout strategy. Also please note that 'fast' is relative to quite slow. :) The layout running time is O(V+E) with an extremely high constant. .SECTION Thanks Thanks to Godzilla for not eating my computer so that this class could be written.

To create an instance of class vtkCommunity2DLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkCommunity2DLayoutStrategy
```

36.21.2 Methods

The class vtkCommunity2DLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCommunity2DLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCommunity2DLayoutStrategy = obj.NewInstance ()`
- `vtkCommunity2DLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetRandomSeed (int )` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMinValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMaxValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeed ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `obj.SetMaxNumberOfIterations (int )` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `int = obj.GetMaxNumberOfIterationsMinValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `int = obj.GetMaxNumberOfIterationsMaxValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `int = obj.GetMaxNumberOfIterations ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `obj.SetIterationsPerLayout (int )` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)
- `int = obj.GetIterationsPerLayoutMinValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)
- `int = obj.GetIterationsPerLayoutMaxValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)
- `int = obj.GetIterationsPerLayout ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)
- `obj.SetInitialTemperature (float )` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `float = obj.GetInitialTemperatureMinValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `float = obj.GetInitialTemperatureMaxValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `float = obj.GetInitialTemperature ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `obj.SetCoolDownRate (double )` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRateMinValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRateMaxValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRate ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `obj.SetRestDistance (float )` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `float = obj.GetRestDistance ()` - Manually set the resting distance. Otherwise the distance is computed automatically.

- `obj.Initialize ()` - This strategy sets up some data structures for faster processing of each Layout() call
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out. The method can either entirely layout the graph or iteratively lay out the graph. If you have an iterative layout please implement the IsLayoutComplete() method.
- `int = obj.IsLayoutComplete ()` - Get/Set the community array name
- `string = obj.GetCommunityArrayName ()` - Get/Set the community array name
- `obj.SetCommunityArrayName (string )` - Get/Set the community array name
- `obj.SetCommunityStrength (float )` - Set the community 'strength'. The default is '1' which means vertices in the same community will be placed close together, values closer to .1 (minimum) will mean a layout closer to traditional force directed.
- `float = obj.GetCommunityStrengthMinValue ()` - Set the community 'strength'. The default is '1' which means vertices in the same community will be placed close together, values closer to .1 (minimum) will mean a layout closer to traditional force directed.
- `float = obj.GetCommunityStrengthMaxValue ()` - Set the community 'strength'. The default is '1' which means vertices in the same community will be placed close together, values closer to .1 (minimum) will mean a layout closer to traditional force directed.
- `float = obj.GetCommunityStrength ()` - Set the community 'strength'. The default is '1' which means vertices in the same community will be placed close together, values closer to .1 (minimum) will mean a layout closer to traditional force directed.

36.22 vtkComputeHistogram2DOutliers

36.22.1 Usage

This class takes a table and one or more vtkImageData histograms as input and computes the outliers in that data. In general it does so by identifying histogram bins that are removed by a median (salt and pepper) filter and below a threshold. This threshold is automatically identified to retrieve a number of outliers close to a user-determined value. This value is set by calling SetPreferredNumberOfOutliers(int).

The image data input can come either as a multiple vtkImageData via the repeatable INPUT_HISTOGRAM_IMAGE_DATA port, or as a single vtkMultiBlockDataSet containing vtkImageData objects as blocks. One or the other must be set, not both (or neither).

The output can be retrieved as a set of row ids in a vtkSelection or as a vtkTable containing the actual outlier row data.

To create an instance of class vtkComputeHistogram2DOutliers, simply invoke its constructor as follows

```
obj = vtkComputeHistogram2DOutliers
```

36.22.2 Methods

The class vtkComputeHistogram2DOutliers has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkComputeHistogram2DOutliers class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkComputeHistogram2DOutliers = obj.NewInstance ()`

- `vtkComputeHistogram2DOutliers = obj.SafeDownCast (vtkObject o)`
- `obj.SetPreferredNumberOfOutliers (int )`
- `int = obj.GetPreferredNumberOfOutliers ()`
- `vtkTable = obj.GetOutputTable ()`
- `obj.SetInputTableConnection (vtkAlgorithmOutput cxn)` - Set the input histogram data as a (repeatable) vtkImageData
- `obj.SetInputHistogramImageDataConnection (vtkAlgorithmOutput cxn)` - Set the input histogram data as a vtkMultiBlockData set containing multiple vtkImageData objects.
- `obj.SetInputHistogramMultiBlockConnection (vtkAlgorithmOutput cxn)`

36.23 vtkConeLayoutStrategy

36.23.1 Usage

vtkConeLayoutStrategy positions the nodes of a tree(forest) in 3D space based on the cone-tree approach first described by Robertson, Mackinlay and Card in Proc. CHI'91. This implementation incorporates refinements to the layout developed by Carriere and Kazman, and by Auber.

The input graph must be a forest (i.e. a set of trees, or a single tree); in the case of a forest, the input will be converted to a single tree by introducing a new root node, and connecting each root in the input forest to the meta-root. The tree is then laid out, after which the meta-root is removed.

The cones are positioned so that children lie in planes parallel to the X-Y plane, with the axis of cones parallel to Z, and with Z coordinate increasing with distance of nodes from the root.

.SECTION Thanks Thanks to David Duke from the University of Leeds for providing this implementation.

To create an instance of class vtkConeLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkConeLayoutStrategy
```

36.23.2 Methods

The class vtkConeLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkConeLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkConeLayoutStrategy = obj.NewInstance ()`
- `vtkConeLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetCompactness (float )` - Determine the compactness, the ratio between the average width of a cone in the tree, and the height of the cone. The default setting is 0.75 which (empirically) seems reasonable, but this will need adapting depending on the data.
- `float = obj.GetCompactness ()` - Determine the compactness, the ratio between the average width of a cone in the tree, and the height of the cone. The default setting is 0.75 which (empirically) seems reasonable, but this will need adapting depending on the data.
- `obj.SetCompression (int )` - Determine if layout should be compressed, i.e. the layout puts children closer together, possibly allowing sub-trees to overlap. This is useful if the tree is actually the spanning tree of a graph. For "real" trees, non-compressed layout is best, and is the default.

- `int = obj.GetCompression ()` - Determine if layout should be compressed, i.e. the layout puts children closer together, possibly allowing sub-trees to overlap. This is useful if the tree is actually the spanning tree of a graph. For "real" trees, non-compressed layout is best, and is the default.
- `obj.CompressionOn ()` - Determine if layout should be compressed, i.e. the layout puts children closer together, possibly allowing sub-trees to overlap. This is useful if the tree is actually the spanning tree of a graph. For "real" trees, non-compressed layout is best, and is the default.
- `obj.CompressionOff ()` - Determine if layout should be compressed, i.e. the layout puts children closer together, possibly allowing sub-trees to overlap. This is useful if the tree is actually the spanning tree of a graph. For "real" trees, non-compressed layout is best, and is the default.
- `obj.SetSpacing (float )` - Set the spacing parameter that affects space between layers of the tree. If compression is on, Spacing is the actual distance between layers. If compression is off, actual distance also includes a factor of the compactness and maximum cone radius.
- `float = obj.GetSpacing ()` - Set the spacing parameter that affects space between layers of the tree. If compression is on, Spacing is the actual distance between layers. If compression is off, actual distance also includes a factor of the compactness and maximum cone radius.
- `obj.Layout ()` - Perform the layout.

36.24 vtkConstrained2DLayoutStrategy

36.24.1 Usage

This class is a density grid based force directed layout strategy. Also please note that 'fast' is relative to quite slow. :) The layout running time is O(V+E) with an extremely high constant. .SECTION Thanks We would like to thank Mothra for distracting Godzilla while we wrote this class.

To create an instance of class vtkConstrained2DLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkConstrained2DLayoutStrategy
```

36.24.2 Methods

The class vtkConstrained2DLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkConstrained2DLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkConstrained2DLayoutStrategy = obj.NewInstance ()`
- `vtkConstrained2DLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetRandomSeed (int )` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMinValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMaxValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeed ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.

- `obj.SetMaxNumberOfIterations (int )` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `int = obj.GetMaxNumberOfIterationsMinValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `int = obj.GetMaxNumberOfIterationsMaxValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `int = obj.GetMaxNumberOfIterations ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `obj.SetIterationsPerLayout (int )` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)
- `int = obj.GetIterationsPerLayoutMinValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)
- `int = obj.GetIterationsPerLayoutMaxValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)
- `int = obj.GetIterationsPerLayout ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)
- `obj.SetInitialTemperature (float )` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `float = obj.GetInitialTemperatureMinValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `float = obj.GetInitialTemperatureMaxValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `float = obj.GetInitialTemperature ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)
- `obj.SetCoolDownRate (double )` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)

- `double = obj.GetCoolDownRateMinValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRateMaxValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRate ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `obj.SetRestDistance (float )` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `float = obj.GetRestDistance ()` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `obj.Initialize ()` - This strategy sets up some data structures for faster processing of each Layout() call
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out. The method can either entirely layout the graph or iteratively lay out the graph. If you have an iterative layout please implement the IsLayoutComplete() method.
- `int = obj.IsLayoutComplete ()` - Set/Get the input constraint array name. If no input array name is set then the name 'constraint' is used.
- `obj.SetInputArrayName (string )` - Set/Get the input constraint array name. If no input array name is set then the name 'constraint' is used.
- `string = obj.GetInputArrayName ()` - Set/Get the input constraint array name. If no input array name is set then the name 'constraint' is used.

36.25 vtkContingencyStatistics

36.25.1 Usage

Given a pair of columns of interest, this class provides the following functionalities, depending on the execution mode it is executed in: * Learn: calculate contigency tables and corresponding discrete bivariate probability distribution. * Assess: given two columns of interest with the same number of entries as input in port INPUT_DATA, and a corresponding bivariate probability distribution,

.SECTION Thanks Thanks to Philippe Pebay and David Thompson from Sandia National Laboratories for implementing this class.

To create an instance of class vtkContingencyStatistics, simply invoke its constructor as follows

```
obj = vtkContingencyStatistics
```

36.25.2 Methods

The class vtkContingencyStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkContingencyStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkContingencyStatistics = obj.NewInstance ()`
- `vtkContingencyStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.Aggregate (vtkDataObjectCollection , vtkDataObject )` - Given a collection of models, calculate aggregate model NB: not implemented

36.26 vtkCorrelativeStatistics

36.26.1 Usage

Given a selection of pairs of columns of interest, this class provides the following functionalities, depending on the execution mode it is executed in: * Learn: calculate means, unbiased variance and covariance estimators of column pairs, and corresponding linear regressions and linear correlation coefficient. More precisely, Learn calculates the sums; if finalize is set to true (default), the final statistics are calculated with the function CalculateFromSums. Otherwise, only raw sums are output; this option is made for efficient parallel calculations. Note that CalculateFromSums is a static function, so that it can be used directly with no need to instantiate a vtkCorrelativeStatistics object. * Assess: given two data vectors X and Y with the same number of entries as input in port INPUT_DATA, and reference means, variances, and covariance, along with an acceptable threshold t¿1, assess all pairs of values of (X,Y) whose relative PDF (assuming a bivariate Gaussian model) is below t.

.SECTION Thanks Thanks to Philippe Pebay and David Thompson from Sandia National Laboratories for implementing this class.

To create an instance of class vtkCorrelativeStatistics, simply invoke its constructor as follows

```
obj = vtkCorrelativeStatistics
```

36.26.2 Methods

The class vtkCorrelativeStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCorrelativeStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCorrelativeStatistics = obj.NewInstance ()`
- `vtkCorrelativeStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.Aggregate (vtkDataObjectCollection , vtkDataObject )` - Given a collection of models, calculate aggregate model

36.27 vtkCosmicTreeLayoutStrategy

36.27.1 Usage

This layout strategy takes an input tree and places all the children of a node into a containing circle. The placement is such that each child placed can be represented with a circle tangent to the containing circle and (usually) 2 other children. The interior of the circle is left empty so that graph edges drawn on top of the tree will not obfuscate the tree. However, when one child is much larger than all the others, it may

encroach on the center of the containing circle; that's OK, because it's large enough not to be obscured by edges drawn atop it.

.SECTION Thanks Thanks to the galaxy and David Thompson hierarchically nested inside it for inspiring this layout strategy.

To create an instance of class vtkCosmicTreeLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkCosmicTreeLayoutStrategy
```

36.27.2 Methods

The class vtkCosmicTreeLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCosmicTreeLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCosmicTreeLayoutStrategy = obj.NewInstance ()`
- `vtkCosmicTreeLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout ()` - Perform the layout.
- `obj.SetSizeLeafNodesOnly (int )` - Should node size specifications be obeyed at leaf nodes only or (with scaling as required to meet constraints) at every node in the tree? This defaults to true, so that leaf nodes are scaled according to the size specification provided, and the parent node sizes are calculated by the algorithm.
- `int = obj.GetSizeLeafNodesOnly ()` - Should node size specifications be obeyed at leaf nodes only or (with scaling as required to meet constraints) at every node in the tree? This defaults to true, so that leaf nodes are scaled according to the size specification provided, and the parent node sizes are calculated by the algorithm.
- `obj.SizeLeafNodesOnlyOn ()` - Should node size specifications be obeyed at leaf nodes only or (with scaling as required to meet constraints) at every node in the tree? This defaults to true, so that leaf nodes are scaled according to the size specification provided, and the parent node sizes are calculated by the algorithm.
- `obj.SizeLeafNodesOnlyOff ()` - Should node size specifications be obeyed at leaf nodes only or (with scaling as required to meet constraints) at every node in the tree? This defaults to true, so that leaf nodes are scaled according to the size specification provided, and the parent node sizes are calculated by the algorithm.
- `obj.SetLayoutDepth (int )` - How many levels of the tree should be laid out? For large trees, you may wish to set the root and maximum depth in order to retrieve the layout for the visible portion of the tree. When this value is zero or negative, all nodes below and including the LayoutRoot will be presented. This defaults to 0.
- `int = obj.GetLayoutDepth ()` - How many levels of the tree should be laid out? For large trees, you may wish to set the root and maximum depth in order to retrieve the layout for the visible portion of the tree. When this value is zero or negative, all nodes below and including the LayoutRoot will be presented. This defaults to 0.
- `obj.SetLayoutRoot (vtkIdType )` - What is the top-most tree node to lay out? This node will become the largest containing circle in the layout. Use this in combination with SetLayoutDepth to retrieve the layout of a subtree of interest for rendering. Setting LayoutRoot to a negative number signals that the root node of the tree should be used as the root node of the layout. This defaults to -1.

- `vtkIdType = obj.GetLayoutRoot ()` - What is the top-most tree node to lay out? This node will become the largest containing circle in the layout. Use this in combination with SetLayoutDepth to retrieve the layout of a subtree of interest for rendering. Setting LayoutRoot to a negative number signals that the root node of the tree should be used as the root node of the layout. This defaults to -1.
- `obj.SetNodeSizeArrayName (string )` - Set the array to be used for sizing nodes. If this is set to an empty string or NULL (the default), then all leaf nodes (or all nodes, when SizeLeafNodesOnly is false) will be assigned a unit size.
- `string = obj.GetNodeSizeArrayName ()` - Set the array to be used for sizing nodes. If this is set to an empty string or NULL (the default), then all leaf nodes (or all nodes, when SizeLeafNodesOnly is false) will be assigned a unit size.

36.28 vtkDataObjectToTable

36.28.1 Usage

This filter is used to extract either the field, cell or point data of any data object as a table.

To create an instance of class vtkDataObjectToTable, simply invoke its constructor as follows

```
obj = vtkDataObjectToTable
```

36.28.2 Methods

The class vtkDataObjectToTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObjectToTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataObjectToTable = obj.NewInstance ()`
- `vtkDataObjectToTable = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetFieldType ()` - The field type to copy into the output table. Should be one of FIELD_DATA, POINT_DATA, CELL_DATA, VERTEX_DATA, EDGE_DATA.
- `obj.SetFieldType (int )` - The field type to copy into the output table. Should be one of FIELD_DATA, POINT_DATA, CELL_DATA, VERTEX_DATA, EDGE_DATA.
- `int = obj.GetFieldTypeMinValue ()` - The field type to copy into the output table. Should be one of FIELD_DATA, POINT_DATA, CELL_DATA, VERTEX_DATA, EDGE_DATA.
- `int = obj.GetFieldTypeMaxValue ()` - The field type to copy into the output table. Should be one of FIELD_DATA, POINT_DATA, CELL_DATA, VERTEX_DATA, EDGE_DATA.

36.29 vtkDelimitedTextReader

36.29.1 Usage

vtkDelimitedTextReader is an interface for pulling in data from a flat, delimited ascii or unicode text file (delimiter can be any character).

The behavior of the reader with respect to ascii or unicode input is controlled by the SetUnicodeCharacterSet() method. By default (without calling SetUnicodeCharacterSet()), the reader will expect to read

ascii text and will output vtkStdString columns. Use the Set and Get methods to set delimiters that do not contain UTF8 in the name when operating the reader in default ascii mode. If the SetUnicodeCharacterSet() method is called, the reader will output vtkUnicodeString columns in the output table. In addition, it is necessary to use the Set and Get methods that contain UTF8 in the name to specify delimiters when operating in unicode mode.

This class emits ProgressEvent for every 100 lines it reads.

.SECTION Thanks Thanks to Andy Wilson, Brian Wylie, Tim Shead, and Thomas Otahal from Sandia National Laboratories for implementing this class.

To create an instance of class vtkDelimitedTextReader, simply invoke its constructor as follows

```
obj = vtkDelimitedTextReader
```

36.29.2 Methods

The class vtkDelimitedTextReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDelimitedTextReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDelimitedTextReader = obj.NewInstance ()`
- `vtkDelimitedTextReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileName ()`
- `obj.SetFileName (string )`
- `string = obj.GetUnicodeCharacterSet ()` - Specifies the character set used in the input file. Valid character set names will be drawn from the list maintained by the Internet Assigned Name Authority at

  http://www.iana.org/assignments/character-sets

  Where multiple aliases are provided for a character set, the preferred MIME name will be used. vtkUnicodeDelimitedTextReader currently supports "US-ASCII", "UTF-8", "UTF-16", "UTF-16BE", and "UTF-16LE" character sets.

- `obj.SetUnicodeCharacterSet (string )` - Specifies the character set used in the input file. Valid character set names will be drawn from the list maintained by the Internet Assigned Name Authority at

  http://www.iana.org/assignments/character-sets

  Where multiple aliases are provided for a character set, the preferred MIME name will be used. vtkUnicodeDelimitedTextReader currently supports "US-ASCII", "UTF-8", "UTF-16", "UTF-16BE", and "UTF-16LE" character sets.

- `obj.SetUTF8RecordDelimiters (string delimiters)` - Specify the character(s) that will be used to separate records. The order of characters in the string does not matter. Defaults to " ◦
  n".

- `string = obj.GetUTF8RecordDelimiters ()` - Specify the character(s) that will be used to separate records. The order of characters in the string does not matter. Defaults to " ◦
  n".

- `obj.SetFieldDelimiterCharacters (string )` - Specify the character(s) that will be used to separate fields. For example, set this to "," for a comma-separated value file. Set it to ".:;" for a file where columns can be separated by a period, colon or semicolon. The order of the characters in the string does not matter. Defaults to a comma.
- `string = obj.GetFieldDelimiterCharacters ()` - Specify the character(s) that will be used to separate fields. For example, set this to "," for a comma-separated value file. Set it to ".:;" for a file where columns can be separated by a period, colon or semicolon. The order of the characters in the string does not matter. Defaults to a comma.
- `obj.SetUTF8FieldDelimiters (string delimiters)`
- `string = obj.GetUTF8FieldDelimiters ()`
- `char = obj.GetStringDelimiter ()` - Get/set the character that will begin and end strings. Microsoft Excel, for example, will export the following format:

  "First Field","Second Field","Field, With, Commas","Fourth Field"

  The third field has a comma in it. By using a string delimiter, this will be correctly read. The delimiter defaults to "".
- `obj.SetStringDelimiter (char )` - Get/set the character that will begin and end strings. Microsoft Excel, for example, will export the following format:

  "First Field","Second Field","Field, With, Commas","Fourth Field"

  The third field has a comma in it. By using a string delimiter, this will be correctly read. The delimiter defaults to "".
- `obj.SetUTF8StringDelimiters (string delimiters)`
- `string = obj.GetUTF8StringDelimiters ()`
- `obj.SetUseStringDelimiter (bool )` - Set/get whether to use the string delimiter. Defaults to on.
- `bool = obj.GetUseStringDelimiter ()` - Set/get whether to use the string delimiter. Defaults to on.
- `obj.UseStringDelimiterOn ()` - Set/get whether to use the string delimiter. Defaults to on.
- `obj.UseStringDelimiterOff ()` - Set/get whether to use the string delimiter. Defaults to on.
- `bool = obj.GetHaveHeaders ()` - Set/get whether to treat the first line of the file as headers.
- `obj.SetHaveHeaders (bool )` - Set/get whether to treat the first line of the file as headers.
- `obj.SetMergeConsecutiveDelimiters (bool )` - Set/get whether to merge successive delimiters. Use this if (for example) your fields are separated by spaces but you don't know exactly how many.
- `bool = obj.GetMergeConsecutiveDelimiters ()` - Set/get whether to merge successive delimiters. Use this if (for example) your fields are separated by spaces but you don't know exactly how many.
- `obj.MergeConsecutiveDelimitersOn ()` - Set/get whether to merge successive delimiters. Use this if (for example) your fields are separated by spaces but you don't know exactly how many.
- `obj.MergeConsecutiveDelimitersOff ()` - Set/get whether to merge successive delimiters. Use this if (for example) your fields are separated by spaces but you don't know exactly how many.
- `vtkIdType = obj.GetMaxRecords ()` - Specifies the maximum number of records to read from the file. Limiting the number of records to read is useful for previewing the contents of a file.
- `obj.SetMaxRecords (vtkIdType )` - Specifies the maximum number of records to read from the file. Limiting the number of records to read is useful for previewing the contents of a file.

- `obj.SetDetectNumericColumns (bool )` - When set to true, the reader will detect numeric columns and create vtkDoubleArray or vtkIntArray for those instead of vtkStringArray. Default is off.
- `bool = obj.GetDetectNumericColumns ()` - When set to true, the reader will detect numeric columns and create vtkDoubleArray or vtkIntArray for those instead of vtkStringArray. Default is off.
- `obj.DetectNumericColumnsOn ()` - When set to true, the reader will detect numeric columns and create vtkDoubleArray or vtkIntArray for those instead of vtkStringArray. Default is off.
- `obj.DetectNumericColumnsOff ()` - When set to true, the reader will detect numeric columns and create vtkDoubleArray or vtkIntArray for those instead of vtkStringArray. Default is off.
- `obj.SetPedigreeIdArrayName (string )` - The name of the array for generating or assigning pedigree ids (default "id").
- `string = obj.GetPedigreeIdArrayName ()` - The name of the array for generating or assigning pedigree ids (default "id").
- `obj.SetGeneratePedigreeIds (bool )` - If on (default), generates pedigree ids automatically. If off, assign one of the arrays to be the pedigree id.
- `bool = obj.GetGeneratePedigreeIds ()` - If on (default), generates pedigree ids automatically. If off, assign one of the arrays to be the pedigree id.
- `obj.GeneratePedigreeIdsOn ()` - If on (default), generates pedigree ids automatically. If off, assign one of the arrays to be the pedigree id.
- `obj.GeneratePedigreeIdsOff ()` - If on (default), generates pedigree ids automatically. If off, assign one of the arrays to be the pedigree id.
- `obj.SetOutputPedigreeIds (bool )` - If on, assigns pedigree ids to output. Defaults to off.
- `bool = obj.GetOutputPedigreeIds ()` - If on, assigns pedigree ids to output. Defaults to off.
- `obj.OutputPedigreeIdsOn ()` - If on, assigns pedigree ids to output. Defaults to off.
- `obj.OutputPedigreeIdsOff ()` - If on, assigns pedigree ids to output. Defaults to off.
- `vtkStdString = obj.GetLastError ()` - Returns a human-readable description of the most recent error, if any. Otherwise, returns an empty string. Note that the result is only valid after calling Update().

36.30 vtkDescriptiveStatistics

36.30.1 Usage

Given a selection of columns of interest in an input data table, this class provides the following functionalities, depending on the execution mode it is executed in: * Learn: calculate extremal values, arithmetic mean, unbiased variance estimator, skewness estimator, and both sample and G2 estimation of the kurtosis excess. More precisely, Learn calculates the sums; if finalize is set to true (default), the final statistics are calculated with CalculateFromSums. Otherwise, only raw sums are output; this option is made for efficient parallel calculations. Note that CalculateFromSums is a static function, so that it can be used directly with no need to instantiate a vtkDescriptiveStatistics object. * Assess: given an input data set in port INPUT_DATA, and a reference value x along with an acceptable deviation d¿0, assess all entries in the data set which are outside of [x-d,x+d].

.SECTION Thanks Thanks to Philippe Pebay and David Thompson from Sandia National Laboratories for implementing this class.

To create an instance of class vtkDescriptiveStatistics, simply invoke its constructor as follows

```
obj = vtkDescriptiveStatistics
```

36.30.2 Methods

The class vtkDescriptiveStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDescriptiveStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDescriptiveStatistics = obj.NewInstance ()`
- `vtkDescriptiveStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetUnbiasedVariance (int )` - Set/get whether the unbiased estimator for the variance should be used, or if the population variance will be calculated. The default is that the unbiased estimator will be used.
- `int = obj.GetUnbiasedVariance ()` - Set/get whether the unbiased estimator for the variance should be used, or if the population variance will be calculated. The default is that the unbiased estimator will be used.
- `obj.UnbiasedVarianceOn ()` - Set/get whether the unbiased estimator for the variance should be used, or if the population variance will be calculated. The default is that the unbiased estimator will be used.
- `obj.UnbiasedVarianceOff ()` - Set/get whether the unbiased estimator for the variance should be used, or if the population variance will be calculated. The default is that the unbiased estimator will be used.
- `obj.SetSignedDeviations (int )` - Set/get whether the deviations returned should be signed, or should only have their magnitude reported. The default is that signed deviations will be computed.
- `int = obj.GetSignedDeviations ()` - Set/get whether the deviations returned should be signed, or should only have their magnitude reported. The default is that signed deviations will be computed.
- `obj.SignedDeviationsOn ()` - Set/get whether the deviations returned should be signed, or should only have their magnitude reported. The default is that signed deviations will be computed.
- `obj.SignedDeviationsOff ()` - Set/get whether the deviations returned should be signed, or should only have their magnitude reported. The default is that signed deviations will be computed.
- `obj.SetNominalParameter (string name)` - A convenience method (in particular for UI wrapping) to set the name of the column that contains the nominal value for the Assess option.
- `obj.SetDeviationParameter (string name)` - A convenience method (in particular for UI wrapping) to set the name of the column that contains the deviation for the Assess option.
- `obj.Aggregate (vtkDataObjectCollection , vtkDataObject )` - Given a collection of models, calculate aggregate model

36.31 vtkDotProductSimilarity

36.31.1 Usage

Treats matrices as collections of vectors and computes dot-product similarity metrics between vectors.

The results are returned as an edge-table that lists the index of each vector and their computed similarity. The output edge-table is typically used with vtkTableToGraph to create a similarity graph.

This filter can be used with one or two input matrices. If you provide a single matrix as input, every vector in the matrix is compared with every other vector. If you provide two matrices, every vector in the first matrix is compared with every vector in the second matrix.

Note that this filter *only* computes the dot-product between each pair of vectors; if you want to compute the cosine of the angles between vectors, you will need to normalize the inputs yourself.

Inputs: Input port 0: (required) A vtkDenseArray¡double¿ with two dimensions (a matrix). Input port 1: (optional) A vtkDenseArray¡double¿ with two dimensions (a matrix).

Outputs: Output port 0: A vtkTable containing "source", "target", and "similarity" columns.

To create an instance of class vtkDotProductSimilarity, simply invoke its constructor as follows

```
obj = vtkDotProductSimilarity
```

36.31.2 Methods

The class vtkDotProductSimilarity has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDotProductSimilarity class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDotProductSimilarity = obj.NewInstance ()`
- `vtkDotProductSimilarity = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetVectorDimension ()` - Controls whether to compute similarities for row-vectors or column-vectors. 0 = rows, 1 = columns.
- `obj.SetVectorDimension (vtkIdType )` - Controls whether to compute similarities for row-vectors or column-vectors. 0 = rows, 1 = columns.
- `int = obj.GetUpperDiagonal ()` - When computing similarities for a single input matrix, controls whether the results will include the upper diagonal of the similarity matrix. Default: true.
- `obj.SetUpperDiagonal (int )` - When computing similarities for a single input matrix, controls whether the results will include the upper diagonal of the similarity matrix. Default: true.
- `int = obj.GetDiagonal ()` - When computing similarities for a single input matrix, controls whether the results will include the diagonal of the similarity matrix. Default: false.
- `obj.SetDiagonal (int )` - When computing similarities for a single input matrix, controls whether the results will include the diagonal of the similarity matrix. Default: false.
- `int = obj.GetLowerDiagonal ()` - When computing similarities for a single input matrix, controls whether the results will include the lower diagonal of the similarity matrix. Default: false.
- `obj.SetLowerDiagonal (int )` - When computing similarities for a single input matrix, controls whether the results will include the lower diagonal of the similarity matrix. Default: false.
- `int = obj.GetFirstSecond ()` - When computing similarities for two input matrices, controls whether the results will include comparisons from the first matrix to the second matrix.
- `obj.SetFirstSecond (int )` - When computing similarities for two input matrices, controls whether the results will include comparisons from the first matrix to the second matrix.
- `int = obj.GetSecondFirst ()` - When computing similarities for two input matrices, controls whether the results will include comparisons from the second matrix to the first matrix.

- `obj.SetSecondFirst (int )` - When computing similarities for two input matrices, controls whether the results will include comparisons from the second matrix to the first matrix.
- `double = obj.GetMinimumThreshold ()` - Specifies a minimum threshold that a similarity must exceed to be included in the output.
- `obj.SetMinimumThreshold (double )` - Specifies a minimum threshold that a similarity must exceed to be included in the output.
- `vtkIdType = obj.GetMinimumCount ()` - Specifies a minimum number of edges to include for each vector.
- `obj.SetMinimumCount (vtkIdType )` - Specifies a minimum number of edges to include for each vector.
- `vtkIdType = obj.GetMaximumCount ()` - Specifies a maximum number of edges to include for each vector.
- `obj.SetMaximumCount (vtkIdType )` - Specifies a maximum number of edges to include for each vector.

36.32 vtkEdgeCenters

36.32.1 Usage

vtkEdgeCenters is a filter that takes as input any graph and generates on output points at the center of the cells in the dataset. These points can be used for placing glyphs (vtkGlyph3D) or labeling (vtkLabeledDataMapper). (The center is the parametric center of the cell, not necessarily the geometric or bounding box center.) The edge attributes will be associated with the points on output.

To create an instance of class vtkEdgeCenters, simply invoke its constructor as follows

```
obj = vtkEdgeCenters
```

36.32.2 Methods

The class vtkEdgeCenters has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEdgeCenters class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEdgeCenters = obj.NewInstance ()`
- `vtkEdgeCenters = obj.SafeDownCast (vtkObject o)`
- `obj.SetVertexCells (int )` - Enable/disable the generation of vertex cells.
- `int = obj.GetVertexCells ()` - Enable/disable the generation of vertex cells.
- `obj.VertexCellsOn ()` - Enable/disable the generation of vertex cells.
- `obj.VertexCellsOff ()` - Enable/disable the generation of vertex cells.

36.33 vtkEdgeLayout

36.33.1 Usage

This class is a shell for many edge layout strategies which may be set using the SetLayoutStrategy() function. The layout strategies do the actual work.

To create an instance of class vtkEdgeLayout, simply invoke its constructor as follows

```
obj = vtkEdgeLayout
```

36.33.2 Methods

The class vtkEdgeLayout has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEdgeLayout class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEdgeLayout = obj.NewInstance ()`
- `vtkEdgeLayout = obj.SafeDownCast (vtkObject o)`
- `obj.SetLayoutStrategy (vtkEdgeLayoutStrategy strategy)` - The layout strategy to use during graph layout.
- `vtkEdgeLayoutStrategy = obj.GetLayoutStrategy ()` - The layout strategy to use during graph layout.
- `long = obj.GetMTime ()` - Get the modification time of the layout algorithm.

36.34 vtkEdgeLayoutStrategy

36.34.1 Usage

All edge layouts should subclass from this class. vtkEdgeLayoutStrategy works as a plug-in to the vtkEdgeLayout algorithm.

To create an instance of class vtkEdgeLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkEdgeLayoutStrategy
```

36.34.2 Methods

The class vtkEdgeLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEdgeLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEdgeLayoutStrategy = obj.NewInstance ()`
- `vtkEdgeLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetGraph (vtkGraph graph)` - Setting the graph for the layout strategy

- `obj.Initialize ()` - This method allows the layout strategy to do initialization of data structures or whatever else it might want to do.
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out.
- `obj.SetEdgeWeightArrayName (string )` - Set/Get the field to use for the edge weights.
- `string = obj.GetEdgeWeightArrayName ()` - Set/Get the field to use for the edge weights.

36.35 vtkExpandSelectedGraph

36.35.1 Usage

The first input is a vtkSelection containing the selected vertices. The second input is a vtkGraph. This filter 'grows' the selection set in one of the following ways 1) SetBFSDistance controls how many 'hops' the selection is grown from each seed point in the selection set (defaults to 1) 2) IncludeShortestPaths controls whether this filter tries to 'connect' the vertices in the selection set by computing the shortest path between the vertices (if such a path exists) Note: IncludeShortestPaths is currently non-functional

To create an instance of class vtkExpandSelectedGraph, simply invoke its constructor as follows

```
obj = vtkExpandSelectedGraph
```

36.35.2 Methods

The class vtkExpandSelectedGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExpandSelectedGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExpandSelectedGraph = obj.NewInstance ()`
- `vtkExpandSelectedGraph = obj.SafeDownCast (vtkObject o)`
- `obj.SetGraphConnection (vtkAlgorithmOutput in)` - A convenience method for setting the second input (i.e. the graph).
- `int = obj.FillInputPortInformation (int port, vtkInformation info)` - Specify the first vtkSelection input and the second vtkGraph input.
- `obj.SetBFSDistance (int )` - Set/Get BFSDistance which controls how many 'hops' the selection is grown from each seed point in the selection set (defaults to 1)
- `int = obj.GetBFSDistance ()` - Set/Get BFSDistance which controls how many 'hops' the selection is grown from each seed point in the selection set (defaults to 1)
- `obj.SetIncludeShortestPaths (bool )` - Set/Get IncludeShortestPaths controls whether this filter tries to 'connect' the vertices in the selection set by computing the shortest path between the vertices (if such a path exists) Note: IncludeShortestPaths is currently non-functional
- `bool = obj.GetIncludeShortestPaths ()` - Set/Get IncludeShortestPaths controls whether this filter tries to 'connect' the vertices in the selection set by computing the shortest path between the vertices (if such a path exists) Note: IncludeShortestPaths is currently non-functional
- `obj.IncludeShortestPathsOn ()` - Set/Get IncludeShortestPaths controls whether this filter tries to 'connect' the vertices in the selection set by computing the shortest path between the vertices (if such a path exists) Note: IncludeShortestPaths is currently non-functional

- `obj.IncludeShortestPathsOff ()` - Set/Get IncludeShortestPaths controls whether this filter tries to 'connect' the vertices in the selection set by computing the shortest path between the vertices (if such a path exists) Note: IncludeShortestPaths is currently non-functional
- `obj.SetDomain (string )` - Set/Get the vertex domain to use in the expansion.
- `string = obj.GetDomain ()` - Set/Get the vertex domain to use in the expansion.
- `obj.SetUseDomain (bool )` - Whether or not to use the domain when deciding to add a vertex to the expansion. Defaults to false.
- `bool = obj.GetUseDomain ()` - Whether or not to use the domain when deciding to add a vertex to the expansion. Defaults to false.
- `obj.UseDomainOn ()` - Whether or not to use the domain when deciding to add a vertex to the expansion. Defaults to false.
- `obj.UseDomainOff ()` - Whether or not to use the domain when deciding to add a vertex to the expansion. Defaults to false.

36.36 vtkExtractHistogram2D

36.36.1 Usage

This class computes a 2D histogram between two columns of an input vtkTable. Just as with a 1D histogram, a 2D histogram breaks up the input domain into bins, and each pair of values (row in the table) fits into a single bin and increments a row counter for that bin.

To use this class, set the input with a table and call AddColumnPair(nameX,nameY), where nameX and nameY are the names of the two columns to be used.

In addition to the number of bins (in X and Y), the domain of the histogram can be customized by toggling the UseCustomHistogramExtents flag and setting the CustomHistogramExtents variable to the desired value.

To create an instance of class vtkExtractHistogram2D, simply invoke its constructor as follows

```
obj = vtkExtractHistogram2D
```

36.36.2 Methods

The class vtkExtractHistogram2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractHistogram2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractHistogram2D = obj.NewInstance ()`
- `vtkExtractHistogram2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfBins (int , int )` - Set/get the number of bins to be used per dimension (x,y)
- `obj.SetNumberOfBins (int  a[2])` - Set/get the number of bins to be used per dimension (x,y)
- `int = obj. GetNumberOfBins ()` - Set/get the number of bins to be used per dimension (x,y)
- `obj.SetComponentsToProcess (int , int )` - Set/get the components of the arrays in the two input columns to be used during histogram computation. Defaults to component 0.

- `obj.SetComponentsToProcess (int a[2])` - Set/get the components of the arrays in the two input columns to be used during histogram computation. Defaults to component 0.
- `int = obj. GetComponentsToProcess ()` - Set/get the components of the arrays in the two input columns to be used during histogram computation. Defaults to component 0.
- `obj.SetCustomHistogramExtents (double , double , double , double )` - Set/get a custom domain for histogram computation. UseCustomHistogramExtents must be called for these to actually be used.
- `obj.SetCustomHistogramExtents (double a[4])` - Set/get a custom domain for histogram computation. UseCustomHistogramExtents must be called for these to actually be used.
- `double = obj. GetCustomHistogramExtents ()` - Set/get a custom domain for histogram computation. UseCustomHistogramExtents must be called for these to actually be used.
- `obj.SetUseCustomHistogramExtents (int )` - Use the extents in CustomHistogramExtents when computing the histogram, rather than the simple range of the input columns.
- `int = obj.GetUseCustomHistogramExtents ()` - Use the extents in CustomHistogramExtents when computing the histogram, rather than the simple range of the input columns.
- `obj.UseCustomHistogramExtentsOn ()` - Use the extents in CustomHistogramExtents when computing the histogram, rather than the simple range of the input columns.
- `obj.UseCustomHistogramExtentsOff ()` - Use the extents in CustomHistogramExtents when computing the histogram, rather than the simple range of the input columns.
- `obj.SetScalarType (int )` - Control the scalar type of the output histogram. If the input is relatively small, you can save space by using a smaller data type. Defaults to unsigned integer.
- `obj.SetScalarTypeToUnsignedInt ()` - Control the scalar type of the output histogram. If the input is relatively small, you can save space by using a smaller data type. Defaults to unsigned integer.
- `obj.SetScalarTypeToUnsignedLong ()` - Control the scalar type of the output histogram. If the input is relatively small, you can save space by using a smaller data type. Defaults to unsigned integer.
- `obj.SetScalarTypeToUnsignedShort ()` - Control the scalar type of the output histogram. If the input is relatively small, you can save space by using a smaller data type. Defaults to unsigned integer.
- `obj.SetScalarTypeToUnsignedChar ()` - Control the scalar type of the output histogram. If the input is relatively small, you can save space by using a smaller data type. Defaults to unsigned integer.
- `obj.SetScalarTypeToFloat ()` - Control the scalar type of the output histogram. If the input is relatively small, you can save space by using a smaller data type. Defaults to unsigned integer.
- `obj.SetScalarTypeToDouble ()` - Control the scalar type of the output histogram. If the input is relatively small, you can save space by using a smaller data type. Defaults to unsigned integer.
- `int = obj.GetScalarType ()` - Control the scalar type of the output histogram. If the input is relatively small, you can save space by using a smaller data type. Defaults to unsigned integer.
- `double = obj.GetMaximumBinCount ()` - Access the count of the histogram bin containing the largest number of input rows.
- `int = obj.GetBinRange (vtkIdType binX, vtkIdType binY, double range[4])` - Compute the range of the bin located at position (binX,binY) in the 2D histogram.
- `int = obj.GetBinRange (vtkIdType bin, double range[4])` - Get the range of the of the bin located at 1D position index bin in the 2D histogram array.

- `obj.GetBinWidth (double bw[2])` - Get the width of all of the bins. Also stored in the spacing ivar of the histogram image output.
- `vtkImageData = obj.GetOutputHistogramImage ()` - Gets the data object at the histogram image output port and casts it to a vtkImageData.
- `obj.SetSwapColumns (int )`
- `int = obj.GetSwapColumns ()`
- `obj.SwapColumnsOn ()`
- `obj.SwapColumnsOff ()`
- `obj.SetRowMask (vtkDataArray )` - Get/Set an optional mask that can ignore rows of the table
- `vtkDataArray = obj.GetRowMask ()` - Get/Set an optional mask that can ignore rows of the table
- `obj.Aggregate (vtkDataObjectCollection , vtkDataObject )` - Given a collection of models, calculate aggregate model. Not used.

36.37 vtkExtractSelectedGraph

36.37.1 Usage

The first input is a vtkGraph to take a subgraph from. The second input (optional) is a vtkSelection containing selected indices. The third input (optional) is a vtkAnnotationsLayers whose annotations contain selected specifying selected indices. The vtkSelection may have FIELD_TYPE set to POINTS (a vertex selection) or CELLS (an edge selection). A vertex selection preserves all edges that connect selected vertices. An edge selection preserves all vertices that are adjacent to at least one selected edge. Alternately, you may indicate that an edge selection should maintain the full set of vertices, by turning RemoveIsolatedVertices off.

To create an instance of class vtkExtractSelectedGraph, simply invoke its constructor as follows

```
obj = vtkExtractSelectedGraph
```

36.37.2 Methods

The class vtkExtractSelectedGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractSelectedGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractSelectedGraph = obj.NewInstance ()`
- `vtkExtractSelectedGraph = obj.SafeDownCast (vtkObject o)`
- `obj.SetSelectionConnection (vtkAlgorithmOutput in)` - A convenience method for setting the second input (i.e. the selection).
- `obj.SetAnnotationLayersConnection (vtkAlgorithmOutput in)` - A convenience method for setting the third input (i.e. the annotation layers).
- `obj.SetRemoveIsolatedVertices (bool )` - If set, removes vertices with no adjacent edges in an edge selection. A vertex selection ignores this flag and always returns the full set of selected vertices. Default is on.

- `bool = obj.GetRemoveIsolatedVertices ()` - If set, removes vertices with no adjacent edges in an edge selection. A vertex selection ignores this flag and always returns the full set of selected vertices. Default is on.
- `obj.RemoveIsolatedVerticesOn ()` - If set, removes vertices with no adjacent edges in an edge selection. A vertex selection ignores this flag and always returns the full set of selected vertices. Default is on.
- `obj.RemoveIsolatedVerticesOff ()` - If set, removes vertices with no adjacent edges in an edge selection. A vertex selection ignores this flag and always returns the full set of selected vertices. Default is on.
- `int = obj.FillInputPortInformation (int port, vtkInformation info)` - Specify the first vtkGraph input and the second vtkSelection input.

36.38 vtkFast2DLayoutStrategy

36.38.1 Usage

This class is a density grid based force directed layout strategy. Also please note that 'fast' is relative to quite slow. :) The layout running time is O(V+E) with an extremely high constant. .SECTION Thanks Thanks to Godzilla for not eating my computer so that this class could be written.

To create an instance of class vtkFast2DLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkFast2DLayoutStrategy
```

36.38.2 Methods

The class vtkFast2DLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFast2DLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFast2DLayoutStrategy = obj.NewInstance ()`
- `vtkFast2DLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetRandomSeed (int )` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMinValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMaxValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeed ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `obj.SetMaxNumberOfIterations (int )` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterationsMinValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterationsMaxValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterations ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `obj.SetIterationsPerLayout (int )` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayoutMinValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayoutMaxValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayout ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `obj.SetInitialTemperature (float )` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperatureMinValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperatureMaxValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperature ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `obj.SetCoolDownRate (double )` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)

- `double = obj.GetCoolDownRateMinValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)

- `double = obj.GetCoolDownRateMaxValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRate ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `obj.SetRestDistance (float )` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `float = obj.GetRestDistance ()` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `obj.Initialize ()` - This strategy sets up some data structures for faster processing of each Layout() call
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out. The method can either entirely layout the graph or iteratively lay out the graph. If you have an iterative layout please implement the IsLayoutComplete() method.
- `int = obj.IsLayoutComplete ()`

36.39 vtkFixedWidthTextReader

36.39.1 Usage

vtkFixedWidthTextReader reads in a table from a text file where each column occupies a certain number of characters.

This class emits ProgressEvent for every 100 lines it reads.

To create an instance of class vtkFixedWidthTextReader, simply invoke its constructor as follows

```
obj = vtkFixedWidthTextReader
```

36.39.2 Methods

The class vtkFixedWidthTextReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFixedWidthTextReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFixedWidthTextReader = obj.NewInstance ()`
- `vtkFixedWidthTextReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileName ()`
- `obj.SetFileName (string )`
- `obj.SetFieldWidth (int )` - Set/get the field width
- `int = obj.GetFieldWidth ()` - Set/get the field width

- `obj.SetStripWhiteSpace (bool )` - If set, this flag will cause the reader to strip whitespace from the beginning and ending of each field. Defaults to off.
- `bool = obj.GetStripWhiteSpace ()` - If set, this flag will cause the reader to strip whitespace from the beginning and ending of each field. Defaults to off.
- `obj.StripWhiteSpaceOn ()` - If set, this flag will cause the reader to strip whitespace from the beginning and ending of each field. Defaults to off.
- `obj.StripWhiteSpaceOff ()` - If set, this flag will cause the reader to strip whitespace from the beginning and ending of each field. Defaults to off.
- `bool = obj.GetHaveHeaders ()` - Set/get whether to treat the first line of the file as headers.
- `obj.SetHaveHeaders (bool )` - Set/get whether to treat the first line of the file as headers.
- `obj.HaveHeadersOn ()` - Set/get whether to treat the first line of the file as headers.
- `obj.HaveHeadersOff ()` - Set/get whether to treat the first line of the file as headers.

36.40 vtkForceDirectedLayoutStrategy

36.40.1 Usage

Lays out a graph in 2D or 3D using a force-directed algorithm. The user may specify whether to layout the graph randomly initially, the bounds, the number of dimensions (2 or 3), and the cool-down rate.

.SECTION Thanks Thanks to Brian Wylie for adding functionality for allowing this layout to be incremental.

To create an instance of class vtkForceDirectedLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkForceDirectedLayoutStrategy
```

36.40.2 Methods

The class vtkForceDirectedLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkForceDirectedLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkForceDirectedLayoutStrategy = obj.NewInstance ()`
- `vtkForceDirectedLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetRandomSeed (int )` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMinValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMaxValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeed ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.

- `obj.SetGraphBounds (double , double , double , double , double , double )` - Set / get the region in space in which to place the final graph. The GraphBounds only affects the results if AutomaticBoundsComputation is off.
- `obj.SetGraphBounds (double a[6])` - Set / get the region in space in which to place the final graph. The GraphBounds only affects the results if AutomaticBoundsComputation is off.
- `double = obj. GetGraphBounds ()` - Set / get the region in space in which to place the final graph. The GraphBounds only affects the results if AutomaticBoundsComputation is off.
- `obj.SetAutomaticBoundsComputation (int )` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `int = obj.GetAutomaticBoundsComputation ()` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `obj.AutomaticBoundsComputationOn ()` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `obj.AutomaticBoundsComputationOff ()` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `obj.SetMaxNumberOfIterations (int )` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '50' for no particular reason
- `int = obj.GetMaxNumberOfIterationsMinValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '50' for no particular reason
- `int = obj.GetMaxNumberOfIterationsMaxValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '50' for no particular reason
- `int = obj.GetMaxNumberOfIterations ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '50' for no particular reason
- `obj.SetIterationsPerLayout (int )` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '50' to match the default 'MaxNumberOfIterations'
- `int = obj.GetIterationsPerLayoutMinValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '50' to match the default 'MaxNumberOfIterations'
- `int = obj.GetIterationsPerLayoutMaxValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '50' to match the default 'MaxNumberOfIterations'
- `int = obj.GetIterationsPerLayout ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '50' to match the default 'MaxNumberOfIterations'
- `obj.SetCoolDownRate (double )` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified.

- `double = obj.GetCoolDownRateMinValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified.

- `double = obj.GetCoolDownRateMaxValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified.

- `double = obj.GetCoolDownRate ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified.

- `obj.SetThreeDimensionalLayout (int )` - Turn on/off layout of graph in three dimensions. If off, graph layout occurs in two dimensions. By default, three dimensional layout is off.

- `int = obj.GetThreeDimensionalLayout ()` - Turn on/off layout of graph in three dimensions. If off, graph layout occurs in two dimensions. By default, three dimensional layout is off.

- `obj.ThreeDimensionalLayoutOn ()` - Turn on/off layout of graph in three dimensions. If off, graph layout occurs in two dimensions. By default, three dimensional layout is off.

- `obj.ThreeDimensionalLayoutOff ()` - Turn on/off layout of graph in three dimensions. If off, graph layout occurs in two dimensions. By default, three dimensional layout is off.

- `obj.SetRandomInitialPoints (int )` - Turn on/off use of random positions within the graph bounds as initial points.

- `int = obj.GetRandomInitialPoints ()` - Turn on/off use of random positions within the graph bounds as initial points.

- `obj.RandomInitialPointsOn ()` - Turn on/off use of random positions within the graph bounds as initial points.

- `obj.RandomInitialPointsOff ()` - Turn on/off use of random positions within the graph bounds as initial points.

- `obj.SetInitialTemperature (float )` - Set the initial temperature. If zero (the default) , the initial temperature will be computed automatically.

- `float = obj.GetInitialTemperatureMinValue ()` - Set the initial temperature. If zero (the default) , the initial temperature will be computed automatically.

- `float = obj.GetInitialTemperatureMaxValue ()` - Set the initial temperature. If zero (the default) , the initial temperature will be computed automatically.

- `float = obj.GetInitialTemperature ()` - Set the initial temperature. If zero (the default) , the initial temperature will be computed automatically.

- `obj.Initialize ()` - This strategy sets up some data structures for faster processing of each Layout() call

- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out. The method can either entirely layout the graph or iteratively lay out the graph. If you have an iterative layout please implement the IsLayoutComplete() method.

- `int = obj.IsLayoutComplete ()`

36.41 vtkGenerateIndexArray

36.41.1 Usage

Generates a new vtkIdTypeArray containing zero-base indices.

vtkGenerateIndexArray operates in one of two distinct "modes". By default, it simply generates an index array containing monotonically-increasing integers in the range [0, N), where N is appropriately sized for the field type that will store the results. This mode is useful for generating a unique ID field for datasets that have none.

The second "mode" uses an existing array from the input data object as a "reference". Distinct values from the reference array are sorted in ascending order, and an integer index in the range [0, N) is assigned to each. The resulting map is used to populate the output index array, mapping each value in the reference array to its corresponding index and storing the result in the output array. This mode is especially useful when generating tensors, since it allows us to "map" from an array with arbitrary contents to an index that can be used as tensor coordinates.

To create an instance of class vtkGenerateIndexArray, simply invoke its constructor as follows

```
obj = vtkGenerateIndexArray
```

36.41.2 Methods

The class vtkGenerateIndexArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenerateIndexArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenerateIndexArray = obj.NewInstance ()`
- `vtkGenerateIndexArray = obj.SafeDownCast (vtkObject o)`
- `obj.SetArrayName (string )` - Control the output index array name. Default: "index".
- `string = obj.GetArrayName ()` - Control the output index array name. Default: "index".
- `obj.SetFieldType (int )` - Control the location where the index array will be stored.
- `int = obj.GetFieldType ()` - Control the location where the index array will be stored.
- `obj.SetReferenceArrayName (string )` - Specifies an optional reference array for index-generation.
- `string = obj.GetReferenceArrayName ()` - Specifies an optional reference array for index-generation.
- `obj.SetPedigreeID (int )` - Specifies whether the index array should be marked as pedigree ids. Default: false.
- `int = obj.GetPedigreeID ()` - Specifies whether the index array should be marked as pedigree ids. Default: false.

36.42 vtkGeoEdgeStrategy

36.42.1 Usage

vtkGeoEdgeStrategy produces arcs for each edge in the input graph. This is useful for viewing lines on a sphere (e.g. the earth). The arcs may "jump" above the sphere's surface using ExplodeFactor.

To create an instance of class vtkGeoEdgeStrategy, simply invoke its constructor as follows

```
obj = vtkGeoEdgeStrategy
```

36.42.2 Methods

The class vtkGeoEdgeStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoEdgeStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoEdgeStrategy = obj.NewInstance ()`
- `vtkGeoEdgeStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetGlobeRadius (double )` - The base radius used to determine the earth's surface. Default is the earth's radius in meters. TODO: Change this to take in a vtkGeoTerrain to get altitude.
- `double = obj.GetGlobeRadius ()` - The base radius used to determine the earth's surface. Default is the earth's radius in meters. TODO: Change this to take in a vtkGeoTerrain to get altitude.
- `obj.SetExplodeFactor (double )` - Factor on which to "explode" the arcs away from the surface. A value of 0.0 keeps the values on the surface. Values larger than 0.0 push the arcs away from the surface by a distance proportional to the distance between the points. The default is 0.2.
- `double = obj.GetExplodeFactor ()` - Factor on which to "explode" the arcs away from the surface. A value of 0.0 keeps the values on the surface. Values larger than 0.0 push the arcs away from the surface by a distance proportional to the distance between the points. The default is 0.2.
- `obj.SetNumberOfSubdivisions (int )` - The number of subdivisions in the arc. The default is 20.
- `int = obj.GetNumberOfSubdivisions ()` - The number of subdivisions in the arc. The default is 20.
- `obj.Layout ()` - Perform the layout.

36.43 vtkGeoMath

36.43.1 Usage

vtkGeoMath provides some useful geographic calculations.

To create an instance of class vtkGeoMath, simply invoke its constructor as follows

```
obj = vtkGeoMath
```

36.43.2 Methods

The class vtkGeoMath has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGeoMath class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGeoMath = obj.NewInstance ()`
- `vtkGeoMath = obj.SafeDownCast (vtkObject o)`

36.44 vtkGraphHierarchicalBundle

36.44.1 Usage

This algorithm creates a vtkPolyData from a vtkGraph. As opposed to vtkGraphToPolyData, which converts each arc into a straight line, each arc is converted to a polyline, following a tree structure. The filter requires both a vtkGraph and vtkTree as input. The tree vertices must be a superset of the graph vertices. A common example is when the graph vertices correspond to the leaves of the tree, but the internal vertices of the tree represent groupings of graph vertices. The algorithm matches the vertices using the array "PedigreeId". The user may alternately set the DirectMapping flag to indicate that the two structures must have directly corresponding offsets (i.e. node i in the graph must correspond to node i in the tree).

The vtkGraph defines the topology of the output vtkPolyData (i.e. the connections between nodes) while the vtkTree defines the geometry (i.e. the location of nodes and arc routes). Thus, the tree must have been assigned vertex locations, but the graph does not need locations, in fact they will be ignored. The edges approximately follow the path from the source to target nodes in the tree. A bundling parameter controls how closely the edges are bundled together along the tree structure.

You may follow this algorithm with vtkSplineFilter in order to make nicely curved edges.

To create an instance of class vtkGraphHierarchicalBundle, simply invoke its constructor as follows

```
obj = vtkGraphHierarchicalBundle
```

36.44.2 Methods

The class vtkGraphHierarchicalBundle has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphHierarchicalBundle class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphHierarchicalBundle = obj.NewInstance ()`
- `vtkGraphHierarchicalBundle = obj.SafeDownCast (vtkObject o)`
- `obj.SetBundlingStrength (double )` - The level of arc bundling in the graph. A strength of 0 creates straight lines, while a strength of 1 forces arcs to pass directly through hierarchy node points. The default value is 0.8.
- `double = obj.GetBundlingStrengthMinValue ()` - The level of arc bundling in the graph. A strength of 0 creates straight lines, while a strength of 1 forces arcs to pass directly through hierarchy node points. The default value is 0.8.
- `double = obj.GetBundlingStrengthMaxValue ()` - The level of arc bundling in the graph. A strength of 0 creates straight lines, while a strength of 1 forces arcs to pass directly through hierarchy node points. The default value is 0.8.
- `double = obj.GetBundlingStrength ()` - The level of arc bundling in the graph. A strength of 0 creates straight lines, while a strength of 1 forces arcs to pass directly through hierarchy node points. The default value is 0.8.
- `obj.SetDirectMapping (bool )` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `bool = obj.GetDirectMapping ()` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.

- `obj.DirectMappingOn ()` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `obj.DirectMappingOff ()` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `int = obj.FillInputPortInformation (int port, vtkInformation info)` - Set the input type of the algorithm to vtkGraph.

36.45 vtkGraphHierarchicalBundleEdges

36.45.1 Usage

This algorithm creates a vtkPolyData from a vtkGraph. As opposed to vtkGraphToPolyData, which converts each arc into a straight line, each arc is converted to a polyline, following a tree structure. The filter requires both a vtkGraph and vtkTree as input. The tree vertices must be a superset of the graph vertices. A common example is when the graph vertices correspond to the leaves of the tree, but the internal vertices of the tree represent groupings of graph vertices. The algorithm matches the vertices using the array "PedigreeId". The user may alternately set the DirectMapping flag to indicate that the two structures must have directly corresponding offsets (i.e. node i in the graph must correspond to node i in the tree).

The vtkGraph defines the topology of the output vtkPolyData (i.e. the connections between nodes) while the vtkTree defines the geometry (i.e. the location of nodes and arc routes). Thus, the tree must have been assigned vertex locations, but the graph does not need locations, in fact they will be ignored. The edges approximately follow the path from the source to target nodes in the tree. A bundling parameter controls how closely the edges are bundled together along the tree structure.

You may follow this algorithm with vtkSplineFilter in order to make nicely curved edges.

To create an instance of class vtkGraphHierarchicalBundleEdges, simply invoke its constructor as follows

```
obj = vtkGraphHierarchicalBundleEdges
```

36.45.2 Methods

The class vtkGraphHierarchicalBundleEdges has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphHierarchicalBundleEdges class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphHierarchicalBundleEdges = obj.NewInstance ()`
- `vtkGraphHierarchicalBundleEdges = obj.SafeDownCast (vtkObject o)`
- `obj.SetBundlingStrength (double )` - The level of arc bundling in the graph. A strength of 0 creates straight lines, while a strength of 1 forces arcs to pass directly through hierarchy node points. The default value is 0.8.
- `double = obj.GetBundlingStrengthMinValue ()` - The level of arc bundling in the graph. A strength of 0 creates straight lines, while a strength of 1 forces arcs to pass directly through hierarchy node points. The default value is 0.8.
- `double = obj.GetBundlingStrengthMaxValue ()` - The level of arc bundling in the graph. A strength of 0 creates straight lines, while a strength of 1 forces arcs to pass directly through hierarchy node points. The default value is 0.8.

- `double = obj.GetBundlingStrength ()` - The level of arc bundling in the graph. A strength of 0 creates straight lines, while a strength of 1 forces arcs to pass directly through hierarchy node points. The default value is 0.8.
- `obj.SetDirectMapping (bool )` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `bool = obj.GetDirectMapping ()` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `obj.DirectMappingOn ()` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `obj.DirectMappingOff ()` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `int = obj.FillInputPortInformation (int port, vtkInformation info)` - Set the input type of the algorithm to vtkGraph.

36.46 vtkGraphLayout

36.46.1 Usage

This class is a shell for many graph layout strategies which may be set using the SetLayoutStrategy() function. The layout strategies do the actual work.

.SECION Thanks Thanks to Brian Wylie from Sandia National Laboratories for adding incremental layout capabilities.

To create an instance of class vtkGraphLayout, simply invoke its constructor as follows

```
obj = vtkGraphLayout
```

36.46.2 Methods

The class vtkGraphLayout has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphLayout class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphLayout = obj.NewInstance ()`
- `vtkGraphLayout = obj.SafeDownCast (vtkObject o)`
- `obj.SetLayoutStrategy (vtkGraphLayoutStrategy strategy)` - The layout strategy to use during graph layout.
- `vtkGraphLayoutStrategy = obj.GetLayoutStrategy ()` - The layout strategy to use during graph layout.
- `int = obj.IsLayoutComplete ()` - Ask the layout algorithm if the layout is complete
- `long = obj.GetMTime ()` - Get the modification time of the layout algorithm.

- `double = obj.GetZRange ()` - Set the ZRange for the output data. If the initial layout is planar (i.e. all z coordinates are zero), the coordinates will be evenly spaced from 0.0 to ZRange. The default is zero, which has no effect.
- `obj.SetZRange (double )` - Set the ZRange for the output data. If the initial layout is planar (i.e. all z coordinates are zero), the coordinates will be evenly spaced from 0.0 to ZRange. The default is zero, which has no effect.
- `vtkAbstractTransform = obj.GetTransform ()` - Transform the graph vertices after the layout.
- `obj.SetTransform (vtkAbstractTransform t)` - Transform the graph vertices after the layout.
- `obj.SetUseTransform (bool )` - Whether to use the specified transform after layout.
- `bool = obj.GetUseTransform ()` - Whether to use the specified transform after layout.
- `obj.UseTransformOn ()` - Whether to use the specified transform after layout.
- `obj.UseTransformOff ()` - Whether to use the specified transform after layout.

36.47 vtkGraphLayoutStrategy

36.47.1 Usage

All graph layouts should subclass from this class. vtkGraphLayoutStrategy works as a plug-in to the vtkGraphLayout algorithm. The Layout() function should perform some reasonable "chunk" of the layout. This allows the user to be able to see the progress of the layout. Use IsLayoutComplete() to tell the user when there is no more layout to perform.

.SECTION Thanks Thanks to Brian Wylie from Sandia National Laboratories for adding incremental layout capabilities.

To create an instance of class vtkGraphLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkGraphLayoutStrategy
```

36.47.2 Methods

The class vtkGraphLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphLayoutStrategy = obj.NewInstance ()`
- `vtkGraphLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetGraph (vtkGraph graph)` - Setting the graph for the layout strategy
- `obj.Initialize ()` - This method allows the layout strategy to do initialization of data structures or whatever else it might want to do.
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out. The method can either entirely layout the graph or iteratively lay out the graph. If you have an iterative layout please implement the IsLayoutComplete() method.
- `int = obj.IsLayoutComplete ()` - Whether to use edge weights in the layout or not.

- `obj.SetWeightEdges (bool state)` - Whether to use edge weights in the layout or not.
- `bool = obj.GetWeightEdges ()` - Whether to use edge weights in the layout or not.
- `obj.SetEdgeWeightField (string field)` - Set/Get the field to use for the edge weights.
- `string = obj.GetEdgeWeightField ()` - Set/Get the field to use for the edge weights.

36.48 vtkGroupLeafVertices

36.48.1 Usage

Use SetInputArrayToProcess(0, ...) to set the array to group on. Currently this array must be a vtkStringArray.

To create an instance of class vtkGroupLeafVertices, simply invoke its constructor as follows

```
obj = vtkGroupLeafVertices
```

36.48.2 Methods

The class vtkGroupLeafVertices has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGroupLeafVertices class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGroupLeafVertices = obj.NewInstance ()`
- `vtkGroupLeafVertices = obj.SafeDownCast (vtkObject o)`
- `obj.SetGroupDomain (string )` - The name of the domain that non-leaf vertices will be assigned to. If the input graph already contains vertices in this domain: - If the ids for this domain are numeric, starts assignment with max id - If the ids for this domain are strings, starts assignment with "group X" where "X" is the max id. Default is "group_vertex".
- `string = obj.GetGroupDomain ()` - The name of the domain that non-leaf vertices will be assigned to. If the input graph already contains vertices in this domain: - If the ids for this domain are numeric, starts assignment with max id - If the ids for this domain are strings, starts assignment with "group X" where "X" is the max id. Default is "group_vertex".

36.49 vtkISIReader

36.49.1 Usage

ISI is a tagged format for expressing bibliographic citations. Data is structured as a collection of records with each record composed of one-to-many fields. See

http://isibasic.com/help/helpprn.html#dialog_export_format

for details. vtkISIReader will convert an ISI file into a vtkTable, with the set of table columns determined dynamically from the contents of the file.

To create an instance of class vtkISIReader, simply invoke its constructor as follows

```
obj = vtkISIReader
```

36.49.2 Methods

The class vtkISIReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkISIReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkISIReader = obj.NewInstance ()`
- `vtkISIReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileName ()` - Set/get the file to load
- `obj.SetFileName (string )` - Set/get the file to load
- `string = obj.GetDelimiter ()` - Set/get the delimiter to be used for concatenating field data (default: ";")
- `obj.SetDelimiter (string )` - Set/get the delimiter to be used for concatenating field data (default: ";")
- `int = obj.GetMaxRecords ()` - Set/get the maximum number of records to read from the file (zero = unlimited)
- `obj.SetMaxRecords (int )` - Set/get the maximum number of records to read from the file (zero = unlimited)

36.50 vtkKMeansDistanceFunctor

36.50.1 Usage

This is an abstract class (with a default concrete subclass) that implements algorithms used by the vtkKMeansStatistics filter that rely on a distance metric. If you wish to use a non-Euclidean distance metric (this could include working with strings that do not have a Euclidean distance metric, implementing k-mediods, or trying distance metrics in norms other than L2), you should subclass vtkKMeansDistanceFunctor.

To create an instance of class vtkKMeansDistanceFunctor, simply invoke its constructor as follows

```
obj = vtkKMeansDistanceFunctor
```

36.50.2 Methods

The class vtkKMeansDistanceFunctor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkKMeansDistanceFunctor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkKMeansDistanceFunctor = obj.NewInstance ()`
- `vtkKMeansDistanceFunctor = obj.SafeDownCast (vtkObject o)`
- `vtkVariantArray = obj.GetEmptyTuple (vtkIdType dimension)` - Return an empty tuple. These values are used as cluster center coordinates when no initial cluster centers are specified.

- `obj.PairwiseUpdate (vtkTable clusterCenters, vtkIdType row, vtkVariantArray data, vtkIdType dataCa` - This is called once per observation per run per iteration in order to assign the observation to its nearest cluster center after the distance functor has been evaluated for all the cluster centers.

  The distance functor is responsible for incrementally updating the cluster centers to account for the assignment.

- `obj.PerturbElement (vtkTable , vtkTable , vtkIdType , vtkIdType , vtkIdType , double )` - When a cluster center (1) has no observations that are closer to it than other cluster centers or (2) has exactly the same coordinates as another cluster center, its coordinates should be perturbed. This function should perform that perturbation.

  Since perturbation relies on a distance metric, this function is the responsibility of the distance functor.

- `vtkAbstractArray = obj.CreateCoordinateArray ()` - Return a vtkAbstractArray capable of holding cluster center coordinates. This is used by vtkPKMeansStatistics to hold cluster center coordinates sent to (received from) other processes.

- `int = obj.GetDataType ()` - Return the data type used to store cluster center coordinates.

36.51 vtkKMeansDistanceFunctorCalculator

36.51.1 Usage

This is a subclass of the default k-means distance functor that allows the user to specify a distance function as a string. The provided expression is evaluated whenever the parenthesis operator is invoked but this is much slower than the default distance calculation.

User-specified distance expressions should be written in terms of two vector variables named "x" and "y". The length of the vectors will be determined by the k-means request and all columns of interest in the request must contain values that may be converted to a floating point representation. (Strings and vtkObject pointers are not allowed.) An example distance expression is "sqrt((x0-y0)$\hat{}$2 + (x1-y1)$\hat{}$2)" which computes Euclidian distance in a plane defined by the first 2 coordinates of the vectors specified.

To create an instance of class vtkKMeansDistanceFunctorCalculator, simply invoke its constructor as follows

```
obj = vtkKMeansDistanceFunctorCalculator
```

36.51.2 Methods

The class vtkKMeansDistanceFunctorCalculator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkKMeansDistanceFunctorCalculator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkKMeansDistanceFunctorCalculator = obj.NewInstance ()`
- `vtkKMeansDistanceFunctorCalculator = obj.SafeDownCast (vtkObject o)`
- `obj.SetDistanceExpression (string )` - Set/get the distance function expression.
- `string = obj.GetDistanceExpression ()` - Set/get the distance function expression.
- `obj.SetFunctionParser (vtkFunctionParser )` - Set/get the string containing an expression which evaluates to the distance metric used for k-means computation. The scalar variables "x0", "x1", ... "xn" and "y0", "y1", ..., "yn" refer to the coordinates involved in the computation.

- `vtkFunctionParser = obj.GetFunctionParser ()` - Set/get the string containing an expression which evaluates to the distance metric used for k-means computation. The scalar variables "x0", "x1", ... "xn" and "y0", "y1", ..., "yn" refer to the coordinates involved in the computation.

36.52 vtkKMeansStatistics

36.52.1 Usage

This class takes as input an optional vtkTable on port LEARN_PARAMETERS specifying initial set(s) of cluster values of the following form:

```
     K     | Col1             |  ...    | ColN
-----------+------------------+---------+---------------
     M     |clustCoord(1, 1) |  ...    | clustCoord(1, N)
     M     |clustCoord(2, 1) |  ...    | clustCoord(2, N)
     .     |       .          |   .     |        .
     .     |       .          |   .     |        .
     .     |       .          |   .     |        .
     M     |clustCoord(M, 1) |  ...    | clustCoord(M, N)
     L     |clustCoord(1, 1) |  ...    | clustCoord(1, N)
     L     |clustCoord(2, 1) |  ...    | clustCoord(2, N)
     .     |       .          |   .     |        .
     .     |       .          |   .     |        .
     .     |       .          |   .     |        .
     L     |clustCoord(L, 1) |  ...    | clustCoord(L, N)
```

Because the desired value of K is often not known in advance and the results of the algorithm are dependent on the initial cluster centers, we provide a mechanism for the user to test multiple runs or sets of cluster centers within a single call to the Learn phase. The first column of the table identifies the number of clusters K in the particular run (the entries in this column should be of type vtkIdType), while the remaining columns are a subset of the columns contained in the table on port INPUT_DATA. We require that all user specified clusters be of the same dimension N and consequently, that the LEARN_PARAMETERS table have N+1 columns. Due to this restriction, only one request can be processed for each call to the Learn phase and subsequent requests are silently ignored. Note that, if the first column of the LEARN_PARAMETERS table is not of type vtkIdType, then the table will be ignored and a single run will be performed using the first DefaultNumberOfClusters input data observations as initial cluster centers.

When the user does not supply an initial set of clusters, then the first DefaultNumberOfClusters input data observations are used as initial cluster centers and a single run is performed.

This class provides the following functionalities, depending on the mode it is executed in: * Learn: calculates new cluster centers for each run. The output metadata on port OUTPUT_MODEL is a multiblock dataset containing at a minimum one vtkTable with columns specifying the following for each run: the run ID, number of clusters, number of iterations required for convergence, total error associated with the cluster (sum of squared Euclidean distance from each observation to its nearest cluster center), the cardinality of the cluster, and the new cluster coordinates.

*Derive: An additional vtkTable is stored in the multiblock dataset output on port OUTPUT_MODEL. This table contains columns that store for each run: the runID, number of clusters, total error for all clusters in the run, local rank, and global rank. The local rank is computed by comparing squared Euclidean errors of all runs with the same number of clusters. The global rank is computed analagously across all runs.

* Assess: This requires a multiblock dataset (as computed from Learn and Derive) on input port INPUT_MODEL and tabular data on input port INPUT_DATA that contains column names matching those of the tables on input port INPUT_MODEL. The assess mode reports the closest cluster center and associated squared Euclidean distance of each observation in port INPUT_DATA's table to the cluster centers for each run in the multiblock dataset provided on port INPUT_MODEL.

The code can handle a wide variety of data types as it operates on vtkAbstractArrays and is not limited to vtkDataArrays. A default distance functor that computes the sum of the squares of the Euclidean distance between two objects is provided (vtkKMeansDistanceFunctor). The default distance functor can be overridden to use alternative distance metrics.

.SECTION Thanks Thanks to Janine Bennett, David Thompson, and Philippe Pebay of Sandia National Laboratories for implementing this class.

To create an instance of class vtkKMeansStatistics, simply invoke its constructor as follows

```
obj = vtkKMeansStatistics
```

36.52.2 Methods

The class vtkKMeansStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkKMeansStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkKMeansStatistics = obj.NewInstance ()`
- `vtkKMeansStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetDistanceFunctor (vtkKMeansDistanceFunctor )` - Set the DistanceFunctor.
- `vtkKMeansDistanceFunctor = obj.GetDistanceFunctor ()` - Set the DistanceFunctor.
- `obj.SetDefaultNumberOfClusters (int )` - Set/get the DefaultNumberOfClusters, used when no initial cluster coordinates are specified.
- `int = obj.GetDefaultNumberOfClusters ()` - Set/get the DefaultNumberOfClusters, used when no initial cluster coordinates are specified.
- `obj.SetKValuesArrayName (string )` - Set/get the KValuesArrayName.
- `string = obj.GetKValuesArrayName ()` - Set/get the KValuesArrayName.
- `obj.SetMaxNumIterations (int )` - Set/get the MaxNumIterations used to terminate iterations on cluster center coordinates when the relative tolerance can not be met.
- `int = obj.GetMaxNumIterations ()` - Set/get the MaxNumIterations used to terminate iterations on cluster center coordinates when the relative tolerance can not be met.
- `obj.SetTolerance (double )` - Set/get the relative Tolerance used to terminate iterations on cluster center coordinates.
- `double = obj.GetTolerance ()` - Set/get the relative Tolerance used to terminate iterations on cluster center coordinates.
- `obj.Aggregate (vtkDataObjectCollection , vtkDataObject )` - Given a collection of models, calculate aggregate model NB: not implemented

36.53 vtkMatricizeArray

36.53.1 Usage

Given a sparse input array of arbitrary dimension, creates a sparse output matrix (vtkSparseArray¡double¿) where each column is a slice along an arbitrary dimension from the source.

.SECTION Thanks Developed by Timothy M. Shead (tshead@sandia.gov) at Sandia National Laboratories.

To create an instance of class vtkMatricizeArray, simply invoke its constructor as follows

```
obj = vtkMatricizeArray
```

36.53.2 Methods

The class vtkMatricizeArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMatricizeArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMatricizeArray = obj.NewInstance ()`
- `vtkMatricizeArray = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetSliceDimension ()` - Returns the 0-numbered dimension that will be mapped to columns in the output
- `obj.SetSliceDimension (vtkIdType )` - Sets the 0-numbered dimension that will be mapped to columns in the output

36.54 vtkMergeColumns

36.54.1 Usage

vtkMergeColumns replaces two columns in a table with a single column containing data in both columns. The columns are set using

SetInputArrayToProcess(0, 0, 0, vtkDataObject::FIELD_ASSOCIATION_ROWS, "col1")

and

SetInputArrayToProcess(1, 0, 0, vtkDataObject::FIELD_ASSOCIATION_ROWS, "col2")

where "col1" and "col2" are the names of the columns to merge. The user may also specify the name of the merged column. The arrays must be of the same type. If the arrays are numeric, the values are summed in the merged column. If the arrays are strings, the values are concatenated. The strings are separated by a space if they are both nonempty.

To create an instance of class vtkMergeColumns, simply invoke its constructor as follows

```
obj = vtkMergeColumns
```

36.54.2 Methods

The class vtkMergeColumns has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMergeColumns class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkMergeColumns = obj.NewInstance ()`
- `vtkMergeColumns = obj.SafeDownCast (vtkObject o)`
- `obj.SetMergedColumnName (string )` - The name to give the merged column created by this filter.
- `string = obj.GetMergedColumnName ()` - The name to give the merged column created by this filter.

36.55 vtkMergeGraphs

36.55.1 Usage

vtkMergeGraphs combines information from two graphs into one. Both graphs must have pedigree ids assigned to the vertices. The output will contain the vertices/edges in the first graph, in addition to:
- vertices in the second graph whose pedigree id does not match a vertex in the first input
- edges in the second graph

The output will contain the same attribute structure as the input; fields associated only with the second input graph will not be passed to the output. When possible, the vertex/edge data for new vertices and edges will be populated with matching attributes on the second graph. To be considered a matching attribute, the array must have the same name, type, and number of components.

To create an instance of class vtkMergeGraphs, simply invoke its constructor as follows

```
obj = vtkMergeGraphs
```

36.55.2 Methods

The class vtkMergeGraphs has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMergeGraphs class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMergeGraphs = obj.NewInstance ()`
- `vtkMergeGraphs = obj.SafeDownCast (vtkObject o)`
- `int = obj.ExtendGraph (vtkMutableGraphHelper g1, vtkGraph g2)` - This is the core functionality of the algorithm. Adds edges and vertices from g2 into g1.
- `obj.SetMaxEdges (vtkIdType )` - The maximum number of edges in the combined graph. Default is -1, which specifies that there should be no limit on the number of edges.
- `vtkIdType = obj.GetMaxEdges ()` - The maximum number of edges in the combined graph. Default is -1, which specifies that there should be no limit on the number of edges.

36.56 vtkMergeTables

36.56.1 Usage

Combines the columns of two tables into one larger table. The number of rows in the resulting table is the sum of the number of rows in each of the input tables. The number of columns in the output is generally the sum of the number of columns in each input table, except in the case where column names are duplicated in both tables. In this case, if MergeColumnsByName is on (the default), the two columns will be merged into

a single column of the same name. If MergeColumnsByName is off, both columns will exist in the output. You may set the FirstTablePrefix and SecondTablePrefix to define how the columns named are modified. One of these prefixes may be the empty string, but they must be different.

To create an instance of class vtkMergeTables, simply invoke its constructor as follows

```
obj = vtkMergeTables
```

36.56.2 Methods

The class vtkMergeTables has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMergeTables class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMergeTables = obj.NewInstance ()`
- `vtkMergeTables = obj.SafeDownCast (vtkObject o)`
- `obj.SetFirstTablePrefix (string )` - The prefix to give to same-named fields from the first table. Default is "Table1.".
- `string = obj.GetFirstTablePrefix ()` - The prefix to give to same-named fields from the first table. Default is "Table1.".
- `obj.SetSecondTablePrefix (string )` - The prefix to give to same-named fields from the second table. Default is "Table2.".
- `string = obj.GetSecondTablePrefix ()` - The prefix to give to same-named fields from the second table. Default is "Table2.".
- `obj.SetMergeColumnsByName (bool )` - If on, merges columns with the same name. If off, keeps both columns, but calls one FirstTablePrefix + name, and the other SecondTablePrefix + name. Default is on.
- `bool = obj.GetMergeColumnsByName ()` - If on, merges columns with the same name. If off, keeps both columns, but calls one FirstTablePrefix + name, and the other SecondTablePrefix + name. Default is on.
- `obj.MergeColumnsByNameOn ()` - If on, merges columns with the same name. If off, keeps both columns, but calls one FirstTablePrefix + name, and the other SecondTablePrefix + name. Default is on.
- `obj.MergeColumnsByNameOff ()` - If on, merges columns with the same name. If off, keeps both columns, but calls one FirstTablePrefix + name, and the other SecondTablePrefix + name. Default is on.
- `obj.SetPrefixAllButMerged (bool )` - If on, all columns will have prefixes except merged columns. If off, only unmerged columns with the same name will have prefixes. Default is off.
- `bool = obj.GetPrefixAllButMerged ()` - If on, all columns will have prefixes except merged columns. If off, only unmerged columns with the same name will have prefixes. Default is off.
- `obj.PrefixAllButMergedOn ()` - If on, all columns will have prefixes except merged columns. If off, only unmerged columns with the same name will have prefixes. Default is off.
- `obj.PrefixAllButMergedOff ()` - If on, all columns will have prefixes except merged columns. If off, only unmerged columns with the same name will have prefixes. Default is off.

36.57 vtkMultiCorrelativeStatistics

36.57.1 Usage

Given a selection of sets of columns of interest, this class provides the following functionalities, depending on the execution mode it is executed in: * Learn: calculates means, unbiased variance and covariance estimators of column pairs coefficient. More precisely, Learn calculates the averages and centered variance/covariance sums; if finalize is set to true (default), the final statistics are calculated. The output metadata on port OUTPUT_MODEL is a multiblock dataset containing at a minimum one vtkTable holding the raw sums in a sparse matrix style. If finalize is true, then one additional vtkTable will be present for each requested set of column correlations. These additional tables contain column averages, the upper triangular portion of the covariance matrix (in the upper right hand portion of the table) and the Cholesky decomposition of the covariance matrix (in the lower portion of the table beneath the covariance triangle). The leftmost column will be a vector of column averages. The last entry in the column averages vector is the number of samples. As an example, consider a request for a 3-column correlation with columns named ColA, ColB, and ColC. The resulting table will look like this:

```
Column  |Mean     |ColA     |ColB     |ColC
--------+---------+---------+---------+---------
ColA    |avg(A)   |cov(A,A) |cov(A,B) |cov(A,C)
ColB    |avg(B)   |chol(1,1)|cov(B,B) |cov(B,C)
ColC    |avg(C)   |chol(2,1)|chol(2,2)|cov(C,C)
Cholesky|length(A)|chol(3,1)|chol(3,2)|chol(3,3)
```

* Assess: given a set of results matrices as specified above in input port INPUT_MODEL and tabular data on input port INPUT_DATA that contains column names matching those of the tables on input port INPUT_MODEL, the assess mode computes the relative deviation of each observation in port INPUT_DATA's table according to the linear correlations implied by each table in port INPUT_MODEL.

.SECTION Thanks Thanks to Philippe Pebay, Jackson Mayo, and David Thompson of Sandia National Laboratories for implementing this class.

To create an instance of class vtkMultiCorrelativeStatistics, simply invoke its constructor as follows

```
obj = vtkMultiCorrelativeStatistics
```

36.57.2 Methods

The class vtkMultiCorrelativeStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiCorrelativeStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiCorrelativeStatistics = obj.NewInstance ()`
- `vtkMultiCorrelativeStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.Aggregate (vtkDataObjectCollection , vtkDataObject )` - Given a collection of models, calculate aggregate model

36.58 vtkMutableGraphHelper

36.58.1 Usage

vtkMutableGraphHelper has helper methods AddVertex and AddEdge which add vertices/edges to the underlying mutable graph. This is helpful in filters which need to (re)construct graphs which may be either directed or undirected.

To create an instance of class vtkMutableGraphHelper, simply invoke its constructor as follows

```
obj = vtkMutableGraphHelper
```

36.58.2 Methods

The class vtkMutableGraphHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMutableGraphHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMutableGraphHelper = obj.NewInstance ()`
- `vtkMutableGraphHelper = obj.SafeDownCast (vtkObject o)`
- `obj.SetGraph (vtkGraph g)` - Set the underlying graph that you want to modify with this helper. The graph must be an instance of vtkMutableDirectedGraph or vtkMutableUndirectedGraph.
- `vtkGraph = obj.GetGraph ()` - Set the underlying graph that you want to modify with this helper. The graph must be an instance of vtkMutableDirectedGraph or vtkMutableUndirectedGraph.
- `vtkGraphEdge = obj.AddGraphEdge (vtkIdType u, vtkIdType v)` - Add an edge to the underlying mutable graph.
- `vtkIdType = obj.AddVertex ()` - Add a vertex to the underlying mutable graph.
- `obj.RemoveVertex (vtkIdType v)` - Remove a vertex from the underlying mutable graph.
- `obj.RemoveVertices (vtkIdTypeArray verts)` - Remove a collection of vertices from the underlying mutable graph.
- `obj.RemoveEdge (vtkIdType e)` - Remove an edge from the underlying mutable graph.
- `obj.RemoveEdges (vtkIdTypeArray edges)` - Remove a collection of edges from the underlying mutable graph.

36.59 vtkNetworkHierarchy

36.59.1 Usage

Use SetInputArrayToProcess(0, ...) to set the array to that has the network ip addresses. Currently this array must be a vtkStringArray.

To create an instance of class vtkNetworkHierarchy, simply invoke its constructor as follows

```
obj = vtkNetworkHierarchy
```

36.59.2 Methods

The class vtkNetworkHierarchy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkNetworkHierarchy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkNetworkHierarchy = obj.NewInstance ()`
- `vtkNetworkHierarchy = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetIPArrayName ()` - Used to store the ip array name
- `obj.SetIPArrayName (string )` - Used to store the ip array name

36.60 vtkOrderStatistics

36.60.1 Usage

Given a selection of columns of interest in an input data table, this class provides the following functionalities, depending on the execution mode it is executed in: * Learn: calculate 5-point statistics (minimum, 1st quartile, median, third quartile, maximum) and all other deciles (1,2,3,4,6,7,8,9). * Assess: given an input data set in port INPUT_DATA, and two percentiles p1 ¡ p2, assess all entries in the data set which are outside of [p1,p2].

.SECTION Thanks Thanks to Philippe Pebay and David Thompson from Sandia National Laboratories for implementing this class.

To create an instance of class vtkOrderStatistics, simply invoke its constructor as follows

```
obj = vtkOrderStatistics
```

36.60.2 Methods

The class vtkOrderStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOrderStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOrderStatistics = obj.NewInstance ()`
- `vtkOrderStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfIntervals (vtkIdType )` - Set the number of quantiles (with uniform spacing).
- `vtkIdType = obj.GetNumberOfIntervals ()` - Get the number of quantiles (with uniform spacing).
- `obj.SetQuantileDefinition (int )` - Set the quantile definition.
- `vtkIdType = obj.GetQuantileDefinition ()` - Given a collection of models, calculate aggregate model NB: not implemented
- `obj.Aggregate (vtkDataObjectCollection , vtkDataObject )` - Given a collection of models, calculate aggregate model NB: not implemented

36.61 vtkPairwiseExtractHistogram2D

36.61.1 Usage

This class computes a 2D histogram between all adjacent pairs of columns of an input vtkTable. Internally it creates multiple vtkExtractHistogram2D instances (one for each pair of adjacent table columns). It also manages updating histogram computations intelligently, only recomputing those histograms for whom a relevant property has been altered.

Note that there are two different outputs from this filter. One is a table for which each column contains a flattened 2D histogram array. The other is a vtkMultiBlockDataSet for which each block is a vtkImageData representation of the 2D histogram.

To create an instance of class vtkPairwiseExtractHistogram2D, simply invoke its constructor as follows

```
obj = vtkPairwiseExtractHistogram2D
```

36.61.2 Methods

The class vtkPairwiseExtractHistogram2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPairwiseExtractHistogram2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPairwiseExtractHistogram2D = obj.NewInstance ()`
- `vtkPairwiseExtractHistogram2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfBins (int , int )` - Set/get the bin dimensions of the histograms to compute
- `obj.SetNumberOfBins (int  a[2])` - Set/get the bin dimensions of the histograms to compute
- `int = obj. GetNumberOfBins ()` - Set/get the bin dimensions of the histograms to compute
- `obj.SetCustomColumnRangeIndex (int )` - Strange method for setting an index to be used for setting custom column range. This was (probably) necessary to get this class to interact with the ParaView client/server message passing interface.
- `obj.SetCustomColumnRangeByIndex (double , double )` - Strange method for setting an index to be used for setting custom column range. This was (probably) necessary to get this class to interact with the ParaView client/server message passing interface.
- `obj.SetCustomColumnRange (int col, double range[2])` - More standard way to set the custom range for a particular column. This makes sure that only the affected histograms know that they need to be updated.
- `obj.SetCustomColumnRange (int col, double rmin, double rmax)` - More standard way to set the custom range for a particular column. This makes sure that only the affected histograms know that they need to be updated.
- `obj.SetScalarType (int )` - Set the scalar type for each of the computed histograms.
- `obj.SetScalarTypeToUnsignedInt ()` - Set the scalar type for each of the computed histograms.
- `obj.SetScalarTypeToUnsignedLong ()` - Set the scalar type for each of the computed histograms.
- `obj.SetScalarTypeToUnsignedShort ()` - Set the scalar type for each of the computed histograms.
- `obj.SetScalarTypeToUnsignedChar ()` - Set the scalar type for each of the computed histograms.

- `int = obj.GetScalarType ()` - Set the scalar type for each of the computed histograms.
- `double = obj.GetMaximumBinCount (int idx)` - Get the maximum bin count for a single histogram
- `double = obj.GetMaximumBinCount ()` - Get the maximum bin count over all histograms
- `int = obj.GetBinRange (int idx, vtkIdType binX, vtkIdType binY, double range[4])` - Compute the range of the bin located at position (binX,binY) in the 2D histogram at idx.
- `int = obj.GetBinRange (int idx, vtkIdType bin, double range[4])` - Get the range of the of the bin located at 1D position index bin in the 2D histogram array at idx.
- `obj.GetBinWidth (int idx, double bw[2])` - Get the width of all of the bins. Also stored in the spacing ivar of the histogram image output at idx.
- `vtkImageData = obj.GetOutputHistogramImage (int idx)` - Get the vtkImageData output of the idx'th histogram filter
- `vtkExtractHistogram2D = obj.GetHistogramFilter (int idx)` - Get a pointer to the idx'th histogram filter.
- `obj.Aggregate (vtkDataObjectCollection , vtkDataObject )` - Given a collection of models, calculate aggregate model. Not used

36.62 vtkPassArrays

36.62.1 Usage

This filter preserves all the topology of the input, but only a subset of arrays are passed to the output. Add an array to be passed to the output data object with AddArray(). If RemoveArrays is on, the specified arrays will be the ones that are removed instead of the ones that are kept.

Arrays with special attributes (scalars, pedigree ids, etc.) will retain those attributes in the output.

By default, only those field types with at least one array specified through AddArray will be processed. If instead UseFieldTypes is turned on, you explicitly set which field types to process with AddFieldType.

Example 1:

```
passArray->AddArray(vtkDataObject::POINT, ''velocity'');
```

The output will have only that one array "velocity" in the point data, but cell and field data will be untouched.

Example 2:

```
passArray->AddArray(vtkDataObject::POINT, ''velocity'');
passArray->UseFieldTypesOn();
passArray->AddFieldType(vtkDataObject::POINT);
passArray->AddFieldType(vtkDataObject::CELL);
```

The point data would still contain the single array, but the cell data would be cleared since you did not specify any arrays to pass. Field data would still be untouched.

To create an instance of class vtkPassArrays, simply invoke its constructor as follows

```
obj = vtkPassArrays
```

36.62.2 Methods

The class vtkPassArrays has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPassArrays class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPassArrays = obj.NewInstance ()`
- `vtkPassArrays = obj.SafeDownCast (vtkObject o)`
- `obj.AddArray (int fieldType, string name)` - Adds an array to pass through. fieldType where the array that should be passed (point data, cell data, etc.). It should be one of the constants defined in the vtkDataObject::AttributeTypes enumeration.
- `obj.ClearArrays ()` - Clear all arrays to pass through.
- `obj.SetRemoveArrays (bool )` - Instead of passing only the specified arrays, remove the specified arrays and keep all other arrays. Default is off.
- `bool = obj.GetRemoveArrays ()` - Instead of passing only the specified arrays, remove the specified arrays and keep all other arrays. Default is off.
- `obj.RemoveArraysOn ()` - Instead of passing only the specified arrays, remove the specified arrays and keep all other arrays. Default is off.
- `obj.RemoveArraysOff ()` - Instead of passing only the specified arrays, remove the specified arrays and keep all other arrays. Default is off.
- `obj.SetUseFieldTypes (bool )` - Process only those field types explicitly specified with AddFieldType. Otherwise, processes field types associated with at least one specified array. Default is off.
- `bool = obj.GetUseFieldTypes ()` - Process only those field types explicitly specified with AddFieldType. Otherwise, processes field types associated with at least one specified array. Default is off.
- `obj.UseFieldTypesOn ()` - Process only those field types explicitly specified with AddFieldType. Otherwise, processes field types associated with at least one specified array. Default is off.
- `obj.UseFieldTypesOff ()` - Process only those field types explicitly specified with AddFieldType. Otherwise, processes field types associated with at least one specified array. Default is off.
- `obj.AddFieldType (int fieldType)` - Add a field type to process. fieldType where the array that should be passed (point data, cell data, etc.). It should be one of the constants defined in the vtkDataObject::AttributeTypes enumeration. NOTE: These are only used if UseFieldType is turned on.
- `obj.ClearFieldTypes ()` - Clear all field types to process.

36.63 vtkPassThrough

36.63.1 Usage

The output type is always the same as the input object type.

To create an instance of class vtkPassThrough, simply invoke its constructor as follows

```
obj = vtkPassThrough
```

36.63.2 Methods

The class vtkPassThrough has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPassThrough class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPassThrough = obj.NewInstance ()`
- `vtkPassThrough = obj.SafeDownCast (vtkObject o)`
- `int = obj.FillInputPortInformation (int port, vtkInformation info)` - Specify the first input port as optional
- `obj.SetDeepCopyInput (int )` - Whether or not to deep copy the input. This can be useful if you want to create a copy of a data object. You can then disconnect this filter's input connections and it will act like a source. Defaults to OFF.
- `int = obj.GetDeepCopyInput ()` - Whether or not to deep copy the input. This can be useful if you want to create a copy of a data object. You can then disconnect this filter's input connections and it will act like a source. Defaults to OFF.
- `obj.DeepCopyInputOn ()` - Whether or not to deep copy the input. This can be useful if you want to create a copy of a data object. You can then disconnect this filter's input connections and it will act like a source. Defaults to OFF.
- `obj.DeepCopyInputOff ()` - Whether or not to deep copy the input. This can be useful if you want to create a copy of a data object. You can then disconnect this filter's input connections and it will act like a source. Defaults to OFF.

36.64 vtkPassThroughEdgeStrategy

36.64.1 Usage

Simply passes existing edge layout information from the input to the output without making changes.

To create an instance of class vtkPassThroughEdgeStrategy, simply invoke its constructor as follows

```
obj = vtkPassThroughEdgeStrategy
```

36.64.2 Methods

The class vtkPassThroughEdgeStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPassThroughEdgeStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPassThroughEdgeStrategy = obj.NewInstance ()`
- `vtkPassThroughEdgeStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out.

36.65 vtkPassThroughLayoutStrategy

36.65.1 Usage

Yes, this incredible strategy does absoluted nothing to the data so in affect passes through the graph untouched. This strategy is useful in the cases where the graph is already laid out.

To create an instance of class vtkPassThroughLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkPassThroughLayoutStrategy
```

36.65.2 Methods

The class vtkPassThroughLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPassThroughLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPassThroughLayoutStrategy = obj.NewInstance ()`
- `vtkPassThroughLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - This strategy sets up some data structures for faster processing of each Layout() call
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out. The method can either entirely layout the graph or iteratively lay out the graph. If you have an iterative layout please implement the IsLayoutComplete() method.
- `int = obj.IsLayoutComplete ()`

36.66 vtkPBivariateLinearTableThreshold

36.66.1 Usage

Perform the table filtering operations provided by vtkBivariateLinearTableThreshold in parallel.

To create an instance of class vtkPBivariateLinearTableThreshold, simply invoke its constructor as follows

```
obj = vtkPBivariateLinearTableThreshold
```

36.66.2 Methods

The class vtkPBivariateLinearTableThreshold has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPBivariateLinearTableThreshold class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPBivariateLinearTableThreshold = obj.NewInstance ()`
- `vtkPBivariateLinearTableThreshold = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Set the vtkMultiProcessController to be used for combining filter results from the individual nodes.
- `vtkMultiProcessController = obj.GetController ()` - Set the vtkMultiProcessController to be used for combining filter results from the individual nodes.

36.67 vtkPCAStatistics

36.67.1 Usage

This class derives from the multi-correlative statistics algorithm and uses the covariance matrix and Cholesky decomposition computed by it. However, when it finalizes the statistics in Learn mode, the PCA class computes the SVD of the covariance matrix in order to obtain its eigenvectors.

In the assess mode, the input data are - projected into the basis defined by the eigenvectors, - the energy associated with each datum is computed, - or some combination thereof. Additionally, the user may specify some threshold energy or eigenvector entry below which the basis is truncated. This allows projection into a lower-dimensional state while minimizing (in a least squares sense) the projection error.

.SECTION Thanks Thanks to David Thompson, Philippe Pebay and Jackson Mayo from Sandia National Laboratories for implementing this class.

To create an instance of class vtkPCAStatistics, simply invoke its constructor as follows

```
obj = vtkPCAStatistics
```

36.67.2 Methods

The class vtkPCAStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPCAStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPCAStatistics = obj.NewInstance ()`
- `vtkPCAStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetNormalizationScheme (int )` - This determines how (or if) the covariance matrix cov is normalized before PCA.

  When set to NONE, no normalization is performed. This is the default.

  When set to TRIANGLE_SPECIFIED, each entry cov(i,j) is divided by V(i,j). The list V of normalization factors must be set using the SetNormalization method before the filter is executed.

  When set to DIAGONAL_SPECIFIED, each entry cov(i,j) is divided by sqrt(V(i)*V(j)). The list V of normalization factors must be set using the SetNormalization method before the filter is executed.

  When set to DIAGONAL_VARIANCE, each entry cov(i,j) is divided by sqrt(cov(i,i)*cov(j,j)). ¡b¿Warning¡/b¿: Although this is accepted practice in some fields, some people think you should not turn this option on unless there is a good physically-based reason for doing so. Much better instead to determine how component magnitudes should be compared using physical reasoning and use DIAGONAL_SPECIFIED, TRIANGLE_SPECIFIED, or perform some pre-processing to shift and scale input data columns appropriately than to expect magical results from a shady normalization hack.

- `int = obj.GetNormalizationScheme ()` - This determines how (or if) the covariance matrix cov is normalized before PCA.

  When set to NONE, no normalization is performed. This is the default.

  When set to TRIANGLE_SPECIFIED, each entry cov(i,j) is divided by V(i,j). The list V of normalization factors must be set using the SetNormalization method before the filter is executed.

  When set to DIAGONAL_SPECIFIED, each entry cov(i,j) is divided by sqrt(V(i)*V(j)). The list V of normalization factors must be set using the SetNormalization method before the filter is executed.

  When set to DIAGONAL_VARIANCE, each entry cov(i,j) is divided by sqrt(cov(i,i)*cov(j,j)). ¡b¿Warning¡/b¿: Although this is accepted practice in some fields, some people think you should not turn this option on

unless there is a good physically-based reason for doing so. Much better instead to determine how component magnitudes should be compared using physical reasoning and use DIAGONAL_SPECIFIED, TRIANGLE_SPECIFIED, or perform some pre-processing to shift and scale input data columns appropriately than to expect magical results from a shady normalization hack.

- `obj.SetNormalizationSchemeByName (string sname)` - This determines how (or if) the covariance matrix cov is normalized before PCA.

  When set to NONE, no normalization is performed. This is the default.

  When set to TRIANGLE_SPECIFIED, each entry cov(i,j) is divided by V(i,j). The list V of normalization factors must be set using the SetNormalization method before the filter is executed.

  When set to DIAGONAL_SPECIFIED, each entry cov(i,j) is divided by sqrt(V(i)*V(j)). The list V of normalization factors must be set using the SetNormalization method before the filter is executed.

  When set to DIAGONAL_VARIANCE, each entry cov(i,j) is divided by sqrt(cov(i,i)*cov(j,j)). ¡b¿Warning¡/b¿: Although this is accepted practice in some fields, some people think you should not turn this option on unless there is a good physically-based reason for doing so. Much better instead to determine how component magnitudes should be compared using physical reasoning and use DIAGONAL_SPECIFIED, TRIANGLE_SPECIFIED, or perform some pre-processing to shift and scale input data columns appropriately than to expect magical results from a shady normalization hack.

- `string = obj.GetNormalizationSchemeName (int scheme)` - This determines how (or if) the covariance matrix cov is normalized before PCA.

  When set to NONE, no normalization is performed. This is the default.

  When set to TRIANGLE_SPECIFIED, each entry cov(i,j) is divided by V(i,j). The list V of normalization factors must be set using the SetNormalization method before the filter is executed.

  When set to DIAGONAL_SPECIFIED, each entry cov(i,j) is divided by sqrt(V(i)*V(j)). The list V of normalization factors must be set using the SetNormalization method before the filter is executed.

  When set to DIAGONAL_VARIANCE, each entry cov(i,j) is divided by sqrt(cov(i,i)*cov(j,j)). ¡b¿Warning¡/b¿: Although this is accepted practice in some fields, some people think you should not turn this option on unless there is a good physically-based reason for doing so. Much better instead to determine how component magnitudes should be compared using physical reasoning and use DIAGONAL_SPECIFIED, TRIANGLE_SPECIFIED, or perform some pre-processing to shift and scale input data columns appropriately than to expect magical results from a shady normalization hack.

- `vtkTable = obj.GetSpecifiedNormalization ()` - These methods allow you to set/get values used to normalize the covariance matrix before PCA. The normalization values apply to all requests, so you do not specify a single vector but a 3-column table.

  The first two columns contain the names of columns from input 0 and the third column contains the value to normalize the corresponding entry in the covariance matrix. The table must always have 3 columns even when the NormalizationScheme is DIAGONAL_SPECIFIED. When only diagonal entries are to be used, only table rows where the first two columns are identical to one another will be employed. If there are multiple rows specifying different values for the same pair of columns, the entry nearest the bottom of the table takes precedence.

  These functions are actually convenience methods that set/get the third input of the filter. Because the table is the third input, you may use other filters to produce a table of normalizations and have the pipeline take care of updates.

  Any missing entries will be set to 1.0 and a warning issued. An error will occur if the third input to the filter is not set and the NormalizationScheme is DIAGONAL_SPECIFIED or TRIANGLE_SPECIFIED.

- `obj.SetSpecifiedNormalization (vtkTable )` - These methods allow you to set/get values used to normalize the covariance matrix before PCA. The normalization values apply to all requests, so you do not specify a single vector but a 3-column table.

The first two columns contain the names of columns from input 0 and the third column contains the value to normalize the corresponding entry in the covariance matrix. The table must always have 3 columns even when the NormalizationScheme is DIAGONAL_SPECIFIED. When only diagonal entries are to be used, only table rows where the first two columns are identical to one another will be employed. If there are multiple rows specifying different values for the same pair of columns, the entry nearest the bottom of the table takes precedence.

These functions are actually convenience methods that set/get the third input of the filter. Because the table is the third input, you may use other filters to produce a table of normalizations and have the pipeline take care of updates.

Any missing entries will be set to 1.0 and a warning issued. An error will occur if the third input to the filter is not set and the NormalizationScheme is DIAGONAL_SPECIFIED or TRIANGLE_SPECIFIED.

- `obj.SetBasisScheme (int )` - This variable controls the dimensionality of output tuples in Assess mode. Consider the case where you have requested a PCA on D columns.

  When set to vtkPCAStatistics::FULL_BASIS, the entire set of basis vectors is used to derive new coordinates for each tuple being assessed. In this mode, you are guaranteed to have output tuples of the same dimension as the input tuples. (That dimension is D, so there will be D additional columns added to the table for the request.)

  When set to vtkPCAStatistics::FIXED_BASIS_SIZE, only the first N basis vectors are used to derive new coordinates for each tuple being assessed. In this mode, you are guaranteed to have output tuples of dimension min(N,D). You must set N prior to assessing data using the SetFixedBasisSize() method. When N ¡ D, this turns the PCA into a projection (instead of change of basis).

  When set to vtkPCAStatistics::FIXED_BASIS_ENERGY, the number of basis vectors used to derive new coordinates for each tuple will be the minimum number of columns N that satisfy

  $$\frac{\sum_{i=1}^{N} \lambda_i}{\sum_{i=1}^{D} \lambda_i} < T$$

  You must set T prior to assessing data using the SetFixedBasisEnergy() method. When T ¡ 1, this turns the PCA into a projection (instead of change of basis).

  By default BasisScheme is set to vtkPCAStatistics::FULL_BASIS.

- `int = obj.GetBasisScheme ()` - This variable controls the dimensionality of output tuples in Assess mode. Consider the case where you have requested a PCA on D columns.

  When set to vtkPCAStatistics::FULL_BASIS, the entire set of basis vectors is used to derive new coordinates for each tuple being assessed. In this mode, you are guaranteed to have output tuples of the same dimension as the input tuples. (That dimension is D, so there will be D additional columns added to the table for the request.)

  When set to vtkPCAStatistics::FIXED_BASIS_SIZE, only the first N basis vectors are used to derive new coordinates for each tuple being assessed. In this mode, you are guaranteed to have output tuples of dimension min(N,D). You must set N prior to assessing data using the SetFixedBasisSize() method. When N ¡ D, this turns the PCA into a projection (instead of change of basis).

  When set to vtkPCAStatistics::FIXED_BASIS_ENERGY, the number of basis vectors used to derive new coordinates for each tuple will be the minimum number of columns N that satisfy

  $$\frac{\sum_{i=1}^{N} \lambda_i}{\sum_{i=1}^{D} \lambda_i} < T$$

  You must set T prior to assessing data using the SetFixedBasisEnergy() method. When T ¡ 1, this turns the PCA into a projection (instead of change of basis).

  By default BasisScheme is set to vtkPCAStatistics::FULL_BASIS.

- `string = obj.GetBasisSchemeName (int schemeIndex)` - This variable controls the dimensionality of output tuples in Assess mode. Consider the case where you have requested a PCA on D columns.

  When set to vtkPCAStatistics::FULL_BASIS, the entire set of basis vectors is used to derive new coordinates for each tuple being assessed. In this mode, you are guaranteed to have output tuples of the same dimension as the input tuples. (That dimension is D, so there will be D additional columns added to the table for the request.)

  When set to vtkPCAStatistics::FIXED_BASIS_SIZE, only the first N basis vectors are used to derive new coordinates for each tuple being assessed. In this mode, you are guaranteed to have output tuples of dimension min(N,D). You must set N prior to assessing data using the SetFixedBasisSize() method. When N ¡ D, this turns the PCA into a projection (instead of change of basis).

  When set to vtkPCAStatistics::FIXED_BASIS_ENERGY, the number of basis vectors used to derive new coordinates for each tuple will be the minimum number of columns N that satisfy

  $$\frac{\sum_{i=1}^{N} \lambda_i}{\sum_{i=1}^{D} \lambda_i} < T$$

  You must set T prior to assessing data using the SetFixedBasisEnergy() method. When T ¡ 1, this turns the PCA into a projection (instead of change of basis).

  By default BasisScheme is set to vtkPCAStatistics::FULL_BASIS.

- `obj.SetBasisSchemeByName (string schemeName)` - This variable controls the dimensionality of output tuples in Assess mode. Consider the case where you have requested a PCA on D columns.

  When set to vtkPCAStatistics::FULL_BASIS, the entire set of basis vectors is used to derive new coordinates for each tuple being assessed. In this mode, you are guaranteed to have output tuples of the same dimension as the input tuples. (That dimension is D, so there will be D additional columns added to the table for the request.)

  When set to vtkPCAStatistics::FIXED_BASIS_SIZE, only the first N basis vectors are used to derive new coordinates for each tuple being assessed. In this mode, you are guaranteed to have output tuples of dimension min(N,D). You must set N prior to assessing data using the SetFixedBasisSize() method. When N ¡ D, this turns the PCA into a projection (instead of change of basis).

  When set to vtkPCAStatistics::FIXED_BASIS_ENERGY, the number of basis vectors used to derive new coordinates for each tuple will be the minimum number of columns N that satisfy

  $$\frac{\sum_{i=1}^{N} \lambda_i}{\sum_{i=1}^{D} \lambda_i} < T$$

  You must set T prior to assessing data using the SetFixedBasisEnergy() method. When T ¡ 1, this turns the PCA into a projection (instead of change of basis).

  By default BasisScheme is set to vtkPCAStatistics::FULL_BASIS.

- `obj.SetFixedBasisSize (int )` - The number of basis vectors to use. See SetBasisScheme() for more information. When FixedBasisSize ¡= 0 (the default), the fixed basis size scheme is equivalent to the full basis scheme.

- `int = obj.GetFixedBasisSize ()` - The number of basis vectors to use. See SetBasisScheme() for more information. When FixedBasisSize ¡= 0 (the default), the fixed basis size scheme is equivalent to the full basis scheme.

- `obj.SetFixedBasisEnergy (double )` - The minimum energy the new basis should use, as a fraction. See SetBasisScheme() for more information. When FixedBasisEnergy ¿= 1 (the default), the fixed basis energy scheme is equivalent to the full basis scheme.

- `double = obj.GetFixedBasisEnergyMinValue ()` - The minimum energy the new basis should use, as a fraction. See SetBasisScheme() for more information. When FixedBasisEnergy ¿= 1 (the default), the fixed basis energy scheme is equivalent to the full basis scheme.

- `double = obj.GetFixedBasisEnergyMaxValue ()` - The minimum energy the new basis should use, as a fraction. See SetBasisScheme() for more information. When FixedBasisEnergy ¿= 1 (the default), the fixed basis energy scheme is equivalent to the full basis scheme.
- `double = obj.GetFixedBasisEnergy ()` - The minimum energy the new basis should use, as a fraction. See SetBasisScheme() for more information. When FixedBasisEnergy ¿= 1 (the default), the fixed basis energy scheme is equivalent to the full basis scheme.

36.68 vtkPComputeHistogram2DOutliers

36.68.1 Usage

This class does exactly the same this as vtkComputeHistogram2DOutliers, but does it in a multi-process environment. After each node computes their own local outliers, class does an AllGather that distributes the outliers to every node. This could probably just be a Gather onto the root node instead.

After this operation, the row selection will only contain local row ids, since I'm not sure how to deal with distributed ids.

To create an instance of class vtkPComputeHistogram2DOutliers, simply invoke its constructor as follows

```
obj = vtkPComputeHistogram2DOutliers
```

36.68.2 Methods

The class vtkPComputeHistogram2DOutliers has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPComputeHistogram2DOutliers class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPComputeHistogram2DOutliers = obj.NewInstance ()`
- `vtkPComputeHistogram2DOutliers = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )`
- `vtkMultiProcessController = obj.GetController ()`

36.69 vtkPContingencyStatistics

36.69.1 Usage

vtkPContingencyStatistics is vtkContingencyStatistics subclass for parallel datasets. It learns and derives the global statistical model on each node, but assesses each individual data points on the node that owns it.

To create an instance of class vtkPContingencyStatistics, simply invoke its constructor as follows

```
obj = vtkPContingencyStatistics
```

36.69.2 Methods

The class vtkPContingencyStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPContingencyStatistics class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkPContingencyStatistics = obj.NewInstance ()`
- `vtkPContingencyStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.
- `vtkMultiProcessController = obj.GetController ()` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.
- `obj.Learn (vtkTable inData, vtkTable inParameters, vtkDataObject outMeta)` - Execute the parallel calculations required by the Learn option.

36.70 vtkPCorrelativeStatistics

36.70.1 Usage

vtkPCorrelativeStatistics is vtkCorrelativeStatistics subclass for parallel datasets. It learns and derives the global statistical model on each node, but assesses each individual data points on the node that owns it.

To create an instance of class vtkPCorrelativeStatistics, simply invoke its constructor as follows

```
obj = vtkPCorrelativeStatistics
```

36.70.2 Methods

The class vtkPCorrelativeStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPCorrelativeStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPCorrelativeStatistics = obj.NewInstance ()`
- `vtkPCorrelativeStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.
- `vtkMultiProcessController = obj.GetController ()` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.
- `obj.Learn (vtkTable inData, vtkTable inParameters, vtkDataObject outMeta)` - Execute the parallel calculations required by the Learn option.

36.71 vtkPDescriptiveStatistics

36.71.1 Usage

vtkPDescriptiveStatistics is vtkDescriptiveStatistics subclass for parallel datasets. It learns and derives the global statistical model on each node, but assesses each individual data points on the node that owns it.

To create an instance of class vtkPDescriptiveStatistics, simply invoke its constructor as follows

```
obj = vtkPDescriptiveStatistics
```

36.71.2 Methods

The class vtkPDescriptiveStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPDescriptiveStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPDescriptiveStatistics = obj.NewInstance ()`
- `vtkPDescriptiveStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.
- `vtkMultiProcessController = obj.GetController ()` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.
- `obj.Learn (vtkTable inData, vtkTable inParameters, vtkDataObject outMeta)` - Execute the parallel calculations required by the Learn option.

36.72 vtkPerturbCoincidentVertices

36.72.1 Usage

This filter perturbs vertices in a graph that have coincident coordinates. In particular this happens all the time with graphs that are georeferenced, so we need a nice scheme to perturb the vertices so that when the user zooms in the vertices can be distiquished.

To create an instance of class vtkPerturbCoincidentVertices, simply invoke its constructor as follows

```
obj = vtkPerturbCoincidentVertices
```

36.72.2 Methods

The class vtkPerturbCoincidentVertices has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPerturbCoincidentVertices class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPerturbCoincidentVertices = obj.NewInstance ()`
- `vtkPerturbCoincidentVertices = obj.SafeDownCast (vtkObject o)`
- `obj.SetPerturbFactor (double )` - Specify the perturbation factor (defaults to 1.0)
- `double = obj.GetPerturbFactor ()` - Specify the perturbation factor (defaults to 1.0)

36.73 vtkPExtractHistogram2D

36.73.1 Usage

This class does exactly the same this as vtkExtractHistogram2D, but does it in a multi-process environment. After each node computes their own local histograms, this class does an AllReduce that distributes the sum of all local histograms onto each node.

To create an instance of class vtkPExtractHistogram2D, simply invoke its constructor as follows

```
obj = vtkPExtractHistogram2D
```

36.73.2 Methods

The class vtkPExtractHistogram2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPExtractHistogram2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPExtractHistogram2D = obj.NewInstance ()`
- `vtkPExtractHistogram2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )`
- `vtkMultiProcessController = obj.GetController ()`

36.74 vtkPKMeansStatistics

36.74.1 Usage

vtkPKMeansStatistics is vtkKMeansStatistics subclass for parallel datasets. It learns and derives the global statistical model on each node, but assesses each individual data points on the node that owns it.

To create an instance of class vtkPKMeansStatistics, simply invoke its constructor as follows

```
obj = vtkPKMeansStatistics
```

36.74.2 Methods

The class vtkPKMeansStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPKMeansStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPKMeansStatistics = obj.NewInstance ()`
- `vtkPKMeansStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.
- `vtkMultiProcessController = obj.GetController ()` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.

- `obj.UpdateClusterCenters (vtkTable newClusterElements, vtkTable curClusterElements, vtkIdTypeArray` - Subroutine to update new cluster centers from the old centers.
- `vtkIdType = obj.GetTotalNumberOfObservations (vtkIdType numObservations)` - Subroutine to get the total number of data objects.
- `obj.CreateInitialClusterCenters (vtkIdType numToAllocate, vtkIdTypeArray numberOfClusters, vtkTable` - Subroutine to initialize cluster centerss if not provided by the user.

36.75 vtkPMultiCorrelativeStatistics

36.75.1 Usage

vtkPMultiCorrelativeStatistics is vtkMultiCorrelativeStatistics subclass for parallel datasets. It learns and derives the global statistical model on each node, but assesses each individual data points on the node that owns it.

To create an instance of class vtkPMultiCorrelativeStatistics, simply invoke its constructor as follows

```
obj = vtkPMultiCorrelativeStatistics
```

36.75.2 Methods

The class vtkPMultiCorrelativeStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPMultiCorrelativeStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPMultiCorrelativeStatistics = obj.NewInstance ()`
- `vtkPMultiCorrelativeStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.
- `vtkMultiProcessController = obj.GetController ()` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.

36.76 vtkPPairwiseExtractHistogram2D

36.76.1 Usage

This class does exactly the same this as vtkPairwiseExtractHistogram2D, but does it in a multi-process environment. After each node computes their own local histograms, this class does an AllReduce that distributes the sum of all local histograms onto each node.

Because vtkPairwiseExtractHistogram2D is a light wrapper around a series of vtkExtractHistogram2D classes, this class just overrides the function that instantiates new histogram filters and returns the parallel version (vtkPExtractHistogram2D).

To create an instance of class vtkPPairwiseExtractHistogram2D, simply invoke its constructor as follows

```
obj = vtkPPairwiseExtractHistogram2D
```

36.76.2 Methods

The class vtkPPairwiseExtractHistogram2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPPairwiseExtractHistogram2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPPairwiseExtractHistogram2D = obj.NewInstance ()`
- `vtkPPairwiseExtractHistogram2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )`
- `vtkMultiProcessController = obj.GetController ()`

36.77 vtkPPCAStatistics

36.77.1 Usage

vtkPPCAStatistics is vtkPCAStatistics subclass for parallel datasets. It learns and derives the global statistical model on each node, but assesses each individual data points on the node that owns it.

To create an instance of class vtkPPCAStatistics, simply invoke its constructor as follows

```
obj = vtkPPCAStatistics
```

36.77.2 Methods

The class vtkPPCAStatistics has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPPCAStatistics class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPPCAStatistics = obj.NewInstance ()`
- `vtkPPCAStatistics = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.
- `vtkMultiProcessController = obj.GetController ()` - Get/Set the multiprocess controller. If no controller is set, single process is assumed.

36.78 vtkPruneTreeFilter

36.78.1 Usage

Removes a subtree rooted at a particular vertex in a vtkTree.

To create an instance of class vtkPruneTreeFilter, simply invoke its constructor as follows

```
obj = vtkPruneTreeFilter
```

36.78.2 Methods

The class vtkPruneTreeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPruneTreeFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPruneTreeFilter = obj.NewInstance ()`
- `vtkPruneTreeFilter = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetParentVertex ()` - Set the parent vertex of the subtree to remove.
- `obj.SetParentVertex (vtkIdType )` - Set the parent vertex of the subtree to remove.

36.79 vtkRandomGraphSource

36.79.1 Usage

Generates a graph with a specified number of vertices, with the density of edges specified by either an exact number of edges or the probability of an edge. You may additionally specify whether to begin with a random tree (which enforces graph connectivity).

To create an instance of class vtkRandomGraphSource, simply invoke its constructor as follows

```
obj = vtkRandomGraphSource
```

36.79.2 Methods

The class vtkRandomGraphSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRandomGraphSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRandomGraphSource = obj.NewInstance ()`
- `vtkRandomGraphSource = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfVertices ()` - The number of vertices in the graph.
- `obj.SetNumberOfVertices (int )` - The number of vertices in the graph.
- `int = obj.GetNumberOfVerticesMinValue ()` - The number of vertices in the graph.
- `int = obj.GetNumberOfVerticesMaxValue ()` - The number of vertices in the graph.
- `int = obj.GetNumberOfEdges ()` - If UseEdgeProbability is off, creates a graph with the specified number of edges. Duplicate (parallel) edges are allowed.
- `obj.SetNumberOfEdges (int )` - If UseEdgeProbability is off, creates a graph with the specified number of edges. Duplicate (parallel) edges are allowed.
- `int = obj.GetNumberOfEdgesMinValue ()` - If UseEdgeProbability is off, creates a graph with the specified number of edges. Duplicate (parallel) edges are allowed.

- `int = obj.GetNumberOfEdgesMaxValue ()` - If UseEdgeProbability is off, creates a graph with the specified number of edges. Duplicate (parallel) edges are allowed.
- `double = obj.GetEdgeProbability ()` - If UseEdgeProbability is on, adds an edge with this probability between 0 and 1 for each pair of vertices in the graph.
- `obj.SetEdgeProbability (double )` - If UseEdgeProbability is on, adds an edge with this probability between 0 and 1 for each pair of vertices in the graph.
- `double = obj.GetEdgeProbabilityMinValue ()` - If UseEdgeProbability is on, adds an edge with this probability between 0 and 1 for each pair of vertices in the graph.
- `double = obj.GetEdgeProbabilityMaxValue ()` - If UseEdgeProbability is on, adds an edge with this probability between 0 and 1 for each pair of vertices in the graph.
- `obj.SetIncludeEdgeWeights (bool )` - When set, includes edge weights in an array named "edge_weights". Defaults to off. Weights are random between 0 and 1.
- `bool = obj.GetIncludeEdgeWeights ()` - When set, includes edge weights in an array named "edge_weights". Defaults to off. Weights are random between 0 and 1.
- `obj.IncludeEdgeWeightsOn ()` - When set, includes edge weights in an array named "edge_weights". Defaults to off. Weights are random between 0 and 1.
- `obj.IncludeEdgeWeightsOff ()` - When set, includes edge weights in an array named "edge_weights". Defaults to off. Weights are random between 0 and 1.
- `obj.SetEdgeWeightArrayName (string )` - The name of the edge weight array. Default "edge weight".
- `string = obj.GetEdgeWeightArrayName ()` - The name of the edge weight array. Default "edge weight".
- `obj.SetDirected (bool )` - When set, creates a directed graph, as opposed to an undirected graph.
- `bool = obj.GetDirected ()` - When set, creates a directed graph, as opposed to an undirected graph.
- `obj.DirectedOn ()` - When set, creates a directed graph, as opposed to an undirected graph.
- `obj.DirectedOff ()` - When set, creates a directed graph, as opposed to an undirected graph.
- `obj.SetUseEdgeProbability (bool )` - When set, uses the EdgeProbability parameter to determine the density of edges. Otherwise, NumberOfEdges is used.
- `bool = obj.GetUseEdgeProbability ()` - When set, uses the EdgeProbability parameter to determine the density of edges. Otherwise, NumberOfEdges is used.
- `obj.UseEdgeProbabilityOn ()` - When set, uses the EdgeProbability parameter to determine the density of edges. Otherwise, NumberOfEdges is used.
- `obj.UseEdgeProbabilityOff ()` - When set, uses the EdgeProbability parameter to determine the density of edges. Otherwise, NumberOfEdges is used.
- `obj.SetStartWithTree (bool )` - When set, builds a random tree structure first, then adds additional random edges.
- `bool = obj.GetStartWithTree ()` - When set, builds a random tree structure first, then adds additional random edges.
- `obj.StartWithTreeOn ()` - When set, builds a random tree structure first, then adds additional random edges.

- `obj.StartWithTreeOff ()` - When set, builds a random tree structure first, then adds additional random edges.
- `obj.SetAllowSelfLoops (bool )` - If this flag is set to true, edges where the source and target vertex are the same can be generated. The default is to forbid such loops.
- `bool = obj.GetAllowSelfLoops ()` - If this flag is set to true, edges where the source and target vertex are the same can be generated. The default is to forbid such loops.
- `obj.AllowSelfLoopsOn ()` - If this flag is set to true, edges where the source and target vertex are the same can be generated. The default is to forbid such loops.
- `obj.AllowSelfLoopsOff ()` - If this flag is set to true, edges where the source and target vertex are the same can be generated. The default is to forbid such loops.
- `obj.SetAllowParallelEdges (bool )` - When set, multiple edges from a source to a target vertex are allowed. The default is to forbid such loops.
- `bool = obj.GetAllowParallelEdges ()` - When set, multiple edges from a source to a target vertex are allowed. The default is to forbid such loops.
- `obj.AllowParallelEdgesOn ()` - When set, multiple edges from a source to a target vertex are allowed. The default is to forbid such loops.
- `obj.AllowParallelEdgesOff ()` - When set, multiple edges from a source to a target vertex are allowed. The default is to forbid such loops.
- `obj.SetGeneratePedigreeIds (bool )` - Add pedigree ids to vertex and edge data.
- `bool = obj.GetGeneratePedigreeIds ()` - Add pedigree ids to vertex and edge data.
- `obj.GeneratePedigreeIdsOn ()` - Add pedigree ids to vertex and edge data.
- `obj.GeneratePedigreeIdsOff ()` - Add pedigree ids to vertex and edge data.
- `obj.SetVertexPedigreeIdArrayName (string )` - The name of the vertex pedigree id array. Default "vertex id".
- `string = obj.GetVertexPedigreeIdArrayName ()` - The name of the vertex pedigree id array. Default "vertex id".
- `obj.SetEdgePedigreeIdArrayName (string )` - The name of the edge pedigree id array. Default "edge id".
- `string = obj.GetEdgePedigreeIdArrayName ()` - The name of the edge pedigree id array. Default "edge id".
- `obj.SetSeed (int )` - Control the seed used for pseudo-random-number generation. This ensures that vtkRandomGraphSource can produce repeatable results.
- `int = obj.GetSeed ()` - Control the seed used for pseudo-random-number generation. This ensures that vtkRandomGraphSource can produce repeatable results.

36.80 vtkRandomLayoutStrategy

36.80.1 Usage

Assigns points to the vertices of a graph randomly within a bounded range.

.SECION Thanks Thanks to Brian Wylie from Sandia National Laboratories for adding incremental layout capabilities.

To create an instance of class vtkRandomLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkRandomLayoutStrategy
```

36.80.2 Methods

The class vtkRandomLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRandomLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRandomLayoutStrategy = obj.NewInstance ()`
- `vtkRandomLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetRandomSeed (int )` - Seed the random number generator used to compute point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMinValue ()` - Seed the random number generator used to compute point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMaxValue ()` - Seed the random number generator used to compute point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeed ()` - Seed the random number generator used to compute point positions. This has a significant effect on their final positions when the layout is complete.
- `obj.SetGraphBounds (double , double , double , double , double , double )` - Set / get the region in space in which to place the final graph. The GraphBounds only affects the results if AutomaticBoundsComputation is off.
- `obj.SetGraphBounds (double a[6])` - Set / get the region in space in which to place the final graph. The GraphBounds only affects the results if AutomaticBoundsComputation is off.
- `double = obj. GetGraphBounds ()` - Set / get the region in space in which to place the final graph. The GraphBounds only affects the results if AutomaticBoundsComputation is off.
- `obj.SetAutomaticBoundsComputation (int )` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `int = obj.GetAutomaticBoundsComputation ()` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `obj.AutomaticBoundsComputationOn ()` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `obj.AutomaticBoundsComputationOff ()` - Turn on/off automatic graph bounds calculation. If this boolean is off, then the manually specified GraphBounds is used. If on, then the input's bounds us used as the graph bounds.
- `obj.SetThreeDimensionalLayout (int )` - Turn on/off layout of graph in three dimensions. If off, graph layout occurs in two dimensions. By default, three dimensional layout is on.
- `int = obj.GetThreeDimensionalLayout ()` - Turn on/off layout of graph in three dimensions. If off, graph layout occurs in two dimensions. By default, three dimensional layout is on.
- `obj.ThreeDimensionalLayoutOn ()` - Turn on/off layout of graph in three dimensions. If off, graph layout occurs in two dimensions. By default, three dimensional layout is on.

- `obj.ThreeDimensionalLayoutOff ()` - Turn on/off layout of graph in three dimensions. If off, graph layout occurs in two dimensions. By default, three dimensional layout is on.
- `obj.SetGraph (vtkGraph graph)` - Set the graph to layout.
- `obj.Layout ()` - Perform the random layout.

36.81 vtkRemoveHiddenData

36.81.1 Usage

Output only those rows/vertices/edges of the input vtkDataObject that are visible, as defined by the vtkAnnotation::HIDE() flag of the input vtkAnnotationLayers. Inputs: Port 0 - vtkDataObject Port 1 - vtkAnnotationLayers (optional)

To create an instance of class vtkRemoveHiddenData, simply invoke its constructor as follows

```
obj = vtkRemoveHiddenData
```

36.81.2 Methods

The class vtkRemoveHiddenData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRemoveHiddenData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRemoveHiddenData = obj.NewInstance ()`
- `vtkRemoveHiddenData = obj.SafeDownCast (vtkObject o)`

36.82 vtkRemoveIsolatedVertices

36.82.1 Usage

To create an instance of class vtkRemoveIsolatedVertices, simply invoke its constructor as follows

```
obj = vtkRemoveIsolatedVertices
```

36.82.2 Methods

The class vtkRemoveIsolatedVertices has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRemoveIsolatedVertices class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRemoveIsolatedVertices = obj.NewInstance ()`
- `vtkRemoveIsolatedVertices = obj.SafeDownCast (vtkObject o)`

36.83 vtkRISReader

36.83.1 Usage

RIS is a tagged format for expressing bibliographic citations. Data is structured as a collection of records with each record composed of one-to-many fields. See

http://en.wikipedia.org/wiki/RIS_(file_format) http://www.refman.com/support/risformat_intro.asp http://www.adeptsc

for details. vtkRISReader will convert an RIS file into a vtkTable, with the set of table columns determined dynamically from the contents of the file.

To create an instance of class vtkRISReader, simply invoke its constructor as follows

```
obj = vtkRISReader
```

36.83.2 Methods

The class vtkRISReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRISReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRISReader = obj.NewInstance ()`
- `vtkRISReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileName ()` - Set/get the file to load
- `obj.SetFileName (string )` - Set/get the file to load
- `string = obj.GetDelimiter ()` - Set/get the delimiter to be used for concatenating field data (default: ";")
- `obj.SetDelimiter (string )` - Set/get the delimiter to be used for concatenating field data (default: ";")
- `int = obj.GetMaxRecords ()` - Set/get the maximum number of records to read from the file (zero = unlimited)
- `obj.SetMaxRecords (int )` - Set/get the maximum number of records to read from the file (zero = unlimited)

36.84 vtkSCurveSpline

36.84.1 Usage

vtkSCurveSpline is a concrete implementation of vtkSpline using a SCurve basis.

To create an instance of class vtkSCurveSpline, simply invoke its constructor as follows

```
obj = vtkSCurveSpline
```

36.84.2 Methods

The class vtkSCurveSpline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSCurveSpline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSCurveSpline = obj.NewInstance ()`
- `vtkSCurveSpline = obj.SafeDownCast (vtkObject o)`
- `obj.Compute ()`
- `double = obj.Evaluate (double t)` - Evaluate a 1D SCurve spline.
- `obj.DeepCopy (vtkSpline s)` - Deep copy of SCurve spline data.
- `obj.SetNodeWeight (double )`
- `double = obj.GetNodeWeight ()`

36.85 vtkSimple2DLayoutStrategy

36.85.1 Usage

This class is an implementation of the work presented in: Fruchterman & Reingold "Graph Drawing by Force-directed Placement" Software-Practice and Experience 21(11) 1991). The class includes some optimizations but nothing too fancy.

.SECTION Thanks Thanks to Brian Wylie from Sandia National Laboratories for creating this class.

To create an instance of class vtkSimple2DLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkSimple2DLayoutStrategy
```

36.85.2 Methods

The class vtkSimple2DLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSimple2DLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSimple2DLayoutStrategy = obj.NewInstance ()`
- `vtkSimple2DLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetRandomSeed (int )` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMinValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.
- `int = obj.GetRandomSeedMaxValue ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.

- `int = obj.GetRandomSeed ()` - Seed the random number generator used to jitter point positions. This has a significant effect on their final positions when the layout is complete.

- `obj.SetMaxNumberOfIterations (int )` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterationsMinValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterationsMaxValue ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `int = obj.GetMaxNumberOfIterations ()` - Set/Get the maximum number of iterations to be used. The higher this number, the more iterations through the algorithm is possible, and thus, the more the graph gets modified. The default is '100' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `obj.SetIterationsPerLayout (int )` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayoutMinValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayoutMaxValue ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `int = obj.GetIterationsPerLayout ()` - Set/Get the number of iterations per layout. The only use for this ivar is for the application to do visualizations of the layout before it's complete. The default is '100' to match the default 'MaxNumberOfIterations' Note: Changing this parameter is just fine :)

- `obj.SetInitialTemperature (float )` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperatureMinValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperatureMaxValue ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `float = obj.GetInitialTemperature ()` - Set the initial temperature. The temperature default is '5' for no particular reason Note: The strong recommendation is that you do not change this parameter. :)

- `obj.SetCoolDownRate (double )` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRateMinValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRateMaxValue ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `double = obj.GetCoolDownRate ()` - Set/Get the Cool-down rate. The higher this number is, the longer it will take to "cool-down", and thus, the more the graph will be modified. The default is '10' for no particular reason. Note: The strong recommendation is that you do not change this parameter. :)
- `obj.SetJitter (bool )` - Set Random jitter of the nodes at initialization to on or off. Note: It's strongly recommendation to have jitter ON even if you have initial coordinates in your graph. Default is ON
- `bool = obj.GetJitter ()` - Set Random jitter of the nodes at initialization to on or off. Note: It's strongly recommendation to have jitter ON even if you have initial coordinates in your graph. Default is ON
- `obj.SetRestDistance (float )` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `float = obj.GetRestDistance ()` - Manually set the resting distance. Otherwise the distance is computed automatically.
- `obj.Initialize ()` - This strategy sets up some data structures for faster processing of each Layout() call
- `obj.Layout ()` - This is the layout method where the graph that was set in SetGraph() is laid out. The method can either entirely layout the graph or iteratively lay out the graph. If you have an iterative layout please implement the IsLayoutComplete() method.
- `int = obj.IsLayoutComplete ()`

36.86 vtkSimple3DCirclesStrategy

36.86.1 Usage

Places vertices on circles depending on the graph vertices hierarchy level. The source graph could be vtkDirectedAcyclicGraph or vtkDirectedGraph if MarkedStartPoints array was added. The algorithm collects the standalone points, too and take them to a separated circle. If method is FixedRadiusMethod, the radius of the circles will be equal. If method is FixedDistanceMethod, the distance beetwen the points on circles will be equal.

In first step initial points are searched. A point is initial, if its in degree equal zero and out degree is greater than zero (or marked by MarkedStartVertices and out degree is greater than zero). Independent vertices (in and out degree equal zero) are collected separatelly. In second step the hierarchical level is generated for every vertex. In third step the hierarchical order is generated. If a vertex has no hierarchical level and it is not independent, the graph has loop so the algorithm exit with error message. Finally the vertices positions are calculated by the hierarchical order and by the vertices hierarchy levels.

.SECTION Thanks Ferenc Nasztanovics, naszta@naszta.hu, Budapest University of Technology and Economics, Department of Structural Mechanics

.SECTION References in 3D rotation was used: http://en.citizendium.org/wiki/Rotation_matrix

To create an instance of class vtkSimple3DCirclesStrategy, simply invoke its constructor as follows

```
obj = vtkSimple3DCirclesStrategy
```

36.86.2 Methods

The class vtkSimple3DCirclesStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSimple3DCirclesStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSimple3DCirclesStrategy = obj.NewInstance ()`
- `vtkSimple3DCirclesStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetMethod (int )` - Set or get cicrle generating method (FixedRadiusMethod/FixedDistanceMethod). Default is FixedRadiusMethod.
- `int = obj.GetMethod ()` - Set or get cicrle generating method (FixedRadiusMethod/FixedDistanceMethod). Default is FixedRadiusMethod.
- `obj.SetRadius (double )` - If Method is FixedRadiusMethod: Set or get the radius of the circles. If Method is FixedDistanceMethod: Set or get the distance of the points in the circle.
- `double = obj.GetRadius ()` - If Method is FixedRadiusMethod: Set or get the radius of the circles. If Method is FixedDistanceMethod: Set or get the distance of the points in the circle.
- `obj.SetHeight (double )` - Set or get the vertical (local z) distance between the circles. If AutoHeight is on, this is the minimal height between the circle layers
- `double = obj.GetHeight ()` - Set or get the vertical (local z) distance between the circles. If AutoHeight is on, this is the minimal height between the circle layers
- `obj.SetOrign (double , double , double )` - Set or get the orign of the geometry. This is the center of the first circle. SetOrign(x,y,z)
- `obj.SetOrign (double  a[3])` - Set or get the orign of the geometry. This is the center of the first circle. SetOrign(x,y,z)
- `double = obj. GetOrign ()` - Set or get the orign of the geometry. This is the center of the first circle. SetOrign(x,y,z)
- `obj.SetDirection (double dx, double dy, double dz)` - Set or get the normal vector of the circles plain. The height is growing in this direction. The direction must not be zero vector. The default vector is (0.0,0.0,1.0)
- `obj.SetDirection (double d[3])` - Set or get the normal vector of the circles plain. The height is growing in this direction. The direction must not be zero vector. The default vector is (0.0,0.0,1.0)
- `double = obj. GetDirection ()` - Set or get the normal vector of the circles plain. The height is growing in this direction. The direction must not be zero vector. The default vector is (0.0,0.0,1.0)

- `obj.SetMarkedStartVertices (vtkIntArray \_arg)` - Set or get initial vertices. If MarkedStartVertices is added, loop is accepted in the graph. (If all of the loop start vertices are marked in MarkedStartVertices array.) MarkedStartVertices size must be equal with the number of the vertices in the graph. Start vertices must be marked by MarkedValue. (E.g.: if MarkedValue=3 and MarkedStartPoints is 0, 3, 5, 3 , the start points ids will be 1,3.))

- `vtkIntArray = obj.GetMarkedStartVertices ()` - Set or get initial vertices. If MarkedStartVertices is added, loop is accepted in the graph. (If all of the loop start vertices are marked in MarkedStartVertices array.) MarkedStartVertices size must be equal with the number of the vertices in the graph. Start vertices must be marked by MarkedValue. (E.g.: if MarkedValue=3 and MarkedStartPoints is 0, 3, 5, 3 , the start points ids will be 1,3.))

- `obj.SetMarkedValue (int )` - Set or get MarkedValue. See: MarkedStartVertices.

- `int = obj.GetMarkedValue ()` - Set or get MarkedValue. See: MarkedStartVertices.

- `obj.SetForceToUseUniversalStartPointsFinder (int )` - Set or get ForceToUseUniversalStartPointsFinder. If ForceToUseUniversalStartPointsFinder is true, MarkedStartVertices won't be used. In this case the input graph must be vtkDirectedAcyclicGraph (Defualt: false).

- `int = obj.GetForceToUseUniversalStartPointsFinder ()` - Set or get ForceToUseUniversalStartPointsFinder. If ForceToUseUniversalStartPointsFinder is true, MarkedStartVertices won't be used. In this case the input graph must be vtkDirectedAcyclicGraph (Defualt: false).

- `obj.ForceToUseUniversalStartPointsFinderOn ()` - Set or get ForceToUseUniversalStartPointsFinder. If ForceToUseUniversalStartPointsFinder is true, MarkedStartVertices won't be used. In this case the input graph must be vtkDirectedAcyclicGraph (Defualt: false).

- `obj.ForceToUseUniversalStartPointsFinderOff ()` - Set or get ForceToUseUniversalStartPointsFinder. If ForceToUseUniversalStartPointsFinder is true, MarkedStartVertices won't be used. In this case the input graph must be vtkDirectedAcyclicGraph (Defualt: false).

- `obj.SetAutoHeight (int )` - Set or get auto height (Default: false). If AutoHeight is true, (r(i+1) - r(i-1))/Height will be smaller than tan(MinimumRadian). If you want equal distances and parallel circles, you should turn off AutoHeight.

- `int = obj.GetAutoHeight ()` - Set or get auto height (Default: false). If AutoHeight is true, (r(i+1) - r(i-1))/Height will be smaller than tan(MinimumRadian). If you want equal distances and parallel circles, you should turn off AutoHeight.

- `obj.AutoHeightOn ()` - Set or get auto height (Default: false). If AutoHeight is true, (r(i+1) - r(i-1))/Height will be smaller than tan(MinimumRadian). If you want equal distances and parallel circles, you should turn off AutoHeight.

- `obj.AutoHeightOff ()` - Set or get auto height (Default: false). If AutoHeight is true, (r(i+1) - r(i-1))/Height will be smaller than tan(MinimumRadian). If you want equal distances and parallel circles, you should turn off AutoHeight.

- `obj.SetMinimumRadian (double )` - Set or get minimum radian (used by auto height).

- `double = obj.GetMinimumRadian ()` - Set or get minimum radian (used by auto height).

- `obj.SetMinimumDegree (double degree)` - Set or get minimum degree (used by auto height). There is no separated minimum degree, so minimum radian will be changed.

- `double = obj.GetMinimumDegree (void )` - Set or get minimum degree (used by auto height). There is no separated minimum degree, so minimum radian will be changed.

- `obj.SetHierarchicalLayers (vtkIntArray \_arg)` - Set or get hierarchical layers id by vertices (An usual vertex's layer id is greater or equal to zero. If a vertex is standalone, its layer id is -2.) If no HierarchicalLayers array is defined, vtkSimple3DCirclesStrategy will generate it automatically (default).

- `vtkIntArray = obj.GetHierarchicalLayers ()` - Set or get hierarchical layers id by vertices (An usual vertex's layer id is greater or equal to zero. If a vertex is standalone, its layer id is -2.) If no HierarchicalLayers array is defined, vtkSimple3DCirclesStrategy will generate it automatically (default).

- `obj.SetHierarchicalOrder (vtkIdTypeArray \_arg)` - Set or get hierarchical ordering of vertices (The array starts from the first vertex's id. All id must be greater or equal to zero!) If no HierarchicalOrder is defined, vtkSimple3DCirclesStrategy will generate it automatically (default).

- `vtkIdTypeArray = obj.GetHierarchicalOrder ()` - Set or get hierarchical ordering of vertices (The array starts from the first vertex's id. All id must be greater or equal to zero!) If no HierarchicalOrder is defined, vtkSimple3DCirclesStrategy will generate it automatically (default).

- `obj.Layout (void )` - Standard layout method

- `obj.SetGraph (vtkGraph graph)` - Set graph (warning: HierarchicalOrder and HierarchicalLayers will set to zero. These reference counts will be decreased!)

36.87 vtkSliceAndDiceLayoutStrategy

36.87.1 Usage

Lays out a tree-map alternating between horizontal and vertical slices, taking into account the relative size of each vertex.

.SECTION Thanks Slice and dice algorithm comes from: Shneiderman, B. 1992. Tree visualization with tree-maps: 2-d space-filling approach. ACM Trans. Graph. 11, 1 (Jan. 1992), 92-99.

To create an instance of class vtkSliceAndDiceLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkSliceAndDiceLayoutStrategy
```

36.87.2 Methods

The class vtkSliceAndDiceLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSliceAndDiceLayoutStrategy class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkSliceAndDiceLayoutStrategy = obj.NewInstance ()`

- `vtkSliceAndDiceLayoutStrategy = obj.SafeDownCast (vtkObject o)`

- `obj.Layout (vtkTree inputTree, vtkDataArray coordsArray, vtkDataArray sizeArray)` - Perform the layout of a tree and place the results as 4-tuples in coordsArray (Xmin, Xmax, Ymin, Ymax).

36.88 vtkSpanTreeLayoutStrategy

36.88.1 Usage

vtkSpanTreeLayout is a strategy for drawing directed graphs that works by first extracting a spanning tree (more accurately, a spanning forest), and using this both to position graph vertices and to plan the placement of non-tree edges. The latter are drawn with the aid of edge points to produce a tidy drawing.

The approach is best suited to "quasi-trees", graphs where the number of edges is of the same order as the number of nodes; it is less well suited to denser graphs. The boolean flag DepthFirstSpanningTree determines whether a depth-first or breadth-first strategy is used to construct the underlying forest, and the choice of strategy affects the output layout significantly. Informal experiments suggest that the breadth-first strategy is better for denser graphs.

Different layouts could also be produced by plugging in alternative tree layout strategies. To work with the method of routing non-tree edges, any strategy should draw a tree so that levels are equally spaced along the z-axis, precluding for example the use of a radial or balloon layout.

vtkSpanTreeLayout is based on an approach to 3D graph layout first developed as part of the "tulip" tool by Dr. David Auber at LaBRI, U.Bordeaux: see www.tulip-software.org

This implementation departs from the original version in that: (a) it is reconstructed to use Titan/VTK data structures; (b) it uses a faster method for dealing with non-tree edges, requiring at most two edge points per edge (c) allows for plugging in different tree layout methods (d) allows selection of two different strategies for building the underlying layout tree, which can yield significantly different results depending on the data.

.SECTION Thanks Thanks to David Duke from the University of Leeds for providing this implementation.

To create an instance of class vtkSpanTreeLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkSpanTreeLayoutStrategy
```

36.88.2 Methods

The class vtkSpanTreeLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSpanTreeLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSpanTreeLayoutStrategy = obj.NewInstance ()`
- `vtkSpanTreeLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.SetDepthFirstSpanningTree (bool )` - If set, base the layout on a depth-first spanning tree, rather than the default breadth-first spanning tree. Switching between DFT and BFT may significantly change the layout, and choice must be made on a per-graph basis. Default value is off.
- `bool = obj.GetDepthFirstSpanningTree ()` - If set, base the layout on a depth-first spanning tree, rather than the default breadth-first spanning tree. Switching between DFT and BFT may significantly change the layout, and choice must be made on a per-graph basis. Default value is off.
- `obj.DepthFirstSpanningTreeOn ()` - If set, base the layout on a depth-first spanning tree, rather than the default breadth-first spanning tree. Switching between DFT and BFT may significantly change the layout, and choice must be made on a per-graph basis. Default value is off.
- `obj.DepthFirstSpanningTreeOff ()` - If set, base the layout on a depth-first spanning tree, rather than the default breadth-first spanning tree. Switching between DFT and BFT may significantly change the layout, and choice must be made on a per-graph basis. Default value is off.
- `obj.Layout ()` - Perform the layout.

36.89 vtkSparseArrayToTable

36.89.1 Usage

Converts any sparse array to a vtkTable containing one row for each value stored in the array. The table will contain one column of coordinates for each dimension in the source array, plus one column of array values. A common use-case for vtkSparseArrayToTable would be converting a sparse array into a table suitable for use as an input to vtkTableToGraph.

The coordinate columns in the output table will be named using the dimension labels from the source array, The value column name can be explicitly set using SetValueColumn().

.SECTION Thanks Developed by Timothy M. Shead (tshead@sandia.gov) at Sandia National Laboratories.

To create an instance of class vtkSparseArrayToTable, simply invoke its constructor as follows

```
obj = vtkSparseArrayToTable
```

36.89.2 Methods

The class vtkSparseArrayToTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSparseArrayToTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSparseArrayToTable = obj.NewInstance ()`
- `vtkSparseArrayToTable = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetValueColumn ()` - Specify the name of the output table column that contains array values. Default: "value"
- `obj.SetValueColumn (string )` - Specify the name of the output table column that contains array values. Default: "value"

36.90 vtkSplineGraphEdges

36.90.1 Usage

vtkSplineGraphEdges uses a vtkSpline to make edges into nicely sampled splines. By default, the filter will use an optimized b-spline. Otherwise, it will use a custom vtkSpline instance set by the user.

To create an instance of class vtkSplineGraphEdges, simply invoke its constructor as follows

```
obj = vtkSplineGraphEdges
```

36.90.2 Methods

The class vtkSplineGraphEdges has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSplineGraphEdges class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSplineGraphEdges = obj.NewInstance ()`

- `vtkSplineGraphEdges = obj.SafeDownCast (vtkObject o)`
- `obj.SetSpline (vtkSpline s)` - If SplineType is CUSTOM, uses this spline.
- `vtkSpline = obj.GetSpline ()` - If SplineType is CUSTOM, uses this spline.
- `obj.SetSplineType (int )` - Spline type used by the filter. BSPLINE (0) - Use optimized b-spline (default). CUSTOM (1) - Use spline set with SetSpline.
- `int = obj.GetSplineType ()` - Spline type used by the filter. BSPLINE (0) - Use optimized b-spline (default). CUSTOM (1) - Use spline set with SetSpline.
- `obj.SetNumberOfSubdivisions (vtkIdType )` - The number of subdivisions in the spline.
- `vtkIdType = obj.GetNumberOfSubdivisions ()` - The number of subdivisions in the spline.

36.91 vtkSplitColumnComponents

36.91.1 Usage

Splits any columns in a table that have more than one component into individual columns. Single component columns are passed through without any data duplication. So if column names "Points" had three components this column would be split into "Points (0)", "Points (1)" and Points (2)".

To create an instance of class vtkSplitColumnComponents, simply invoke its constructor as follows

```
obj = vtkSplitColumnComponents
```

36.91.2 Methods

The class vtkSplitColumnComponents has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSplitColumnComponents class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSplitColumnComponents = obj.NewInstance ()`
- `vtkSplitColumnComponents = obj.SafeDownCast (vtkObject o)`
- `obj.SetCalculateMagnitudes (bool )` - If on this filter will calculate an additional magnitude column for all columns it splits with two or more components. Default is on.
- `bool = obj.GetCalculateMagnitudes ()` - If on this filter will calculate an additional magnitude column for all columns it splits with two or more components. Default is on.

36.92 vtkSQLDatabaseGraphSource

36.92.1 Usage

This class combines vtkSQLDatabase, vtkSQLQuery, and vtkQueryToGraph to provide a convenience class for generating graphs from databases. Also this class can be easily wrapped and used within ParaView / OverView.

To create an instance of class vtkSQLDatabaseGraphSource, simply invoke its constructor as follows

```
obj = vtkSQLDatabaseGraphSource
```

36.92.2 Methods

The class vtkSQLDatabaseGraphSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSQLDatabaseGraphSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSQLDatabaseGraphSource = obj.NewInstance ()`
- `vtkSQLDatabaseGraphSource = obj.SafeDownCast (vtkObject o)`
- `vtkStdString = obj.GetURL ()`
- `obj.SetURL (vtkStdString \&url)`
- `obj.SetPassword (vtkStdString \&password)`
- `vtkStdString = obj.GetEdgeQuery ()`
- `obj.SetEdgeQuery (vtkStdString \&query)`
- `vtkStdString = obj.GetVertexQuery ()`
- `obj.SetVertexQuery (vtkStdString \&query)`
- `obj.AddLinkVertex (string column, string domain, int hidden)`
- `obj.ClearLinkVertices ()`
- `obj.AddLinkEdge (string column1, string column2)`
- `obj.ClearLinkEdges ()`
- `bool = obj.GetGenerateEdgePedigreeIds ()` - If on (default), generate edge pedigree ids. If off, assign an array to be edge pedigree ids.
- `obj.SetGenerateEdgePedigreeIds (bool )` - If on (default), generate edge pedigree ids. If off, assign an array to be edge pedigree ids.
- `obj.GenerateEdgePedigreeIdsOn ()` - If on (default), generate edge pedigree ids. If off, assign an array to be edge pedigree ids.
- `obj.GenerateEdgePedigreeIdsOff ()` - If on (default), generate edge pedigree ids. If off, assign an array to be edge pedigree ids.
- `obj.SetEdgePedigreeIdArrayName (string )` - Use this array name for setting or generating edge pedigree ids.
- `string = obj.GetEdgePedigreeIdArrayName ()` - Use this array name for setting or generating edge pedigree ids.
- `obj.SetDirected (bool )` - If on (default), generate a directed output graph. If off, generate an undirected output graph.
- `bool = obj.GetDirected ()` - If on (default), generate a directed output graph. If off, generate an undirected output graph.
- `obj.DirectedOn ()` - If on (default), generate a directed output graph. If off, generate an undirected output graph.
- `obj.DirectedOff ()` - If on (default), generate a directed output graph. If off, generate an undirected output graph.

36.93 vtkSQLDatabaseTableSource

36.93.1 Usage

This class combines vtkSQLDatabase, vtkSQLQuery, and vtkQueryToTable to provide a convenience class for generating tables from databases. Also this class can be easily wrapped and used within ParaView / OverView.

To create an instance of class vtkSQLDatabaseTableSource, simply invoke its constructor as follows

```
obj = vtkSQLDatabaseTableSource
```

36.93.2 Methods

The class vtkSQLDatabaseTableSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSQLDatabaseTableSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSQLDatabaseTableSource = obj.NewInstance ()`
- `vtkSQLDatabaseTableSource = obj.SafeDownCast (vtkObject o)`
- `vtkStdString = obj.GetURL ()`
- `obj.SetURL (vtkStdString \&url)`
- `obj.SetPassword (vtkStdString \&password)`
- `vtkStdString = obj.GetQuery ()`
- `obj.SetQuery (vtkStdString \&query)`
- `obj.SetPedigreeIdArrayName (string )` - The name of the array for generating or assigning pedigree ids (default "id").
- `string = obj.GetPedigreeIdArrayName ()` - The name of the array for generating or assigning pedigree ids (default "id").
- `obj.SetGeneratePedigreeIds (bool )` - If on (default), generates pedigree ids automatically. If off, assign one of the arrays to be the pedigree id.
- `bool = obj.GetGeneratePedigreeIds ()` - If on (default), generates pedigree ids automatically. If off, assign one of the arrays to be the pedigree id.
- `obj.GeneratePedigreeIdsOn ()` - If on (default), generates pedigree ids automatically. If off, assign one of the arrays to be the pedigree id.
- `obj.GeneratePedigreeIdsOff ()` - If on (default), generates pedigree ids automatically. If off, assign one of the arrays to be the pedigree id.

36.94 vtkSQLGraphReader

36.94.1 Usage

Creates a vtkGraph using one or two vtkSQLQuery's. The first (required) query must have one row for each arc in the graph. The query must have two columns which represent the source and target node ids.

The second (optional) query has one row for each node in the graph. The table must have a field whose values match those in the arc table. If the node table is not given, a node will be created for each unique source or target identifier in the arc table.

The source, target, and node ID fields must be of the same type, and must be either vtkStringArray or a subclass of vtkDataArray.

All columns in the queries, including the source, target, and node index fields, are copied into the arc data and node data of the resulting vtkGraph. If the node query is not given, the node data will contain a single "id" column with the same type as the source/target id arrays.

If parallel arcs are collected, not all the arc data is not copied into the output. Only the source and target id arrays will be transferred. An additional vtkIdTypeArray column called "weight" is created which contains the number of times each arc appeared in the input.

If the node query contains positional data, the user may specify the names of these fields. These arrays must be data arrays. The z-coordinate array is optional, and if not given the z-coordinates are set to zero.

To create an instance of class vtkSQLGraphReader, simply invoke its constructor as follows

```
obj = vtkSQLGraphReader
```

36.94.2 Methods

The class vtkSQLGraphReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSQLGraphReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSQLGraphReader = obj.NewInstance ()`
- `vtkSQLGraphReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetDirected (bool )` - When set, creates a directed graph, as opposed to an undirected graph.
- `bool = obj.GetDirected ()` - When set, creates a directed graph, as opposed to an undirected graph.
- `obj.DirectedOn ()` - When set, creates a directed graph, as opposed to an undirected graph.
- `obj.DirectedOff ()` - When set, creates a directed graph, as opposed to an undirected graph.
- `obj.SetVertexQuery (vtkSQLQuery q)` - The query that retrieves the node information.
- `vtkSQLQuery = obj.GetVertexQuery ()` - The query that retrieves the node information.
- `obj.SetEdgeQuery (vtkSQLQuery q)` - The query that retrieves the arc information.
- `vtkSQLQuery = obj.GetEdgeQuery ()` - The query that retrieves the arc information.
- `obj.SetSourceField (string )` - The name of the field in the arc query for the source node of each arc.
- `string = obj.GetSourceField ()` - The name of the field in the arc query for the source node of each arc.

- `obj.SetTargetField (string )` - The name of the field in the arc query for the target node of each arc.
- `string = obj.GetTargetField ()` - The name of the field in the arc query for the target node of each arc.
- `obj.SetVertexIdField (string )` - The name of the field in the node query for the node ID.
- `string = obj.GetVertexIdField ()` - The name of the field in the node query for the node ID.
- `obj.SetXField (string )` - The name of the field in the node query for the node's x coordinate.
- `string = obj.GetXField ()` - The name of the field in the node query for the node's x coordinate.
- `obj.SetYField (string )` - The name of the field in the node query for the node's y coordinate.
- `string = obj.GetYField ()` - The name of the field in the node query for the node's y coordinate.
- `obj.SetZField (string )` - The name of the field in the node query for the node's z coordinate.
- `string = obj.GetZField ()` - The name of the field in the node query for the node's z coordinate.
- `obj.SetCollapseEdges (bool )` - When set, creates a graph with no parallel arcs. Parallel arcs are combined into one arc. No cell fields are passed to the output, except the vtkGhostLevels array if it exists, but a new field "weight" is created that holds the number of duplicates of that arc in the input.
- `bool = obj.GetCollapseEdges ()` - When set, creates a graph with no parallel arcs. Parallel arcs are combined into one arc. No cell fields are passed to the output, except the vtkGhostLevels array if it exists, but a new field "weight" is created that holds the number of duplicates of that arc in the input.
- `obj.CollapseEdgesOn ()` - When set, creates a graph with no parallel arcs. Parallel arcs are combined into one arc. No cell fields are passed to the output, except the vtkGhostLevels array if it exists, but a new field "weight" is created that holds the number of duplicates of that arc in the input.
- `obj.CollapseEdgesOff ()` - When set, creates a graph with no parallel arcs. Parallel arcs are combined into one arc. No cell fields are passed to the output, except the vtkGhostLevels array if it exists, but a new field "weight" is created that holds the number of duplicates of that arc in the input.

36.95 vtkSquarifyLayoutStrategy

36.95.1 Usage

vtkSquarifyLayoutStrategy partitions the space for child vertices into regions that use all avaliable space and are as close to squares as possible. The algorithm also takes into account the relative vertex size.

.SECTION Thanks The squarified tree map algorithm comes from: Bruls, D.M., C. Huizing, J.J. van Wijk. Squarified Treemaps. In: W. de Leeuw, R. van Liere (eds.), Data Visualization 2000, Proceedings of the joint Eurographics and IEEE TCVG Symposium on Visualization, 2000, Springer, Vienna, p. 33-42.

To create an instance of class vtkSquarifyLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkSquarifyLayoutStrategy
```

36.95.2 Methods

The class vtkSquarifyLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSquarifyLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSquarifyLayoutStrategy = obj.NewInstance ()`
- `vtkSquarifyLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout (vtkTree inputTree, vtkDataArray coordsArray, vtkDataArray sizeArray)` - Perform the layout of a tree and place the results as 4-tuples in coordsArray (Xmin, Xmax, Ymin, Ymax).

36.96 vtkStackedTreeLayoutStrategy

36.96.1 Usage

Performs a tree ring layout or "icicle" layout on a tree. This involves assigning a sector region to each vertex in the tree, and placing that information in a data array with four components per tuple representing (innerRadius, outerRadius, startAngle, endAngle).

This class may be assigned as the layout strategy to vtkAreaLayout.

.SECTION Thanks Thanks to Jason Shepherd from Sandia National Laboratories for help developing this class.

To create an instance of class vtkStackedTreeLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkStackedTreeLayoutStrategy
```

36.96.2 Methods

The class vtkStackedTreeLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStackedTreeLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStackedTreeLayoutStrategy = obj.NewInstance ()`
- `vtkStackedTreeLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout (vtkTree inputTree, vtkDataArray sectorArray, vtkDataArray sizeArray)` - Perform the layout of the input tree, and store the sector bounds of each vertex as a tuple (innerRadius, outerRadius, startAngle, endAngle) in a data array.
- `obj.LayoutEdgePoints (vtkTree inputTree, vtkDataArray sectorArray, vtkDataArray sizeArray, vtkTree` - Fill edgeRoutingTree with points suitable for routing edges of an overlaid graph.
- `obj.SetInteriorRadius (double )` - Define the tree ring's interior radius.
- `double = obj.GetInteriorRadius ()` - Define the tree ring's interior radius.
- `obj.SetRingThickness (double )` - Define the thickness of each of the tree rings.

- `double = obj.GetRingThickness ()` - Define the thickness of each of the tree rings.
- `obj.SetRootStartAngle (double )` - Define the start angle for the root node. NOTE: It is assumed that the root end angle is greater than the root start angle and subtends no more than 360 degrees.
- `double = obj.GetRootStartAngle ()` - Define the start angle for the root node. NOTE: It is assumed that the root end angle is greater than the root start angle and subtends no more than 360 degrees.
- `obj.SetRootEndAngle (double )` - Define the end angle for the root node. NOTE: It is assumed that the root end angle is greater than the root start angle and subtends no more than 360 degrees.
- `double = obj.GetRootEndAngle ()` - Define the end angle for the root node. NOTE: It is assumed that the root end angle is greater than the root start angle and subtends no more than 360 degrees.
- `obj.SetUseRectangularCoordinates (bool )` - Define whether or not rectangular coordinates are being used (as opposed to polar coordinates).
- `bool = obj.GetUseRectangularCoordinates ()` - Define whether or not rectangular coordinates are being used (as opposed to polar coordinates).
- `obj.UseRectangularCoordinatesOn ()` - Define whether or not rectangular coordinates are being used (as opposed to polar coordinates).
- `obj.UseRectangularCoordinatesOff ()` - Define whether or not rectangular coordinates are being used (as opposed to polar coordinates).
- `obj.SetReverse (bool )` - Define whether to reverse the order of the tree stacks from low to high.
- `bool = obj.GetReverse ()` - Define whether to reverse the order of the tree stacks from low to high.
- `obj.ReverseOn ()` - Define whether to reverse the order of the tree stacks from low to high.
- `obj.ReverseOff ()` - Define whether to reverse the order of the tree stacks from low to high.
- `obj.SetInteriorLogSpacingValue (double )` - The spacing of tree levels in the edge routing tree. Levels near zero give more space to levels near the root, while levels near one (the default) create evenly-spaced levels. Levels above one give more space to levels near the leaves.
- `double = obj.GetInteriorLogSpacingValue ()` - The spacing of tree levels in the edge routing tree. Levels near zero give more space to levels near the root, while levels near one (the default) create evenly-spaced levels. Levels above one give more space to levels near the leaves.
- `vtkIdType = obj.FindVertex (vtkTree tree, vtkDataArray array, float pnt[2])` - Returns the vertex id that contains pnt (or -1 if no one contains it).

36.97 vtkStatisticsAlgorithm

36.97.1 Usage

All statistics algorithms can conceptually be operated with several options: * Learn: given an input data set, calculate a minimal statistical model (e.g., sums, raw moments, joint probabilities). * Derive: given an input minimal statistical model, derive the full model (e.g., descriptive statistics, quantiles, correlations, conditional probabilities). NB: It may be, or not be, a problem that a full model was not derived. For instance, when doing parallel calculations, one only wants to derive the full model after all partial calculations have completed. On the other hand, one can also directly provide a full model, that was previously calculated or guessed, and not derive a new one. * Assess: given an input data set, input statistics, and some form of threshold, assess a subset of the data set. * Test: perform at least one statistical test. Therefore, a vtkStatisticsAlgorithm has the following vtkTable ports * 3 input ports: * Data (mandatory) * Parameters to the learn phase (optional) * Input model (optional) * 3 output port (called Output): * Data (annotated

with assessments when the Assess option is ON). * Output model (identical to the the input model when Learn option is OFF). * Meta information about the model and/or the overall fit of the data to the model; is filled only when the Assess option is ON.

.SECTION Thanks Thanks to Philippe Pebay and David Thompson from Sandia National Laboratories for implementing this class.

To create an instance of class vtkStatisticsAlgorithm, simply invoke its constructor as follows

```
obj = vtkStatisticsAlgorithm
```

36.97.2 Methods

The class vtkStatisticsAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStatisticsAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStatisticsAlgorithm = obj.NewInstance ()`
- `vtkStatisticsAlgorithm = obj.SafeDownCast (vtkObject o)`
- `obj.SetLearnOptionParameterConnection (vtkAlgorithmOutput params)` - A convenience method for setting learn input parameters (if one is expected or allowed). It is equivalent to calling SetInput(1, params);
- `obj.SetLearnOptionParameters (vtkDataObject params)` - A convenience method for setting the input model (if one is expected or allowed). It is equivalent to calling SetInputConnection(2, model);
- `obj.SetInputModelConnection (vtkAlgorithmOutput model)` - //
- `obj.SetInputModel (vtkDataObject model)` - Set/Get the Learn option.
- `obj.SetLearnOption (bool )` - Set/Get the Learn option.
- `bool = obj.GetLearnOption ()` - Set/Get the Learn option.
- `obj.SetDeriveOption (bool )` - Set/Get the Derive option.
- `bool = obj.GetDeriveOption ()` - Set/Get the Derive option.
- `obj.SetAssessOption (bool )` - Set/Get the Assess option.
- `bool = obj.GetAssessOption ()` - Set/Get the Assess option.
- `obj.SetTestOption (bool )` - Set/Get the Test option.
- `bool = obj.GetTestOption ()` - Set/Get the Test option.
- `obj.SetAssessParameters (vtkStringArray )` - Set/get assessment parameters.
- `vtkStringArray = obj.GetAssessParameters ()` - Set/get assessment parameters.
- `obj.SetAssessNames (vtkStringArray )` - Set/get assessment names.
- `vtkStringArray = obj.GetAssessNames ()` - Set/get assessment names.

- `obj.SetColumnStatus (string namCol, int status)` - Add or remove a column from the current analysis request. Once all the column status values are set, call RequestSelectedColumns() before selecting another set of columns for a different analysis request. The way that columns selections are used varies from algorithm to algorithm.

  Note: the set of selected columns is maintained in vtkStatisticsAlgorithmPrivate::Buffer until RequestSelectedColumns() is called, at which point the set is appended to vtkStatisticsAlgorithmPrivate::Requests. If there are any columns in vtkStatisticsAlgorithmPrivate::Buffer at the time RequestData() is called, RequestSelectedColumns() will be called and the selection added to the list of requests.

- `obj.ResetAllColumnStates ()` - Set the the status of each and every column in the current request to OFF (0).

- `int = obj.RequestSelectedColumns ()` - Use the current column status values to produce a new request for statistics to be produced when RequestData() is called. See SetColumnStatus() for more information.

- `obj.ResetRequests ()` - Empty the list of current requests.

- `vtkIdType = obj.GetNumberOfRequests ()` - Return the number of requests. This does not include any request that is in the column-status buffer but for which RequestSelectedColumns() has not yet been called (even though it is possible this request will be honored when the filter is run – see SetColumnStatus() for more information).

- `vtkIdType = obj.GetNumberOfColumnsForRequest (vtkIdType request)` - Return the number of columns for a given request.

- `string = obj.GetColumnForRequest (vtkIdType r, vtkIdType c)` - Provide the name of the c-th column for the r-th request.

  For the version of this routine that returns an integer, if the request or column does not exist because r or c is out of bounds, this routine returns 0 and the value of columnName is unspecified. Otherwise, it returns 1 and the value of columnName is set.

  For the version of this routine that returns const char*, if the request or column does not exist because r or c is out of bounds, the routine returns NULL. Otherwise it returns the column name. This version is not thread-safe.

- `obj.Aggregate (vtkDataObjectCollection , vtkDataObject )` - Given a collection of models, calculate aggregate model

36.98 vtkStrahlerMetric

36.98.1 Usage

The Strahler metric is a value assigned to each vertex of a tree that characterizes the structural complexity of the sub-tree rooted at that node. The metric originated in the study of river systems, but has been applied to other tree- structured systes, Details of the metric and the rationale for using it in infovis can be found in:

Tree Visualization and Navigation Clues for Information Visualization, I. Herman, M. Delest, and G. Melancon, Computer Graphics Forum, Vol 17(2), Blackwell, 1998.

The input tree is copied to the output, but with a new array added to the output vertex data.

.SECTION Thanks Thanks to David Duke from the University of Leeds for providing this implementation.

To create an instance of class vtkStrahlerMetric, simply invoke its constructor as follows

```
obj = vtkStrahlerMetric
```

36.98.2 Methods

The class vtkStrahlerMetric has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStrahlerMetric class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStrahlerMetric = obj.NewInstance ()`
- `vtkStrahlerMetric = obj.SafeDownCast (vtkObject o)`
- `obj.SetMetricArrayName (string )` - Set the name of the array in which the Strahler values will be stored within the output vertex data. Default is "Strahler"
- `obj.SetNormalize (int )` - Set/get setting of normalize flag. If this is set, the Strahler values are scaled into the range [0..1]. Default is for normalization to be OFF.
- `int = obj.GetNormalize ()` - Set/get setting of normalize flag. If this is set, the Strahler values are scaled into the range [0..1]. Default is for normalization to be OFF.
- `obj.NormalizeOn ()` - Set/get setting of normalize flag. If this is set, the Strahler values are scaled into the range [0..1]. Default is for normalization to be OFF.
- `obj.NormalizeOff ()` - Set/get setting of normalize flag. If this is set, the Strahler values are scaled into the range [0..1]. Default is for normalization to be OFF.
- `float = obj.GetMaxStrahler ()` - Get the maximum strahler value for the tree.

36.99 vtkStreamGraph

36.99.1 Usage

vtkStreamGraph iteratively collects information from the input graph and combines it in the output graph. It internally maintains a graph instance that is incrementally updated every time the filter is called.

Each update, vtkMergeGraphs is used to combine this filter's input with the internal graph.

To create an instance of class vtkStreamGraph, simply invoke its constructor as follows

```
obj = vtkStreamGraph
```

36.99.2 Methods

The class vtkStreamGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStreamGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStreamGraph = obj.NewInstance ()`
- `vtkStreamGraph = obj.SafeDownCast (vtkObject o)`
- `obj.SetMaxEdges (vtkIdType )` - The maximum number of edges in the combined graph. Default is -1, which specifies that there should be no limit on the number of edges.
- `vtkIdType = obj.GetMaxEdges ()` - The maximum number of edges in the combined graph. Default is -1, which specifies that there should be no limit on the number of edges.

36.100 vtkStringToCategory

36.100.1 Usage

vtkStringToCategory creates an integer array named "category" based on the values in a string array. You may use this filter to create an array that you may use to color points/cells by the values in a string array. Currently there is not support to color by a string array directly. The category values will range from zero to N-1, where N is the number of distinct strings in the string array. Set the string array to process with SetInputArrayToProcess(0,0,0,...). The array may be in the point, cell, or field data of the data object.

To create an instance of class vtkStringToCategory, simply invoke its constructor as follows

```
obj = vtkStringToCategory
```

36.100.2 Methods

The class vtkStringToCategory has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStringToCategory class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStringToCategory = obj.NewInstance ()`
- `vtkStringToCategory = obj.SafeDownCast (vtkObject o)`
- `obj.SetCategoryArrayName (string )` - The name to give to the output vtkIntArray of category values.
- `string = obj.GetCategoryArrayName ()` - The name to give to the output vtkIntArray of category values.

36.101 vtkStringToNumeric

36.101.1 Usage

vtkStringToNumeric is a filter for converting a string array into a numeric arrays.

To create an instance of class vtkStringToNumeric, simply invoke its constructor as follows

```
obj = vtkStringToNumeric
```

36.101.2 Methods

The class vtkStringToNumeric has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStringToNumeric class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStringToNumeric = obj.NewInstance ()`
- `vtkStringToNumeric = obj.SafeDownCast (vtkObject o)`
- `obj.SetConvertFieldData (bool )` - Whether to detect and convert field data arrays. Default is on.

- `bool = obj.GetConvertFieldData ()` - Whether to detect and convert field data arrays. Default is on.
- `obj.ConvertFieldDataOn ()` - Whether to detect and convert field data arrays. Default is on.
- `obj.ConvertFieldDataOff ()` - Whether to detect and convert field data arrays. Default is on.
- `obj.SetConvertPointData (bool )` - Whether to detect and convert cell data arrays. Default is on.
- `bool = obj.GetConvertPointData ()` - Whether to detect and convert cell data arrays. Default is on.
- `obj.ConvertPointDataOn ()` - Whether to detect and convert cell data arrays. Default is on.
- `obj.ConvertPointDataOff ()` - Whether to detect and convert cell data arrays. Default is on.
- `obj.SetConvertCellData (bool )` - Whether to detect and convert point data arrays. Default is on.
- `bool = obj.GetConvertCellData ()` - Whether to detect and convert point data arrays. Default is on.
- `obj.ConvertCellDataOn ()` - Whether to detect and convert point data arrays. Default is on.
- `obj.ConvertCellDataOff ()` - Whether to detect and convert point data arrays. Default is on.
- `obj.SetConvertVertexData (bool b)` - Whether to detect and convert vertex data arrays. Default is on.
- `bool = obj.GetConvertVertexData ()` - Whether to detect and convert vertex data arrays. Default is on.
- `obj.ConvertVertexDataOn ()` - Whether to detect and convert vertex data arrays. Default is on.
- `obj.ConvertVertexDataOff ()` - Whether to detect and convert vertex data arrays. Default is on.
- `obj.SetConvertEdgeData (bool b)` - Whether to detect and convert edge data arrays. Default is on.
- `bool = obj.GetConvertEdgeData ()` - Whether to detect and convert edge data arrays. Default is on.
- `obj.ConvertEdgeDataOn ()` - Whether to detect and convert edge data arrays. Default is on.
- `obj.ConvertEdgeDataOff ()` - Whether to detect and convert edge data arrays. Default is on.
- `obj.SetConvertRowData (bool b)` - Whether to detect and convert row data arrays. Default is on.
- `bool = obj.GetConvertRowData ()` - Whether to detect and convert row data arrays. Default is on.
- `obj.ConvertRowDataOn ()` - Whether to detect and convert row data arrays. Default is on.
- `obj.ConvertRowDataOff ()` - Whether to detect and convert row data arrays. Default is on.

36.102 vtkStringToTimePoint

36.102.1 Usage

vtkStringToTimePoint is a filter for converting a string array into a datetime, time or date array. The input strings must conform to one of the ISO8601 formats defined in vtkTimePointUtility.

The input array specified by SetInputArrayToProcess(...) indicates the array to process. This array must be of type vtkStringArray.

The output array will be of type vtkTypeUInt64Array.

To create an instance of class vtkStringToTimePoint, simply invoke its constructor as follows

```
obj = vtkStringToTimePoint
```

36.102.2 Methods

The class vtkStringToTimePoint has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStringToTimePoint class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStringToTimePoint = obj.NewInstance ()`
- `vtkStringToTimePoint = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutputArrayName (string )` - The name of the output array. If this is not specified, the name will be the same as the input array name with either " [to datetime]", " [to date]", or " [to time]" appended.
- `string = obj.GetOutputArrayName ()` - The name of the output array. If this is not specified, the name will be the same as the input array name with either " [to datetime]", " [to date]", or " [to time]" appended.

36.103 vtkTableToArray

36.103.1 Usage

Converts a vtkTable into a dense matrix. Use AddColumn() to designate one-to-many table columns that will become columns in the output matrix.

To create an instance of class vtkTableToArray, simply invoke its constructor as follows

```
obj = vtkTableToArray
```

36.103.2 Methods

The class vtkTableToArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableToArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableToArray = obj.NewInstance ()`
- `vtkTableToArray = obj.SafeDownCast (vtkObject o)`
- `obj.ClearColumns ()` - Specify the set of input table columns that will be mapped to columns in the output matrix.
- `obj.AddColumn (string name)` - Specify the set of input table columns that will be mapped to columns in the output matrix.

36.104 vtkTableToGraph

36.104.1 Usage

vtkTableToGraph converts a table to a graph using an auxilliary link graph. The link graph specifies how each row in the table should be converted to an edge, or a collection of edges. It also specifies which columns of the table should be considered part of the same domain, and which columns should be hidden.

A second, optional, table may be provided as the vertex table. This vertex table must have one or more domain columns whose values match values in the edge table. The linked column name is specified in the domain array in the link graph. The output graph will only contain vertices corresponding to a row in the vertex table. For heterogenous graphs, you may want to use vtkMergeTables to create a single vertex table.

The link graph contains the following arrays:

(1) The "column" array has the names of the columns to connect in each table row. This array is required.

(2) The optional "domain" array provides user-defined domain names for each column. Matching domains in multiple columns will merge vertices with the same value from those columns. By default, all columns are in the same domain. If a vertex table is supplied, the domain indicates the column in the vertex table that the edge table column associates with. If the user provides a vertex table but no domain names, the output will be an empty graph. Hidden columns do not need valid domain names.

(3) The optional "hidden" array is a bit array specifying whether the column should be hidden. The resulting graph will contain edges representing connections "through" the hidden column, but the vertices for that column will not be present. By default, no columns are hidden. Hiding a column in a particular domain hides all columns in that domain.

The output graph will contain three additional arrays in the vertex data. The "domain" column is a string array containing the domain of each vertex. The "label" column is a string version of the distinct value that, along with the domain, defines that vertex. The "ids" column also contains the distinguishing value, but as a vtkVariant holding the raw value instead of being converted to a string. The "ids" column is set as the vertex pedigree ID attribute.

To create an instance of class vtkTableToGraph, simply invoke its constructor as follows

```
obj = vtkTableToGraph
```

36.104.2 Methods

The class vtkTableToGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableToGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableToGraph = obj.NewInstance ()`
- `vtkTableToGraph = obj.SafeDownCast (vtkObject o)`
- `obj.AddLinkVertex (string column, string domain, int hidden)` - Add a vertex to the link graph. Specify the column name, the domain name for the column, and whether the column is hidden.
- `obj.ClearLinkVertices ()` - Clear the link graph vertices. This also clears all edges.
- `obj.AddLinkEdge (string column1, string column2)` - Add an edge to the link graph. Specify the names of the columns to link.
- `obj.ClearLinkEdges ()` - Clear the link graph edges. The graph vertices will remain.
- `vtkMutableDirectedGraph = obj.GetLinkGraph ()` - The graph describing how to link the columns in the table.

- `obj.SetLinkGraph (vtkMutableDirectedGraph g)` - The graph describing how to link the columns in the table.
- `obj.LinkColumnPath (vtkStringArray column, vtkStringArray domain, vtkBitArray hidden)` - Links the columns in a specific order. This creates a simple path as the link graph.
- `obj.SetDirected (bool )` - Specify the directedness of the output graph.
- `bool = obj.GetDirected ()` - Specify the directedness of the output graph.
- `obj.DirectedOn ()` - Specify the directedness of the output graph.
- `obj.DirectedOff ()` - Specify the directedness of the output graph.
- `long = obj.GetMTime ()` - Get the current modified time.
- `obj.SetVertexTableConnection (vtkAlgorithmOutput in)` - A convenience method for setting the vertex table input. This is mainly for the benefit of the VTK client/server layer, vanilla VTK code should use e.g:

  table_to_graph-¿SetInputConnection(1, vertex_table-¿output());

36.105 vtkTableToSparseArray

36.105.1 Usage

Converts a vtkTable into a sparse array. Use AddCoordinateColumn() to designate one-to-many table columns that contain coordinates for each array value, and SetValueColumn() to designate the table column that contains array values.

Thus, the number of dimensions in the output array will equal the number of calls to AddCoordinate-Column().

The coordinate columns will also be used to populate dimension labels in the output array.

.SECTION Thanks Developed by Timothy M. Shead (tshead@sandia.gov) at Sandia National Laboratories.

To create an instance of class vtkTableToSparseArray, simply invoke its constructor as follows

```
obj = vtkTableToSparseArray
```

36.105.2 Methods

The class vtkTableToSparseArray has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableToSparseArray class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableToSparseArray = obj.NewInstance ()`
- `vtkTableToSparseArray = obj.SafeDownCast (vtkObject o)`
- `obj.ClearCoordinateColumns ()` - Specify the set of input table columns that will be mapped to coordinates in the output sparse array.
- `obj.AddCoordinateColumn (string name)` - Specify the set of input table columns that will be mapped to coordinates in the output sparse array.

- `obj.SetValueColumn (string name)` - Specify the input table column that will be mapped to values in the output array.
- `string = obj.GetValueColumn ()` - Specify the input table column that will be mapped to values in the output array.

36.106 vtkTableToTreeFilter

36.106.1 Usage

vtkTableToTreeFilter is a filter for converting a vtkTable data structure into a vtkTree datastructure. Currently, this will convert the table into a star, with each row of the table as a child of a new root node. The columns of the table are passed as node fields of the tree.

To create an instance of class vtkTableToTreeFilter, simply invoke its constructor as follows

```
obj = vtkTableToTreeFilter
```

36.106.2 Methods

The class vtkTableToTreeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableToTreeFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableToTreeFilter = obj.NewInstance ()`
- `vtkTableToTreeFilter = obj.SafeDownCast (vtkObject o)`

36.107 vtkThresholdTable

36.107.1 Usage

vtkThresholdTable uses minimum and/or maximum values to threshold table rows based on the values in a particular column. The column to threshold is specified using SetInputArrayToProcess(0, ...).

To create an instance of class vtkThresholdTable, simply invoke its constructor as follows

```
obj = vtkThresholdTable
```

36.107.2 Methods

The class vtkThresholdTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkThresholdTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkThresholdTable = obj.NewInstance ()`
- `vtkThresholdTable = obj.SafeDownCast (vtkObject o)`

- `obj.SetMode (int )` - The mode of the threshold filter. Options are: ACCEPT_LESS_THAN (0) accepts rows with values ¡ MaxValue; ACCEPT_GREATER_THAN (1) accepts rows with values ¿ MinValue; ACCEPT_BETWEEN (2) accepts rows with values ¿ MinValue and ¡ MaxValue; ACCEPT_OUTSIDE (3) accepts rows with values ¡ MinValue or ¿ MaxValue.
- `int = obj.GetModeMinValue ()` - The mode of the threshold filter. Options are: ACCEPT_LESS_THAN (0) accepts rows with values ¡ MaxValue; ACCEPT_GREATER_THAN (1) accepts rows with values ¿ MinValue; ACCEPT_BETWEEN (2) accepts rows with values ¿ MinValue and ¡ MaxValue; ACCEPT_OUTSIDE (3) accepts rows with values ¡ MinValue or ¿ MaxValue.
- `int = obj.GetModeMaxValue ()` - The mode of the threshold filter. Options are: ACCEPT_LESS_THAN (0) accepts rows with values ¡ MaxValue; ACCEPT_GREATER_THAN (1) accepts rows with values ¿ MinValue; ACCEPT_BETWEEN (2) accepts rows with values ¿ MinValue and ¡ MaxValue; ACCEPT_OUTSIDE (3) accepts rows with values ¡ MinValue or ¿ MaxValue.
- `int = obj.GetMode ()` - The mode of the threshold filter. Options are: ACCEPT_LESS_THAN (0) accepts rows with values ¡ MaxValue; ACCEPT_GREATER_THAN (1) accepts rows with values ¿ MinValue; ACCEPT_BETWEEN (2) accepts rows with values ¿ MinValue and ¡ MaxValue; ACCEPT_OUTSIDE (3) accepts rows with values ¡ MinValue or ¿ MaxValue.
- `obj.SetMinValue (double v)` - The maximum value for the threshold as a double.
- `obj.SetMaxValue (double v)` - Criterion is rows whose scalars are between lower and upper thresholds (inclusive of the end values).
- `obj.ThresholdBetween (double lower, double upper)`

36.108 vtkTimePointToString

36.108.1 Usage

vtkTimePointToString is a filter for converting a timestamp array into string array using one of the formats defined in vtkTimePointUtility.h.

Use SetInputArrayToProcess to indicate the array to process. This array must be an unsigned 64-bit integer array for DATETIME formats, and may be either an unsigned 32-bit or unsigned 64-bit array for DATE and TIME formats.

If the new array name is not specified, the array name will be the old name appended by " [to string]".

To create an instance of class vtkTimePointToString, simply invoke its constructor as follows

```
obj = vtkTimePointToString
```

36.108.2 Methods

The class vtkTimePointToString has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTimePointToString class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTimePointToString = obj.NewInstance ()`
- `vtkTimePointToString = obj.SafeDownCast (vtkObject o)`
- `obj.SetISO8601Format (int )` - The format to use when converting the timestamp to a string.
- `int = obj.GetISO8601Format ()` - The format to use when converting the timestamp to a string.

- `obj.SetOutputArrayName (string )` - The name of the output array. If this is not specified, the name will be the input array name with " [to string]" appended to it.
- `string = obj.GetOutputArrayName ()` - The name of the output array. If this is not specified, the name will be the input array name with " [to string]" appended to it.

36.109 vtkTransferAttributes

36.109.1 Usage

The filter requires both a vtkGraph and vtkTree as input. The tree vertices must be a superset of the graph vertices. A common example is when the graph vertices correspond to the leaves of the tree, but the internal vertices of the tree represent groupings of graph vertices. The algorithm matches the vertices using the array "PedigreeId". The user may alternately set the DirectMapping flag to indicate that the two structures must have directly corresponding offsets (i.e. node i in the graph must correspond to node i in the tree).

.SECTION Thanks

To create an instance of class vtkTransferAttributes, simply invoke its constructor as follows

```
obj = vtkTransferAttributes
```

36.109.2 Methods

The class vtkTransferAttributes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransferAttributes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransferAttributes = obj.NewInstance ()`
- `vtkTransferAttributes = obj.SafeDownCast (vtkObject o)`
- `obj.SetDirectMapping (bool )` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `bool = obj.GetDirectMapping ()` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `obj.DirectMappingOn ()` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `obj.DirectMappingOff ()` - If on, uses direct mapping from tree to graph vertices. If off, both the graph and tree must contain PedigreeId arrays which are used to match graph and tree vertices. Default is off.
- `string = obj.GetSourceArrayName ()` - The field name to use for storing the source array.
- `obj.SetSourceArrayName (string )` - The field name to use for storing the source array.
- `string = obj.GetTargetArrayName ()` - The field name to use for storing the source array.
- `obj.SetTargetArrayName (string )` - The field name to use for storing the source array.

- `int = obj.GetSourceFieldType ()` - The source field type for accessing the source array. Valid values are those from enum vtkDataObject::FieldAssociations.
- `obj.SetSourceFieldType (int )` - The source field type for accessing the source array. Valid values are those from enum vtkDataObject::FieldAssociations.
- `int = obj.GetTargetFieldType ()` - The target field type for accessing the target array. Valid values are those from enum vtkDataObject::FieldAssociations.
- `obj.SetTargetFieldType (int )` - The target field type for accessing the target array. Valid values are those from enum vtkDataObject::FieldAssociations.
- `int = obj.FillInputPortInformation (int port, vtkInformation info)` - Set the input type of the algorithm to vtkGraph.

36.110 vtkTreeFieldAggregator

36.110.1 Usage

vtkTreeFieldAggregator may be used to assign sizes to all the vertices in the tree, based on the sizes of the leaves. The size of a vertex will equal the sum of the sizes of the child vertices. If you have a data array with values for all leaves, you may specify that array, and the values will be filled in for interior tree vertices. If you do not yet have an array, you may tell the filter to create a new array, assuming that the size of each leaf vertex is 1. You may optionally set a flag to first take the log of all leaf values before aggregating.

To create an instance of class vtkTreeFieldAggregator, simply invoke its constructor as follows

```
obj = vtkTreeFieldAggregator
```

36.110.2 Methods

The class vtkTreeFieldAggregator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeFieldAggregator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeFieldAggregator = obj.NewInstance ()`
- `vtkTreeFieldAggregator = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetField ()` - The field to aggregate. If this is a string array, the entries are converted to double. TODO: Remove this field and use the ArrayToProcess in vtkAlgorithm.
- `obj.SetField (string )` - The field to aggregate. If this is a string array, the entries are converted to double. TODO: Remove this field and use the ArrayToProcess in vtkAlgorithm.
- `double = obj.GetMinValue ()` - If the value of the vertex is less than MinValue then consider it's value to be minVal.
- `obj.SetMinValue (double )` - If the value of the vertex is less than MinValue then consider it's value to be minVal.
- `obj.SetLeafVertexUnitSize (bool )` - If set, the algorithm will assume a size of 1 for each leaf vertex.
- `bool = obj.GetLeafVertexUnitSize ()` - If set, the algorithm will assume a size of 1 for each leaf vertex.

- `obj.LeafVertexUnitSizeOn ()` - If set, the algorithm will assume a size of 1 for each leaf vertex.
- `obj.LeafVertexUnitSizeOff ()` - If set, the algorithm will assume a size of 1 for each leaf vertex.
- `obj.SetLogScale (bool )` - If set, the leaf values in the tree will be logarithmically scaled (base 10).
- `bool = obj.GetLogScale ()` - If set, the leaf values in the tree will be logarithmically scaled (base 10).
- `obj.LogScaleOn ()` - If set, the leaf values in the tree will be logarithmically scaled (base 10).
- `obj.LogScaleOff ()` - If set, the leaf values in the tree will be logarithmically scaled (base 10).

36.111 vtkTreeLayoutStrategy

36.111.1 Usage

Assigns points to the nodes of a tree in either a standard or radial layout. The standard layout places each level on a horizontal line, while the radial layout places each level on a concentric circle. You may specify the sweep angle of the tree which constrains the tree to be contained within a wedge. Also, you may indicate the log scale of the tree, which diminishes the length of arcs at lower levels of the tree. Values near zero give a large proportion of the space to the tree levels near the root, while values near one give nearly equal proportions of space to all tree levels.

The user may also specify an array to use to indicate the distance from the root, either vertically (for standard layout) or radially (for radial layout). You specify this with SetDistanceArrayName().

To create an instance of class vtkTreeLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkTreeLayoutStrategy
```

36.111.2 Methods

The class vtkTreeLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeLayoutStrategy = obj.NewInstance ()`
- `vtkTreeLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout ()` - Perform the tree layout.
- `obj.SetAngle (double )` - The sweep angle of the tree. For a standard tree layout, this should be between 0 and 180. For a radial tree layout, this can be between 0 and 360.
- `double = obj.GetAngleMinValue ()` - The sweep angle of the tree. For a standard tree layout, this should be between 0 and 180. For a radial tree layout, this can be between 0 and 360.
- `double = obj.GetAngleMaxValue ()` - The sweep angle of the tree. For a standard tree layout, this should be between 0 and 180. For a radial tree layout, this can be between 0 and 360.
- `double = obj.GetAngle ()` - The sweep angle of the tree. For a standard tree layout, this should be between 0 and 180. For a radial tree layout, this can be between 0 and 360.
- `obj.SetRadial (bool )` - If set, the tree is laid out with levels on concentric circles around the root. If unset (default), the tree is laid out with levels on horizontal lines.

- `bool = obj.GetRadial ()` - If set, the tree is laid out with levels on concentric circles around the root. If unset (default), the tree is laid out with levels on horizontal lines.
- `obj.RadialOn ()` - If set, the tree is laid out with levels on concentric circles around the root. If unset (default), the tree is laid out with levels on horizontal lines.
- `obj.RadialOff ()` - If set, the tree is laid out with levels on concentric circles around the root. If unset (default), the tree is laid out with levels on horizontal lines.
- `obj.SetLogSpacingValue (double )` - The spacing of tree levels. Levels near zero give more space to levels near the root, while levels near one (the default) create evenly-spaced levels. Levels above one give more space to levels near the leaves.
- `double = obj.GetLogSpacingValue ()` - The spacing of tree levels. Levels near zero give more space to levels near the root, while levels near one (the default) create evenly-spaced levels. Levels above one give more space to levels near the leaves.
- `obj.SetLeafSpacing (double )` - The spacing of leaves. Levels near one evenly space leaves with no gaps between subtrees. Levels near zero creates large gaps between subtrees.
- `double = obj.GetLeafSpacingMinValue ()` - The spacing of leaves. Levels near one evenly space leaves with no gaps between subtrees. Levels near zero creates large gaps between subtrees.
- `double = obj.GetLeafSpacingMaxValue ()` - The spacing of leaves. Levels near one evenly space leaves with no gaps between subtrees. Levels near zero creates large gaps between subtrees.
- `double = obj.GetLeafSpacing ()` - The spacing of leaves. Levels near one evenly space leaves with no gaps between subtrees. Levels near zero creates large gaps between subtrees.
- `obj.SetDistanceArrayName (string )` - Get/Set the array to use to determine the distance from the root.
- `string = obj.GetDistanceArrayName ()` - Get/Set the array to use to determine the distance from the root.

36.112 vtkTreeLevelsFilter

36.112.1 Usage

The filter currently add two arrays to the incoming vtkTree datastructure. 1) "levels" this is the distance from the root of the vertex. Root = 0 and you add 1 for each level down from the root 2) "leaf" this array simply indicates whether the vertex is a leaf or not

.SECTION Thanks Thanks to Brian Wylie from Sandia National Laboratories for creating this class.

To create an instance of class vtkTreeLevelsFilter, simply invoke its constructor as follows

```
obj = vtkTreeLevelsFilter
```

36.112.2 Methods

The class vtkTreeLevelsFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeLevelsFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeLevelsFilter = obj.NewInstance ()`
- `vtkTreeLevelsFilter = obj.SafeDownCast (vtkObject o)`

36.113 vtkTreeMapLayout

36.113.1 Usage

vtkTreeMapLayout assigns rectangular regions to each vertex in the tree, creating a tree map. The data is added as a data array with four components per tuple representing the location and size of the rectangle using the format (Xmin, Xmax, Ymin, Ymax).

This algorithm relies on a helper class to perform the actual layout. This helper class is a subclass of vtkTreeMapLayoutStrategy.

.SECTION Thanks Thanks to Brian Wylie and Ken Moreland from Sandia National Laboratories for help developing this class.

Tree map concept comes from: Shneiderman, B. 1992. Tree visualization with tree-maps: 2-d space-filling approach. ACM Trans. Graph. 11, 1 (Jan. 1992), 92-99.

To create an instance of class vtkTreeMapLayout, simply invoke its constructor as follows

```
obj = vtkTreeMapLayout
```

36.113.2 Methods

The class vtkTreeMapLayout has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeMapLayout class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeMapLayout = obj.NewInstance ()`
- `vtkTreeMapLayout = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetRectanglesFieldName ()` - The field name to use for storing the rectangles for each vertex. The rectangles are stored in a quadruple float array (minX, maxX, minY, maxY).
- `obj.SetRectanglesFieldName (string )` - The field name to use for storing the rectangles for each vertex. The rectangles are stored in a quadruple float array (minX, maxX, minY, maxY).
- `obj.SetSizeArrayName (string name)` - The strategy to use when laying out the tree map.
- `vtkTreeMapLayoutStrategy = obj.GetLayoutStrategy ()` - The strategy to use when laying out the tree map.
- `obj.SetLayoutStrategy (vtkTreeMapLayoutStrategy strategy)` - The strategy to use when laying out the tree map.
- `vtkIdType = obj.FindVertex (float pnt[2], float binfo)` - Returns the vertex id that contains pnt (or -1 if no one contains it)
- `obj.GetBoundingBox (vtkIdType id, float binfo)` - Return the min and max 2D points of the vertex's bounding box
- `long = obj.GetMTime ()` - Get the modification time of the layout algorithm.

36.114 vtkTreeMapLayoutStrategy

36.114.1 Usage

All subclasses of this class perform a tree map layout on a tree. This involves assigning a rectangular region to each vertex in the tree, and placing that information in a data array with four components per tuple representing (Xmin, Xmax, Ymin, Ymax).

Instances of subclasses of this class may be assigned as the layout strategy to vtkTreeMapLayout

.SECTION Thanks Thanks to Brian Wylie and Ken Moreland from Sandia National Laboratories for help developing this class.

To create an instance of class vtkTreeMapLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkTreeMapLayoutStrategy
```

36.114.2 Methods

The class vtkTreeMapLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeMapLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeMapLayoutStrategy = obj.NewInstance ()`
- `vtkTreeMapLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.FindVertex (vtkTree tree, vtkDataArray areaArray, float pnt[2])` - Find the vertex at a certain location, or -1 if none found.

36.115 vtkTreeMapToPolyData

36.115.1 Usage

This algorithm requires that the vtkTreeMapLayout filter has already applied to the data in order to create the quadruple array (min x, max x, min y, max y) of bounds for each vertex of the tree.

To create an instance of class vtkTreeMapToPolyData, simply invoke its constructor as follows

```
obj = vtkTreeMapToPolyData
```

36.115.2 Methods

The class vtkTreeMapToPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeMapToPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeMapToPolyData = obj.NewInstance ()`
- `vtkTreeMapToPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetRectanglesArrayName (string name)` - The field containing the level of each tree node. This can be added using vtkTreeLevelsFilter before this filter. If this is not present, the filter simply calls tree-¿GetLevel(v) for each vertex, which will produce the same result, but may not be as efficient.

- `obj.SetLevelArrayName (string name)` - The spacing along the z-axis between tree map levels.
- `double = obj.GetLevelDeltaZ ()` - The spacing along the z-axis between tree map levels.
- `obj.SetLevelDeltaZ (double )` - The spacing along the z-axis between tree map levels.
- `bool = obj.GetAddNormals ()` - The spacing along the z-axis between tree map levels.
- `obj.SetAddNormals (bool )` - The spacing along the z-axis between tree map levels.
- `int = obj.FillInputPortInformation (int port, vtkInformation info)`

36.116 vtkTreeOrbitLayoutStrategy

36.116.1 Usage

Assigns points to the nodes of a tree to an orbital layout. Each parent is orbited by its children, recursively.
To create an instance of class vtkTreeOrbitLayoutStrategy, simply invoke its constructor as follows

```
obj = vtkTreeOrbitLayoutStrategy
```

36.116.2 Methods

The class vtkTreeOrbitLayoutStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeOrbitLayoutStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeOrbitLayoutStrategy = obj.NewInstance ()`
- `vtkTreeOrbitLayoutStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.Layout ()` - Perform the orbital layout.
- `obj.SetLogSpacingValue (double )` - The spacing of orbital levels. Levels near zero give more space to levels near the root, while levels near one (the default) create evenly-spaced levels. Levels above one give more space to levels near the leaves.
- `double = obj.GetLogSpacingValue ()` - The spacing of orbital levels. Levels near zero give more space to levels near the root, while levels near one (the default) create evenly-spaced levels. Levels above one give more space to levels near the leaves.
- `obj.SetLeafSpacing (double )` - The spacing of leaves. Levels near one evenly space leaves with no gaps between subtrees. Levels near zero creates large gaps between subtrees.
- `double = obj.GetLeafSpacingMinValue ()` - The spacing of leaves. Levels near one evenly space leaves with no gaps between subtrees. Levels near zero creates large gaps between subtrees.
- `double = obj.GetLeafSpacingMaxValue ()` - The spacing of leaves. Levels near one evenly space leaves with no gaps between subtrees. Levels near zero creates large gaps between subtrees.
- `double = obj.GetLeafSpacing ()` - The spacing of leaves. Levels near one evenly space leaves with no gaps between subtrees. Levels near zero creates large gaps between subtrees.
- `obj.SetChildRadiusFactor (double )` - This is a magic number right now. Controls the radius of the child layout, all of this should be fixed at some point with a more logical layout. Defaults to .5 :)

- `double = obj.GetChildRadiusFactor ()` - This is a magic number right now. Controls the radius of the child layout, all of this should be fixed at some point with a more logical layout. Defaults to .5 :)

36.117 vtkTreeRingToPolyData

36.117.1 Usage

This algorithm requires that the vtkTreeRingLayout filter has already been applied to the data in order to create the quadruple array (start angle, end angle, inner radius, outer radius) of bounds for each vertex of the tree.

To create an instance of class vtkTreeRingToPolyData, simply invoke its constructor as follows

```
obj = vtkTreeRingToPolyData
```

36.117.2 Methods

The class vtkTreeRingToPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeRingToPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeRingToPolyData = obj.NewInstance ()`
- `vtkTreeRingToPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetSectorsArrayName (string name)` - Define a shrink percentage for each of the sectors.
- `obj.SetShrinkPercentage (double )` - Define a shrink percentage for each of the sectors.
- `double = obj.GetShrinkPercentage ()` - Define a shrink percentage for each of the sectors.
- `int = obj.FillInputPortInformation (int port, vtkInformation info)`

36.118 vtkTulipReader

36.118.1 Usage

vtkTulipReader reads in files in the Tulip format. Definition of the Tulip file format can be found online at: http://tulip.labri.fr/tlpformat.php An example is the following ¡code¿ (nodes 0 1 2 3 4 5 6 7 8 9) (edge 0 0 1) (edge 1 1 2) (edge 2 2 3) (edge 3 3 4) (edge 4 4 5) (edge 5 5 6) (edge 6 6 7) (edge 7 7 8) (edge 8 8 9) (edge 9 9 0) (edge 10 0 5) (edge 11 2 7) (edge 12 4 9) ¡/code¿ where "nodes" defines all the nodes ids in the graph, and "edge" is a triple of edge id, source vertex id, and target vertex id. The graph is read in as undirected graph. NOTE: This currently only supports reading connectivity information. Display information is discarded.

To create an instance of class vtkTulipReader, simply invoke its constructor as follows

```
obj = vtkTulipReader
```

36.118.2 Methods

The class vtkTulipReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTulipReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTulipReader = obj.NewInstance ()`
- `vtkTulipReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileName ()` - The Tulip file name.
- `obj.SetFileName (string )` - The Tulip file name.

36.119 vtkUnivariateStatisticsAlgorithm

36.119.1 Usage

This class specializes statistics algorithms to the univariate case, where a number of columns of interest can be selected in the input data set. This is done by the means of the following functions:

ResetColumns() - reset the list of columns of interest. Add/RemoveColum(namCol) - try to add/remove column with name namCol to/from the list. SetColumnStatus (namCol, status) - mostly for UI wrapping purposes, try to add/remove (depending on status) namCol from the list of columns of interest. The verb "try" is used in the sense that neither attempting to repeat an existing entry nor to remove a non-existent entry will work.

.SECTION Thanks Thanks to Philippe Pebay and David Thompson from Sandia National Laboratories for implementing this class.

To create an instance of class vtkUnivariateStatisticsAlgorithm, simply invoke its constructor as follows

```
obj = vtkUnivariateStatisticsAlgorithm
```

36.119.2 Methods

The class vtkUnivariateStatisticsAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnivariateStatisticsAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnivariateStatisticsAlgorithm = obj.NewInstance ()`
- `vtkUnivariateStatisticsAlgorithm = obj.SafeDownCast (vtkObject o)`
- `obj.AddColumn (string namCol)` - Convenience method to create a request with a single column name namCol in a single call; this is the preferred method to select columns, ensuring selection consistency (a single column per request). Warning: no name checking is performed on namCol; it is the user's responsibility to use valid column names.
- `int = obj.RequestSelectedColumns ()` - Use the current column status values to produce a new request for statistics to be produced when RequestData() is called. Unlike the superclass implementation, this version adds a new request for each selected column instead of a single request containing all the columns.

36.120 vtkVertexDegree

36.120.1 Usage

Adds an attribute array with the degree of each vertex. By default the name of the array will be "VertexDegree", but that can be changed by calling SetOutputArrayName("foo");

To create an instance of class vtkVertexDegree, simply invoke its constructor as follows

```
obj = vtkVertexDegree
```

36.120.2 Methods

The class vtkVertexDegree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVertexDegree class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVertexDegree = obj.NewInstance ()`
- `vtkVertexDegree = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutputArrayName (string )` - Set the output array name. If no output array name is set then the name 'VertexDegree' is used.

36.121 vtkXMLTreeReader

36.121.1 Usage

vtkXMLTreeReader parses an XML file and uses the nesting structure of the XML tags to generate a tree. Node attributes are assigned to node arrays, and the special arrays .tagname and .chardata contain the tag type and the text internal to the tag, respectively. The arrays are of type vtkStringArray. There is an array for each attribute type in the XML file, even if it appears in only one tag. If an attribute is missing from a tag, its value is the empty string.

If MaskArrays is on (the default is off), the filter will additionally make bit arrays whose names are prepended with ".valid." which are 1 if the element contains that attribute, and 0 otherwise.

For example, the XML file containing the text:

```
\&lt;node name=''jeff'' age=''26''\&gt;
  this is text in jeff's node
  \&lt;node name=''joe''\&gt;
    \&lt;node name=''al'' initials=''amb'' other=''something''/\&gt;
    \&lt;node name=''dave'' age=''30''/\&gt;
  \&lt;/node\&gt;
  \&lt;node name=''lisa''\&gt;this is text in lisa's node\&lt;/node\&gt;
  \&lt;node name=''darlene'' age=''29''/\&gt;
\&lt;/node\&gt;
```

would be parsed into a tree with the following node IDs and structure:

```
0 (jeff) - children: 1 (joe), 4 (lisa), 5 (darlene)
1 (joe)  - children: 2 (al), 3 (dave)
2 (al)
```

```
3 (dave)
4 (lisa)
5 (darlene)
```

and the node data arrays would be as follows:

```
name      initials  other     age       .tagname  .chardata
-------------------------------------------------------------------------------------------
jeff      (empty)   (empty)   26        node      ''  this is text in jeff's node\\n  \\n  \\n  \\n''
joe       (empty)   (empty)   (empty)   node      ''\\n    \\n    \\n  ''
al        amb       something (empty)   node      (empty)
dave      (empty)   (empty)   30        node      (empty)
lisa      (empty)   (empty)   (empty)   node      ''this is text in lisa's node''
darlene   (empty)   (empty)   29        node      (empty)
```

There would also be the following bit arrays if MaskArrays is on:

```
.valid.name   .valid.initials   .valid.other   .valid.age
-----------------------------------------------------------
1             0                 0              1
1             0                 0              0
1             1                 1              0
1             0                 0              1
1             0                 0              0
1             0                 0              1
```

To create an instance of class vtkXMLTreeReader, simply invoke its constructor as follows

```
obj = vtkXMLTreeReader
```

36.121.2 Methods

The class vtkXMLTreeReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLTreeReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLTreeReader = obj.NewInstance ()`
- `vtkXMLTreeReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileName ()` - If set, reads in the XML file specified.
- `obj.SetFileName (string )` - If set, reads in the XML file specified.
- `string = obj.GetXMLString ()` - If set, and FileName is not set, reads in the XML string.
- `obj.SetXMLString (string )` - If set, and FileName is not set, reads in the XML string.
- `string = obj.GetEdgePedigreeIdArrayName ()` - The name of the edge pedigree ids. Default is "edge id".

- `obj.SetEdgePedigreeIdArrayName (string )` - The name of the edge pedigree ids. Default is "edge id".
- `string = obj.GetVertexPedigreeIdArrayName ()` - The name of the vertex pedigree ids. Default is "vertex id".
- `obj.SetVertexPedigreeIdArrayName (string )` - The name of the vertex pedigree ids. Default is "vertex id".
- `obj.SetGenerateEdgePedigreeIds (bool )` - Set whether to use an property from the XML file as pedigree ids (off), or generate a new array with integer values starting at zero (on). Default is on.
- `bool = obj.GetGenerateEdgePedigreeIds ()` - Set whether to use an property from the XML file as pedigree ids (off), or generate a new array with integer values starting at zero (on). Default is on.
- `obj.GenerateEdgePedigreeIdsOn ()` - Set whether to use an property from the XML file as pedigree ids (off), or generate a new array with integer values starting at zero (on). Default is on.
- `obj.GenerateEdgePedigreeIdsOff ()` - Set whether to use an property from the XML file as pedigree ids (off), or generate a new array with integer values starting at zero (on). Default is on.
- `obj.SetGenerateVertexPedigreeIds (bool )` - Set whether to use an property from the XML file as pedigree ids (off), or generate a new array with integer values starting at zero (on). Default is on.
- `bool = obj.GetGenerateVertexPedigreeIds ()` - Set whether to use an property from the XML file as pedigree ids (off), or generate a new array with integer values starting at zero (on). Default is on.
- `obj.GenerateVertexPedigreeIdsOn ()` - Set whether to use an property from the XML file as pedigree ids (off), or generate a new array with integer values starting at zero (on). Default is on.
- `obj.GenerateVertexPedigreeIdsOff ()` - Set whether to use an property from the XML file as pedigree ids (off), or generate a new array with integer values starting at zero (on). Default is on.
- `bool = obj.GetMaskArrays ()` - If on, makes bit arrays for each attribute with name .valid.attribute_name for each attribute. Default is off.
- `obj.SetMaskArrays (bool )` - If on, makes bit arrays for each attribute with name .valid.attribute_name for each attribute. Default is off.
- `obj.MaskArraysOn ()` - If on, makes bit arrays for each attribute with name .valid.attribute_name for each attribute. Default is off.
- `obj.MaskArraysOff ()` - If on, makes bit arrays for each attribute with name .valid.attribute_name for each attribute. Default is off.
- `bool = obj.GetReadCharData ()` - If on, stores the XML character data (i.e. textual data between tags) into an array named CharDataField, otherwise this field is skipped. Default is off.
- `obj.SetReadCharData (bool )` - If on, stores the XML character data (i.e. textual data between tags) into an array named CharDataField, otherwise this field is skipped. Default is off.
- `obj.ReadCharDataOn ()` - If on, stores the XML character data (i.e. textual data between tags) into an array named CharDataField, otherwise this field is skipped. Default is off.
- `obj.ReadCharDataOff ()` - If on, stores the XML character data (i.e. textual data between tags) into an array named CharDataField, otherwise this field is skipped. Default is off.
- `bool = obj.GetReadTagName ()` - If on, stores the XML tag name data in a field called .tagname otherwise this field is skipped. Default is on.
- `obj.SetReadTagName (bool )` - If on, stores the XML tag name data in a field called .tagname otherwise this field is skipped. Default is on.

- `obj.ReadTagNameOn ()` - If on, stores the XML tag name data in a field called .tagname otherwise this field is skipped. Default is on.
- `obj.ReadTagNameOff ()` - If on, stores the XML tag name data in a field called .tagname otherwise this field is skipped. Default is on.

Chapter 37

Visualization Toolkit IO Classes

37.1 vtkAbstractParticleWriter

37.1.1 Usage

vtkAbstractParticleWriter is an abstract class which is used by vtkTemporalStreamTracer to write particles out during simulations. This class is abstract and provides a TimeStep and FileName. Subclasses of this should provide the necessary IO.

To create an instance of class vtkAbstractParticleWriter, simply invoke its constructor as follows

```
obj = vtkAbstractParticleWriter
```

37.1.2 Methods

The class vtkAbstractParticleWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractParticleWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractParticleWriter = obj.NewInstance ()`
- `vtkAbstractParticleWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetTimeStep (int )` - Set/get the TimeStep that is being written
- `int = obj.GetTimeStep ()` - Set/get the TimeStep that is being written
- `obj.SetTimeValue (double )` - Before writing the current data out, set the TimeValue (optional) The TimeValue is a float/double value that corresonds to the real time of the data, it may not be regular, whereas the TimeSteps are simple increments.
- `double = obj.GetTimeValue ()` - Before writing the current data out, set the TimeValue (optional) The TimeValue is a float/double value that corresonds to the real time of the data, it may not be regular, whereas the TimeSteps are simple increments.
- `obj.SetFileName (string )` - Set/get the FileName that is being written to
- `string = obj.GetFileName ()` - Set/get the FileName that is being written to
- `obj.SetCollectiveIO (int )` - When running in parallel, this writer may be capable of Collective IO operations (HDF5). By default, this is off.

- `int = obj.GetCollectiveIO ()` - When running in parallel, this writer may be capable of Collective IO operations (HDF5). By default, this is off.
- `obj.SetWriteModeToCollective ()` - When running in parallel, this writer may be capable of Collective IO operations (HDF5). By default, this is off.
- `obj.SetWriteModeToIndependent ()` - When running in parallel, this writer may be capable of Collective IO operations (HDF5). By default, this is off.
- `obj.CloseFile ()` - Close the file after a write. This is optional but may protect against data loss in between steps

37.2 vtkArrayReader

37.2.1 Usage

Reads sparse and dense vtkArray data written with vtkArrayWriter.

Outputs: Output port 0: vtkArrayData containing a dense or sparse array.

To create an instance of class vtkArrayReader, simply invoke its constructor as follows

```
obj = vtkArrayReader
```

37.2.2 Methods

The class vtkArrayReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArrayReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArrayReader = obj.NewInstance ()`
- `vtkArrayReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileName ()` - Set the filesystem location from which data will be read.
- `obj.SetFileName (string )` - Set the filesystem location from which data will be read.

37.3 vtkArrayWriter

37.3.1 Usage

vtkArrayWriter serializes sparse and dense array data using a text-based format that is human-readable and easily parsed (default option). The WriteBinary array option can be set to true in the Write method, which will serialize the sparse and dense array data using a binary format that is optimized for rapid throughput.

Inputs: Input port 0: (required) vtkArrayData object containing a sparse or dense array.

Output Format: The first line of output will contain the array type (sparse or dense) and the type of values stored in the array (double, integer, string, etc).

The second line of output will contain the array extents along each dimension of the array, followed by the number of non-null values stored in the array.

For sparse arrays, each subsequent line of output will contain the coordinates and value for each non-null value stored in the array.

For dense arrays, each subsequent line of output will contain one value from the array, stored in the same order as that used by vtkArrayCoordinateIterator.

To create an instance of class vtkArrayWriter, simply invoke its constructor as follows

```
obj = vtkArrayWriter
```

37.3.2 Methods

The class vtkArrayWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkArrayWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkArrayWriter = obj.NewInstance ()`
- `vtkArrayWriter = obj.SafeDownCast (vtkObject o)`
- `bool = obj.Write (vtkStdString \&file\_name, bool WriteBinaryfalse)` - Write input port 0 data to a file.

37.4 vtkAVSucdReader

37.4.1 Usage

vtkAVSucdReader creates an unstructured grid dataset. It reads binary or ASCII files stored in UCD format, with optional data stored at the nodes or at the cells of the model. A cell-based fielddata stores the material id. The class can automatically detect the endian-ness of the binary files.

To create an instance of class vtkAVSucdReader, simply invoke its constructor as follows

```
obj = vtkAVSucdReader
```

37.4.2 Methods

The class vtkAVSucdReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAVSucdReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAVSucdReader = obj.NewInstance ()`
- `vtkAVSucdReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name of AVS UCD datafile to read
- `string = obj.GetFileName ()` - Specify file name of AVS UCD datafile to read
- `obj.SetBinaryFile (int )` - Is the file to be read written in binary format (as opposed to ascii).
- `int = obj.GetBinaryFile ()` - Is the file to be read written in binary format (as opposed to ascii).
- `obj.BinaryFileOn ()` - Is the file to be read written in binary format (as opposed to ascii).
- `obj.BinaryFileOff ()` - Is the file to be read written in binary format (as opposed to ascii).
- `int = obj.GetNumberOfCells ()` - Get the total number of cells.

- `int = obj.GetNumberOfNodes ()` - Get the total number of nodes.
- `int = obj.GetNumberOfNodeFields ()` - Get the number of data fields at the nodes.
- `int = obj.GetNumberOfCellFields ()` - Get the number of data fields at the cell centers.
- `int = obj.GetNumberOfFields ()` - Get the number of data fields for the model. Unused because VTK has no methods for it.
- `int = obj.GetNumberOfNodeComponents ()` - Get the number of data components at the nodes and cells.
- `int = obj.GetNumberOfCellComponents ()` - Get the number of data components at the nodes and cells.
- `obj.SetByteOrderToBigEndian ()` - Set/Get the endian-ness of the binary file.
- `obj.SetByteOrderToLittleEndian ()` - Set/Get the endian-ness of the binary file.
- `string = obj.GetByteOrderAsString ()` - Set/Get the endian-ness of the binary file.
- `obj.SetByteOrder (int )`
- `int = obj.GetByteOrder ()`
- `int = obj.GetNumberOfPointArrays ()` - The following methods allow selective reading of solutions fields. by default, ALL data fields are the nodes and cells are read, but this can be modified.
- `int = obj.GetNumberOfCellArrays ()` - The following methods allow selective reading of solutions fields. by default, ALL data fields are the nodes and cells are read, but this can be modified.
- `string = obj.GetPointArrayName (int index)` - The following methods allow selective reading of solutions fields. by default, ALL data fields are the nodes and cells are read, but this can be modified.
- `string = obj.GetCellArrayName (int index)` - The following methods allow selective reading of solutions fields. by default, ALL data fields are the nodes and cells are read, but this can be modified.
- `int = obj.GetPointArrayStatus (string name)` - The following methods allow selective reading of solutions fields. by default, ALL data fields are the nodes and cells are read, but this can be modified.
- `int = obj.GetCellArrayStatus (string name)` - The following methods allow selective reading of solutions fields. by default, ALL data fields are the nodes and cells are read, but this can be modified.
- `obj.SetPointArrayStatus (string name, int status)` - The following methods allow selective reading of solutions fields. by default, ALL data fields are the nodes and cells are read, but this can be modified.
- `obj.SetCellArrayStatus (string name, int status)` - The following methods allow selective reading of solutions fields. by default, ALL data fields are the nodes and cells are read, but this can be modified.
- `obj.DisableAllCellArrays ()`
- `obj.EnableAllCellArrays ()`
- `obj.DisableAllPointArrays ()`
- `obj.EnableAllPointArrays ()`
- `obj.GetCellDataRange (int cellComp, int index, float min, float max)`
- `obj.GetNodeDataRange (int nodeComp, int index, float min, float max)`

37.5 vtkBase64InputStream

37.5.1 Usage

vtkBase64InputStream implements base64 decoding with the vtkInputStream interface.

To create an instance of class vtkBase64InputStream, simply invoke its constructor as follows

```
obj = vtkBase64InputStream
```

37.5.2 Methods

The class vtkBase64InputStream has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBase64InputStream class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBase64InputStream = obj.NewInstance ()`
- `vtkBase64InputStream = obj.SafeDownCast (vtkObject o)`
- `obj.StartReading ()` - Called after the stream position has been set by the caller, but before any Seek or Read calls. The stream position should not be adjusted by the caller until after an EndReading call.
- `int = obj.Seek (long offset)` - Seek to the given offset in the input data. Returns 1 for success, 0 for failure.
- `long = obj.Read (string data, long length)` - Read input data of the given length. Returns amount actually read.
- `obj.EndReading ()` - Called after all desired calls to Seek and Read have been made. After this call, the caller is free to change the position of the stream. Additional reads should not be done until after another call to StartReading.

37.6 vtkBase64OutputStream

37.6.1 Usage

vtkBase64OutputStream implements base64 encoding with the vtkOutputStream interface.

To create an instance of class vtkBase64OutputStream, simply invoke its constructor as follows

```
obj = vtkBase64OutputStream
```

37.6.2 Methods

The class vtkBase64OutputStream has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBase64OutputStream class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBase64OutputStream = obj.NewInstance ()`

- `vtkBase64OutputStream = obj.SafeDownCast (vtkObject o)`
- `int = obj.StartWriting ()` - Called after the stream position has been set by the caller, but before any Write calls. The stream position should not be adjusted by the caller until after an EndWriting call.
- `int = obj.Write (string data, long length)` - Write output data of the given length.
- `int = obj.EndWriting ()` - Called after all desired calls to Write have been made. After this call, the caller is free to change the position of the stream. Additional writes should not be done until after another call to StartWriting.

37.7 vtkBase64Utilities

37.7.1 Usage

vtkBase64Utilities implements base64 encoding and decoding.

To create an instance of class vtkBase64Utilities, simply invoke its constructor as follows

```
obj = vtkBase64Utilities
```

37.7.2 Methods

The class vtkBase64Utilities has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBase64Utilities class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBase64Utilities = obj.NewInstance ()`
- `vtkBase64Utilities = obj.SafeDownCast (vtkObject o)`

37.8 vtkBMPReader

37.8.1 Usage

vtkBMPReader is a source object that reads Windows BMP files. This includes indexed and 24bit bitmaps Usually, all BMPs are converted to 24bit RGB, but BMPs may be output as 8bit images with a LookupTable if the Allow8BitBMP flag is set.

BMPReader creates structured point datasets. The dimension of the dataset depends upon the number of files read. Reading a single file results in a 2D image, while reading more than one file results in a 3D volume.

To read a volume, files must be of the form "FileName.¡number¿" (e.g., foo.bmp.0, foo.bmp.1, ...). You must also specify the image range. This range specifies the beginning and ending files to read (range can be any pair of non-negative numbers).

The default behavior is to read a single file. In this case, the form of the file is simply "FileName" (e.g., foo.bmp).

To create an instance of class vtkBMPReader, simply invoke its constructor as follows

```
obj = vtkBMPReader
```

37.8.2 Methods

The class vtkBMPReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBMPReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBMPReader = obj.NewInstance ()`
- `vtkBMPReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetDepth ()` - Returns the depth of the BMP, either 8 or 24.
- `int = obj.CanReadFile (string fname)` - Is the given file a BMP file?
- `string = obj.GetFileExtensions ()` - Return a descriptive name for the file format that might be useful in a GUI.
- `string = obj.GetDescriptiveName ()` - If this flag is set and the BMP reader encounters an 8bit file, the data will be kept as unsigned chars and a lookuptable will be exported
- `obj.SetAllow8BitBMP (int )` - If this flag is set and the BMP reader encounters an 8bit file, the data will be kept as unsigned chars and a lookuptable will be exported
- `int = obj.GetAllow8BitBMP ()` - If this flag is set and the BMP reader encounters an 8bit file, the data will be kept as unsigned chars and a lookuptable will be exported
- `obj.Allow8BitBMPOn ()` - If this flag is set and the BMP reader encounters an 8bit file, the data will be kept as unsigned chars and a lookuptable will be exported
- `obj.Allow8BitBMPOff ()` - If this flag is set and the BMP reader encounters an 8bit file, the data will be kept as unsigned chars and a lookuptable will be exported
- `vtkLookupTable = obj.GetLookupTable ()`

37.9 vtkBMPWriter

37.9.1 Usage

vtkBMPWriter writes BMP files. The data type of the file is unsigned char regardless of the input type.

To create an instance of class vtkBMPWriter, simply invoke its constructor as follows

```
obj = vtkBMPWriter
```

37.9.2 Methods

The class vtkBMPWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBMPWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBMPWriter = obj.NewInstance ()`
- `vtkBMPWriter = obj.SafeDownCast (vtkObject o)`

37.10 vtkBYUReader

37.10.1 Usage

vtkBYUReader is a source object that reads MOVIE.BYU polygon files. These files consist of a geometry file (.g), a scalar file (.s), a displacement or vector file (.d), and a 2D texture coordinate file (.t).

To create an instance of class vtkBYUReader, simply invoke its constructor as follows

```
obj = vtkBYUReader
```

37.10.2 Methods

The class vtkBYUReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBYUReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBYUReader = obj.NewInstance ()`
- `vtkBYUReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetGeometryFileName (string )` - Specify name of geometry FileName.
- `string = obj.GetGeometryFileName ()` - Specify name of geometry FileName.
- `obj.SetFileName (string f)` - Specify name of geometry FileName (alias).
- `string = obj.GetFileName ()` - Specify name of displacement FileName.
- `obj.SetDisplacementFileName (string )` - Specify name of displacement FileName.
- `string = obj.GetDisplacementFileName ()` - Specify name of displacement FileName.
- `obj.SetScalarFileName (string )` - Specify name of scalar FileName.
- `string = obj.GetScalarFileName ()` - Specify name of scalar FileName.
- `obj.SetTextureFileName (string )` - Specify name of texture coordinates FileName.
- `string = obj.GetTextureFileName ()` - Specify name of texture coordinates FileName.
- `obj.SetReadDisplacement (int )` - Turn on/off the reading of the displacement file.
- `int = obj.GetReadDisplacement ()` - Turn on/off the reading of the displacement file.
- `obj.ReadDisplacementOn ()` - Turn on/off the reading of the displacement file.
- `obj.ReadDisplacementOff ()` - Turn on/off the reading of the displacement file.
- `obj.SetReadScalar (int )` - Turn on/off the reading of the scalar file.
- `int = obj.GetReadScalar ()` - Turn on/off the reading of the scalar file.
- `obj.ReadScalarOn ()` - Turn on/off the reading of the scalar file.
- `obj.ReadScalarOff ()` - Turn on/off the reading of the scalar file.
- `obj.SetReadTexture (int )` - Turn on/off the reading of the texture coordinate file. Specify name of geometry FileName.

- `int = obj.GetReadTexture ()` - Turn on/off the reading of the texture coordinate file. Specify name of geometry FileName.
- `obj.ReadTextureOn ()` - Turn on/off the reading of the texture coordinate file. Specify name of geometry FileName.
- `obj.ReadTextureOff ()` - Turn on/off the reading of the texture coordinate file. Specify name of geometry FileName.
- `obj.SetPartNumber (int )` - Set/Get the part number to be read.
- `int = obj.GetPartNumberMinValue ()` - Set/Get the part number to be read.
- `int = obj.GetPartNumberMaxValue ()` - Set/Get the part number to be read.
- `int = obj.GetPartNumber ()` - Set/Get the part number to be read.

37.11 vtkBYUWriter

37.11.1 Usage

vtkBYUWriter writes MOVIE.BYU polygonal files. These files consist of a geometry file (.g), a scalar file (.s), a displacement or vector file (.d), and a 2D texture coordinate file (.t). These files must be specified to the object, the appropriate boolean variables must be true, and data must be available from the input for the files to be written. WARNING: this writer does not currently write triangle strips. Use vtkTriangleFilter to convert strips to triangles.

To create an instance of class vtkBYUWriter, simply invoke its constructor as follows

```
obj = vtkBYUWriter
```

37.11.2 Methods

The class vtkBYUWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBYUWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBYUWriter = obj.NewInstance ()`
- `vtkBYUWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetGeometryFileName (string )` - Specify the name of the geometry file to write.
- `string = obj.GetGeometryFileName ()` - Specify the name of the geometry file to write.
- `obj.SetDisplacementFileName (string )` - Specify the name of the displacement file to write.
- `string = obj.GetDisplacementFileName ()` - Specify the name of the displacement file to write.
- `obj.SetScalarFileName (string )` - Specify the name of the scalar file to write.
- `string = obj.GetScalarFileName ()` - Specify the name of the scalar file to write.
- `obj.SetTextureFileName (string )` - Specify the name of the texture file to write.
- `string = obj.GetTextureFileName ()` - Specify the name of the texture file to write.

- `obj.SetWriteDisplacement (int )` - Turn on/off writing the displacement file.
- `int = obj.GetWriteDisplacement ()` - Turn on/off writing the displacement file.
- `obj.WriteDisplacementOn ()` - Turn on/off writing the displacement file.
- `obj.WriteDisplacementOff ()` - Turn on/off writing the displacement file.
- `obj.SetWriteScalar (int )` - Turn on/off writing the scalar file.
- `int = obj.GetWriteScalar ()` - Turn on/off writing the scalar file.
- `obj.WriteScalarOn ()` - Turn on/off writing the scalar file.
- `obj.WriteScalarOff ()` - Turn on/off writing the scalar file.
- `obj.SetWriteTexture (int )` - Turn on/off writing the texture file.
- `int = obj.GetWriteTexture ()` - Turn on/off writing the texture file.
- `obj.WriteTextureOn ()` - Turn on/off writing the texture file.
- `obj.WriteTextureOff ()` - Turn on/off writing the texture file.

37.12 vtkCGMWriter

37.12.1 Usage

vtkCGMWriter writes CGM (Computer Graphics Metafile) output. CGM is a 2D graphics vector format typically used by large plotters. This writer can handle vertices, lines, polygons, and triangle strips in any combination. Colors are specified either 1) from cell scalars (assumed to be RGB or RGBA color specification), 2) from a specified color; or 3) randomly assigned colors.

Note: During output of the polygonal data, triangle strips are converted to triangles, and polylines to lines. Also, due to limitations in the CGM color model, only 256 colors are available to the color palette.

To create an instance of class vtkCGMWriter, simply invoke its constructor as follows

```
obj = vtkCGMWriter
```

37.12.2 Methods

The class vtkCGMWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCGMWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCGMWriter = obj.NewInstance ()`
- `vtkCGMWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetViewport (vtkViewport )` - Specify a vtkViewport object to be used to transform the vtkPolyData points into 2D coordinates. By default (no vtkViewport specified), the point coordinates are generated by ignoring the z values. If a viewport is defined, then the points are transformed into viewport coordinates.

- `vtkViewport = obj.GetViewport ()` - Specify a vtkViewport object to be used to transform the vtkPolyData points into 2D coordinates. By default (no vtkViewport specified), the point coordinates are generated by ignoring the z values. If a viewport is defined, then the points are transformed into viewport coordinates.

- `obj.SetSort (int )` - Turn on/off the sorting of the cells via depth. If enabled, polygonal cells will be sorted from back to front, i.e., a Painter's algorithm sort.

- `int = obj.GetSort ()` - Turn on/off the sorting of the cells via depth. If enabled, polygonal cells will be sorted from back to front, i.e., a Painter's algorithm sort.

- `obj.SetResolution (int )` - Specify the resolution of the CGM file. This number is used to integerize the maximum coordinate range of the plot file.

- `int = obj.GetResolutionMinValue ()` - Specify the resolution of the CGM file. This number is used to integerize the maximum coordinate range of the plot file.

- `int = obj.GetResolutionMaxValue ()` - Specify the resolution of the CGM file. This number is used to integerize the maximum coordinate range of the plot file.

- `int = obj.GetResolution ()` - Specify the resolution of the CGM file. This number is used to integerize the maximum coordinate range of the plot file.

- `obj.SetColorMode (int )` - Control how output polydata is colored. By default (ColorModeToDefault), if per cell colors are defined (unsigned chars of 1-4 components), then the cells are colored with these values. (If point colors are defined and cell colors are not, you can use vtkPointDataToCellData to convert the point colors to cell colors.) Otherwise, by default, the cells are set to the specified color. If ColorModeToSpecifiedColor is set, then the primitives will all be set to this color. If ColorModeToRandomColors is set, each cell will be randomly assigned a color.

- `int = obj.GetColorMode ()` - Control how output polydata is colored. By default (ColorModeToDefault), if per cell colors are defined (unsigned chars of 1-4 components), then the cells are colored with these values. (If point colors are defined and cell colors are not, you can use vtkPointDataToCellData to convert the point colors to cell colors.) Otherwise, by default, the cells are set to the specified color. If ColorModeToSpecifiedColor is set, then the primitives will all be set to this color. If ColorModeToRandomColors is set, each cell will be randomly assigned a color.

- `obj.SetColorModeToDefault ()` - Control how output polydata is colored. By default (ColorModeToDefault), if per cell colors are defined (unsigned chars of 1-4 components), then the cells are colored with these values. (If point colors are defined and cell colors are not, you can use vtkPointDataToCellData to convert the point colors to cell colors.) Otherwise, by default, the cells are set to the specified color. If ColorModeToSpecifiedColor is set, then the primitives will all be set to this color. If ColorModeToRandomColors is set, each cell will be randomly assigned a color.

- `obj.SetColorModeToSpecifiedColor ()` - Control how output polydata is colored. By default (ColorModeToDefault), if per cell colors are defined (unsigned chars of 1-4 components), then the cells are colored with these values. (If point colors are defined and cell colors are not, you can use vtkPointDataToCellData to convert the point colors to cell colors.) Otherwise, by default, the cells are set to the specified color. If ColorModeToSpecifiedColor is set, then the primitives will all be set to this color. If ColorModeToRandomColors is set, each cell will be randomly assigned a color.

- `obj.SetColorModeToRandomColors ()` - Control how output polydata is colored. By default (ColorModeToDefault), if per cell colors are defined (unsigned chars of 1-4 components), then the cells are colored with these values. (If point colors are defined and cell colors are not, you can use vtkPointDataToCellData to convert the point colors to cell colors.) Otherwise, by default, the cells are set to the specified color. If ColorModeToSpecifiedColor is set, then the primitives will all be set to this color. If ColorModeToRandomColors is set, each cell will be randomly assigned a color.

- `obj.SetSpecifiedColor (float , float , float )` - Set/Get the specified color to color the polydata cells. This color is only used when the color mode is set to ColorModeToSpecifiedColor, or ColorModeToDefault is set and no cell colors are specified. The specified color is specified as RGB values ranging from (0,1). (Note: CGM will map this color to the closest color it supports.)
- `obj.SetSpecifiedColor (float a[3])` - Set/Get the specified color to color the polydata cells. This color is only used when the color mode is set to ColorModeToSpecifiedColor, or ColorModeToDefault is set and no cell colors are specified. The specified color is specified as RGB values ranging from (0,1). (Note: CGM will map this color to the closest color it supports.)
- `float = obj. GetSpecifiedColor ()` - Set/Get the specified color to color the polydata cells. This color is only used when the color mode is set to ColorModeToSpecifiedColor, or ColorModeToDefault is set and no cell colors are specified. The specified color is specified as RGB values ranging from (0,1). (Note: CGM will map this color to the closest color it supports.)

37.13 vtkChacoReader

37.13.1 Usage

vtkChacoReader is an unstructured grid source object that reads Chaco files. The reader DOES NOT respond to piece requests. Chaco is a graph partitioning package developed at Sandia National Laboratories in the early 1990s. (http://www.cs.sandia.gov/ bahendr/chaco.html)

Note that the Chaco "edges" become VTK "cells", and the Chaco "vertices" become VTK "points".

To create an instance of class vtkChacoReader, simply invoke its constructor as follows

```
obj = vtkChacoReader
```

37.13.2 Methods

The class vtkChacoReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkChacoReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkChacoReader = obj.NewInstance ()`
- `vtkChacoReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetBaseName (string )`
- `string = obj.GetBaseName ()`
- `obj.SetGenerateGlobalElementIdArray (int )`
- `int = obj.GetGenerateGlobalElementIdArray ()`
- `obj.GenerateGlobalElementIdArrayOn ()`
- `obj.GenerateGlobalElementIdArrayOff ()`
- `obj.SetGenerateGlobalNodeIdArray (int )`
- `int = obj.GetGenerateGlobalNodeIdArray ()`
- `obj.GenerateGlobalNodeIdArrayOn ()`

- `obj.GenerateGlobalNodeIdArrayOff ()`
- `obj.SetGenerateVertexWeightArrays (int )`
- `int = obj.GetGenerateVertexWeightArrays ()`
- `obj.GenerateVertexWeightArraysOn ()`
- `obj.GenerateVertexWeightArraysOff ()`
- `int = obj.GetNumberOfVertexWeights ()`
- `string = obj.GetVertexWeightArrayName (int weight)`
- `obj.SetGenerateEdgeWeightArrays (int )`
- `int = obj.GetGenerateEdgeWeightArrays ()`
- `obj.GenerateEdgeWeightArraysOn ()`
- `obj.GenerateEdgeWeightArraysOff ()`
- `int = obj.GetNumberOfEdgeWeights ()`
- `string = obj.GetEdgeWeightArrayName (int weight)`
- `int = obj.GetDimensionality ()` - Access to meta data generated by RequestInformation.
- `vtkIdType = obj.GetNumberOfEdges ()` - Access to meta data generated by RequestInformation.
- `vtkIdType = obj.GetNumberOfVertices ()` - Access to meta data generated by RequestInformation.
- `int = obj.GetNumberOfCellWeightArrays ()`
- `int = obj.GetNumberOfPointWeightArrays ()`

37.14 vtkDataCompressor

37.14.1 Usage

vtkDataCompressor provides a universal interface for data compression. Subclasses provide one compression method and one decompression method. The public interface to all compressors remains the same, and is defined by this class.

To create an instance of class vtkDataCompressor, simply invoke its constructor as follows

```
obj = vtkDataCompressor
```

37.14.2 Methods

The class vtkDataCompressor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataCompressor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataCompressor = obj.NewInstance ()`
- `vtkDataCompressor = obj.SafeDownCast (vtkObject o)`

- `long = obj.GetMaximumCompressionSpace (long size)` - Get the maximum space that may be needed to store data of the given uncompressed size after compression. This is the minimum size of the output buffer that can be passed to the four-argument Compress method.

- `long = obj.Compress (string uncompressedData, long uncompressedSize, string compressedData, long co` - Compress the given input data buffer into the given output buffer. The size of the output buffer must be at least as large as the value given by GetMaximumCompressionSpace for the given input size.

- `long = obj.Uncompress (string compressedData, long compressedSize, string uncompressedData, long un` - Uncompress the given input data into the given output buffer. The size of the uncompressed data must be known by the caller. It should be transmitted from the compressor by a means outside of this class.

- `vtkUnsignedCharArray = obj.Compress (string uncompressedData, long uncompressedSize)` - Compress the given data. A vtkUnsignedCharArray containing the compressed data is returned with a reference count of 1.

- `vtkUnsignedCharArray = obj.Uncompress (string compressedData, long compressedSize, long uncompresse` - Uncompress the given data. A vtkUnsignedCharArray containing the compressed data is returned with a reference count of 1. The size of the uncompressed data must be known by the caller. It should be transmitted from the compressor by a means outside of this class.

37.15 vtkDataObjectReader

37.15.1 Usage

vtkDataObjectReader is a source object that reads ASCII or binary field data files in vtk format. Fields are general matrix structures used represent complex data. (See text for format details). The output of this reader is a single vtkDataObject. The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkDataObjectReader, simply invoke its constructor as follows

```
obj = vtkDataObjectReader
```

37.15.2 Methods

The class vtkDataObjectReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObjectReader class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkDataObjectReader = obj.NewInstance ()`

- `vtkDataObjectReader = obj.SafeDownCast (vtkObject o)`

- `vtkDataObject = obj.GetOutput ()` - Get the output field of this reader.

- `vtkDataObject = obj.GetOutput (int idx)` - Get the output field of this reader.

- `obj.SetOutput (vtkDataObject )` - Get the output field of this reader.

37.16 vtkDataObjectWriter

37.16.1 Usage

vtkDataObjectWriter is a source object that writes ASCII or binary field data files in vtk format. Field data is a general form of data in matrix form.

To create an instance of class vtkDataObjectWriter, simply invoke its constructor as follows

```
obj = vtkDataObjectWriter
```

37.16.2 Methods

The class vtkDataObjectWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataObjectWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataObjectWriter = obj.NewInstance ()`
- `vtkDataObjectWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string filename)` - Methods delegated to vtkDataWriter, see vtkDataWriter.
- `string = obj.GetFileName ()` - Methods delegated to vtkDataWriter, see vtkDataWriter.
- `obj.SetHeader (string header)` - Methods delegated to vtkDataWriter, see vtkDataWriter.
- `string = obj.GetHeader ()` - Methods delegated to vtkDataWriter, see vtkDataWriter.
- `obj.SetFileType (int type)` - Methods delegated to vtkDataWriter, see vtkDataWriter.
- `int = obj.GetFileType ()` - Methods delegated to vtkDataWriter, see vtkDataWriter.
- `obj.SetFileTypeToASCII ()` - Methods delegated to vtkDataWriter, see vtkDataWriter.
- `obj.SetFileTypeToBinary ()` - Methods delegated to vtkDataWriter, see vtkDataWriter.
- `obj.SetFieldDataName (string fieldname)` - Methods delegated to vtkDataWriter, see vtkDataWriter.
- `string = obj.GetFieldDataName ()` - Methods delegated to vtkDataWriter, see vtkDataWriter.

37.17 vtkDataReader

37.17.1 Usage

vtkDataReader is a helper superclass that reads the vtk data file header, dataset type, and attribute data (point and cell attributes such as scalars, vectors, normals, etc.) from a vtk data file. See text for the format of the various vtk file types.

To create an instance of class vtkDataReader, simply invoke its constructor as follows

```
obj = vtkDataReader
```

37.17.2 Methods

The class vtkDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataReader = obj.NewInstance ()`
- `vtkDataReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name of vtk data file to read.
- `string = obj.GetFileName ()` - Specify file name of vtk data file to read.
- `int = obj.IsFileValid (string dstype)` - Is the file a valid vtk file of the passed dataset type ? The dataset type is passed as a lower case string.
- `int = obj.IsFileStructuredPoints ()` - Is the file a valid vtk file of the passed dataset type ? The dataset type is passed as a lower case string.
- `int = obj.IsFilePolyData ()` - Is the file a valid vtk file of the passed dataset type ? The dataset type is passed as a lower case string.
- `int = obj.IsFileStructuredGrid ()` - Is the file a valid vtk file of the passed dataset type ? The dataset type is passed as a lower case string.
- `int = obj.IsFileUnstructuredGrid ()` - Is the file a valid vtk file of the passed dataset type ? The dataset type is passed as a lower case string.
- `int = obj.IsFileRectilinearGrid ()` - Is the file a valid vtk file of the passed dataset type ? The dataset type is passed as a lower case string.
- `obj.SetInputString (string in)` - Specify the InputString for use when reading from a character array. Optionally include the length for binary strings. Note that a copy of the string is made and stored. If this causes exceedingly large memory consumption, consider using InputArray instead.
- `string = obj.GetInputString ()` - Specify the InputString for use when reading from a character array. Optionally include the length for binary strings. Note that a copy of the string is made and stored. If this causes exceedingly large memory consumption, consider using InputArray instead.
- `obj.SetInputString (string in, int len)` - Specify the InputString for use when reading from a character array. Optionally include the length for binary strings. Note that a copy of the string is made and stored. If this causes exceedingly large memory consumption, consider using InputArray instead.
- `int = obj.GetInputStringLength ()` - Specify the InputString for use when reading from a character array. Optionally include the length for binary strings. Note that a copy of the string is made and stored. If this causes exceedingly large memory consumption, consider using InputArray instead.
- `obj.SetBinaryInputString (string , int len)` - Specify the InputString for use when reading from a character array. Optionally include the length for binary strings. Note that a copy of the string is made and stored. If this causes exceedingly large memory consumption, consider using InputArray instead.

- `obj.SetInputArray (vtkCharArray )` - Specify the vtkCharArray to be used when reading from a string. If set, this array has precendence over InputString. Use this instead of InputString to avoid the extra memory copy. It should be noted that if the underlying char* is owned by the user (vtkCharArray::SetArray(array, 1);) and is deleted before the reader, bad things will happen during a pipeline update.
- `vtkCharArray = obj.GetInputArray ()` - Specify the vtkCharArray to be used when reading from a string. If set, this array has precendence over InputString. Use this instead of InputString to avoid the extra memory copy. It should be noted that if the underlying char* is owned by the user (vtkCharArray::SetArray(array, 1);) and is deleted before the reader, bad things will happen during a pipeline update.
- `string = obj.GetHeader ()` - Get the header from the vtk data file.
- `obj.SetReadFromInputString (int )` - Enable reading from an InputString or InputArray instead of the default, a file.
- `int = obj.GetReadFromInputString ()` - Enable reading from an InputString or InputArray instead of the default, a file.
- `obj.ReadFromInputStringOn ()` - Enable reading from an InputString or InputArray instead of the default, a file.
- `obj.ReadFromInputStringOff ()` - Enable reading from an InputString or InputArray instead of the default, a file.
- `int = obj.GetFileType ()` - Get the type of file (ASCII or BINARY). Returned value only valid after file has been read.
- `int = obj.GetNumberOfScalarsInFile ()` - How many attributes of various types are in this file? This requires reading the file, so the filename must be set prior to invoking this operation. (Note: file characteristics are cached, so only a single read is necessary to return file characteristics.)
- `int = obj.GetNumberOfVectorsInFile ()` - How many attributes of various types are in this file? This requires reading the file, so the filename must be set prior to invoking this operation. (Note: file characteristics are cached, so only a single read is necessary to return file characteristics.)
- `int = obj.GetNumberOfTensorsInFile ()` - How many attributes of various types are in this file? This requires reading the file, so the filename must be set prior to invoking this operation. (Note: file characteristics are cached, so only a single read is necessary to return file characteristics.)
- `int = obj.GetNumberOfNormalsInFile ()` - How many attributes of various types are in this file? This requires reading the file, so the filename must be set prior to invoking this operation. (Note: file characteristics are cached, so only a single read is necessary to return file characteristics.)
- `int = obj.GetNumberOfTCoordsInFile ()` - How many attributes of various types are in this file? This requires reading the file, so the filename must be set prior to invoking this operation. (Note: file characteristics are cached, so only a single read is necessary to return file characteristics.)
- `int = obj.GetNumberOfFieldDataInFile ()` - What is the name of the ith attribute of a certain type in this file? This requires reading the file, so the filename must be set prior to invoking this operation.
- `string = obj.GetScalarsNameInFile (int i)` - What is the name of the ith attribute of a certain type in this file? This requires reading the file, so the filename must be set prior to invoking this operation.
- `string = obj.GetVectorsNameInFile (int i)` - What is the name of the ith attribute of a certain type in this file? This requires reading the file, so the filename must be set prior to invoking this operation.

- `string = obj.GetTensorsNameInFile (int i)` - What is the name of the ith attribute of a certain type in this file? This requires reading the file, so the filename must be set prior to invoking this operation.
- `string = obj.GetNormalsNameInFile (int i)` - What is the name of the ith attribute of a certain type in this file? This requires reading the file, so the filename must be set prior to invoking this operation.
- `string = obj.GetTCoordsNameInFile (int i)` - What is the name of the ith attribute of a certain type in this file? This requires reading the file, so the filename must be set prior to invoking this operation.
- `string = obj.GetFieldDataNameInFile (int i)` - What is the name of the ith attribute of a certain type in this file? This requires reading the file, so the filename must be set prior to invoking this operation.
- `obj.SetScalarsName (string )` - Set the name of the scalar data to extract. If not specified, first scalar data encountered is extracted.
- `string = obj.GetScalarsName ()` - Set the name of the scalar data to extract. If not specified, first scalar data encountered is extracted.
- `obj.SetVectorsName (string )` - Set the name of the vector data to extract. If not specified, first vector data encountered is extracted.
- `string = obj.GetVectorsName ()` - Set the name of the vector data to extract. If not specified, first vector data encountered is extracted.
- `obj.SetTensorsName (string )` - Set the name of the tensor data to extract. If not specified, first tensor data encountered is extracted.
- `string = obj.GetTensorsName ()` - Set the name of the tensor data to extract. If not specified, first tensor data encountered is extracted.
- `obj.SetNormalsName (string )` - Set the name of the normal data to extract. If not specified, first normal data encountered is extracted.
- `string = obj.GetNormalsName ()` - Set the name of the normal data to extract. If not specified, first normal data encountered is extracted.
- `obj.SetTCoordsName (string )` - Set the name of the texture coordinate data to extract. If not specified, first texture coordinate data encountered is extracted.
- `string = obj.GetTCoordsName ()` - Set the name of the texture coordinate data to extract. If not specified, first texture coordinate data encountered is extracted.
- `obj.SetLookupTableName (string )` - Set the name of the lookup table data to extract. If not specified, uses lookup table named by scalar. Otherwise, this specification supersedes.
- `string = obj.GetLookupTableName ()` - Set the name of the lookup table data to extract. If not specified, uses lookup table named by scalar. Otherwise, this specification supersedes.
- `obj.SetFieldDataName (string )` - Set the name of the field data to extract. If not specified, uses first field data encountered in file.
- `string = obj.GetFieldDataName ()` - Set the name of the field data to extract. If not specified, uses first field data encountered in file.
- `obj.SetReadAllScalars (int )` - Enable reading all scalars.
- `int = obj.GetReadAllScalars ()` - Enable reading all scalars.

- `obj.ReadAllScalarsOn ()` - Enable reading all scalars.
- `obj.ReadAllScalarsOff ()` - Enable reading all scalars.
- `obj.SetReadAllVectors (int )` - Enable reading all vectors.
- `int = obj.GetReadAllVectors ()` - Enable reading all vectors.
- `obj.ReadAllVectorsOn ()` - Enable reading all vectors.
- `obj.ReadAllVectorsOff ()` - Enable reading all vectors.
- `obj.SetReadAllNormals (int )` - Enable reading all normals.
- `int = obj.GetReadAllNormals ()` - Enable reading all normals.
- `obj.ReadAllNormalsOn ()` - Enable reading all normals.
- `obj.ReadAllNormalsOff ()` - Enable reading all normals.
- `obj.SetReadAllTensors (int )` - Enable reading all tensors.
- `int = obj.GetReadAllTensors ()` - Enable reading all tensors.
- `obj.ReadAllTensorsOn ()` - Enable reading all tensors.
- `obj.ReadAllTensorsOff ()` - Enable reading all tensors.
- `obj.SetReadAllColorScalars (int )` - Enable reading all color scalars.
- `int = obj.GetReadAllColorScalars ()` - Enable reading all color scalars.
- `obj.ReadAllColorScalarsOn ()` - Enable reading all color scalars.
- `obj.ReadAllColorScalarsOff ()` - Enable reading all color scalars.
- `obj.SetReadAllTCoords (int )` - Enable reading all tcoords.
- `int = obj.GetReadAllTCoords ()` - Enable reading all tcoords.
- `obj.ReadAllTCoordsOn ()` - Enable reading all tcoords.
- `obj.ReadAllTCoordsOff ()` - Enable reading all tcoords.
- `obj.SetReadAllFields (int )` - Enable reading all fields.
- `int = obj.GetReadAllFields ()` - Enable reading all fields.
- `obj.ReadAllFieldsOn ()` - Enable reading all fields.
- `obj.ReadAllFieldsOff ()` - Enable reading all fields.
- `int = obj.OpenVTKFile ()` - Open a vtk data file. Returns zero if error.
- `int = obj.ReadHeader ()` - Read the header of a vtk data file. Returns 0 if error.
- `int = obj.ReadCellData (vtkDataSet ds, int numCells)` - Read the cell data of a vtk data file. The number of cells (from the dataset) must match the number of cells defined in cell attributes (unless no geometry was defined).
- `int = obj.ReadPointData (vtkDataSet ds, int numPts)` - Read the point data of a vtk data file. The number of points (from the dataset) must match the number of points defined in point attributes (unless no geometry was defined).
- `int = obj.ReadPoints (vtkPointSet ps, int numPts)` - Read point coordinates. Return 0 if error.

- `int = obj.ReadPoints (vtkGraph g, int numPts)` - Read point coordinates. Return 0 if error.
- `int = obj.ReadVertexData (vtkGraph g, int numVertices)` - Read the vertex data of a vtk data file. The number of vertices (from the graph) must match the number of vertices defined in vertex attributes (unless no geometry was defined).
- `int = obj.ReadEdgeData (vtkGraph g, int numEdges)` - Read the edge data of a vtk data file. The number of edges (from the graph) must match the number of edges defined in edge attributes (unless no geometry was defined).
- `int = obj.ReadRowData (vtkTable t, int numEdges)` - Read the row data of a vtk data file.
- `int = obj.ReadCells (int size, int data)` - Read a bunch of "cells". Return 0 if error.
- `int = obj.ReadCells (int size, int data, int skip1, int read2, int skip3)` - Read a piece of the cells (for streaming compliance)
- `int = obj.ReadCoordinates (vtkRectilinearGrid rg, int axes, int numCoords)` - Read the coordinates for a rectilinear grid. The axes parameter specifies which coordinate axes (0,1,2) is being read.
- `vtkAbstractArray = obj.ReadArray (string dataType, int numTuples, int numComp)` - Helper functions for reading data.
- `vtkFieldData = obj.ReadFieldData ()` - Helper functions for reading data.
- `obj.CloseVTKFile ()` - Close the vtk file.
- `int = obj.ReadMetaData (vtkInformation )`

37.18 vtkDataSetReader

37.18.1 Usage

vtkDataSetReader is a class that provides instance variables and methods to read any type of dataset in Visualization Toolkit (vtk) format. The output type of this class will vary depending upon the type of data file. Convenience methods are provided to keep the data as a particular type. (See text for format description details). The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkDataSetReader, simply invoke its constructor as follows

```
obj = vtkDataSetReader
```

37.18.2 Methods

The class vtkDataSetReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetReader = obj.NewInstance ()`
- `vtkDataSetReader = obj.SafeDownCast (vtkObject o)`
- `vtkDataSet = obj.GetOutput ()` - Get the output of this filter

- `vtkDataSet = obj.GetOutput (int idx)` - Get the output of this filter
- `vtkPolyData = obj.GetPolyDataOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkStructuredPoints = obj.GetStructuredPointsOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkStructuredGrid = obj.GetStructuredGridOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkUnstructuredGrid = obj.GetUnstructuredGridOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkRectilinearGrid = obj.GetRectilinearGridOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `int = obj.ReadOutputType ()` - This method can be used to find out the type of output expected without needing to read the whole file.

37.19 vtkDataSetWriter

37.19.1 Usage

vtkDataSetWriter is an abstract class for mapper objects that write their data to disk (or into a communications port). The input to this object is a dataset of any type.

To create an instance of class vtkDataSetWriter, simply invoke its constructor as follows

```
obj = vtkDataSetWriter
```

37.19.2 Methods

The class vtkDataSetWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetWriter = obj.NewInstance ()`
- `vtkDataSetWriter = obj.SafeDownCast (vtkObject o)`
- `vtkDataSet = obj.GetInput ()` - Get the input to this writer.
- `vtkDataSet = obj.GetInput (int port)` - Get the input to this writer.

37.20 vtkDataWriter

37.20.1 Usage

vtkDataWriter is a helper class that opens and writes the vtk header and point data (e.g., scalars, vectors, normals, etc.) from a vtk data file. See text for various formats.

To create an instance of class vtkDataWriter, simply invoke its constructor as follows

```
obj = vtkDataWriter
```

37.20.2 Methods

The class vtkDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataWriter = obj.NewInstance ()`
- `vtkDataWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name of vtk polygon data file to write.
- `string = obj.GetFileName ()` - Specify file name of vtk polygon data file to write.
- `obj.SetWriteToOutputString (int )` - Enable writing to an OutputString instead of the default, a file.
- `int = obj.GetWriteToOutputString ()` - Enable writing to an OutputString instead of the default, a file.
- `obj.WriteToOutputStringOn ()` - Enable writing to an OutputString instead of the default, a file.
- `obj.WriteToOutputStringOff ()` - Enable writing to an OutputString instead of the default, a file.
- `int = obj.GetOutputStringLength ()` - When WriteToOutputString in on, then a string is allocated, written to, and can be retrieved with these methods. The string is deleted during the next call to write ...
- `string = obj.GetOutputString ()` - When WriteToOutputString in on, then a string is allocated, written to, and can be retrieved with these methods. The string is deleted during the next call to write ...
- `string = obj.RegisterAndGetOutputString ()` - This convenience method returns the string, sets the IVAR to NULL, so that the user is responsible for deleting the string. I am not sure what the name should be, so it may change in the future.
- `obj.SetHeader (string )` - Specify the header for the vtk data file.
- `string = obj.GetHeader ()` - Specify the header for the vtk data file.
- `obj.SetFileType (int )` - Specify file type (ASCII or BINARY) for vtk data file.
- `int = obj.GetFileTypeMinValue ()` - Specify file type (ASCII or BINARY) for vtk data file.
- `int = obj.GetFileTypeMaxValue ()` - Specify file type (ASCII or BINARY) for vtk data file.
- `int = obj.GetFileType ()` - Specify file type (ASCII or BINARY) for vtk data file.

- `obj.SetFileTypeToASCII ()` - Specify file type (ASCII or BINARY) for vtk data file.
- `obj.SetFileTypeToBinary ()` - Specify file type (ASCII or BINARY) for vtk data file.
- `obj.SetScalarsName (string )` - Give a name to the scalar data. If not specified, uses default name "scalars".
- `string = obj.GetScalarsName ()` - Give a name to the scalar data. If not specified, uses default name "scalars".
- `obj.SetVectorsName (string )` - Give a name to the vector data. If not specified, uses default name "vectors".
- `string = obj.GetVectorsName ()` - Give a name to the vector data. If not specified, uses default name "vectors".
- `obj.SetTensorsName (string )` - Give a name to the tensors data. If not specified, uses default name "tensors".
- `string = obj.GetTensorsName ()` - Give a name to the tensors data. If not specified, uses default name "tensors".
- `obj.SetNormalsName (string )` - Give a name to the normals data. If not specified, uses default name "normals".
- `string = obj.GetNormalsName ()` - Give a name to the normals data. If not specified, uses default name "normals".
- `obj.SetTCoordsName (string )` - Give a name to the texture coordinates data. If not specified, uses default name "textureCoords".
- `string = obj.GetTCoordsName ()` - Give a name to the texture coordinates data. If not specified, uses default name "textureCoords".
- `obj.SetGlobalIdsName (string )` - Give a name to the global ids data. If not specified, uses default name "global_ids".
- `string = obj.GetGlobalIdsName ()` - Give a name to the global ids data. If not specified, uses default name "global_ids".
- `obj.SetPedigreeIdsName (string )` - Give a name to the pedigree ids data. If not specified, uses default name "pedigree_ids".
- `string = obj.GetPedigreeIdsName ()` - Give a name to the pedigree ids data. If not specified, uses default name "pedigree_ids".
- `obj.SetLookupTableName (string )` - Give a name to the lookup table. If not specified, uses default name "lookupTable".
- `string = obj.GetLookupTableName ()` - Give a name to the lookup table. If not specified, uses default name "lookupTable".
- `obj.SetFieldDataName (string )` - Give a name to the field data. If not specified, uses default name "field".
- `string = obj.GetFieldDataName ()` - Give a name to the field data. If not specified, uses default name "field".

37.21 vtkDEMReader

37.21.1 Usage

vtkDEMReader reads digital elevation files and creates image data. Digital elevation files are produced by the ¡A HREF="http://www.usgs.gov"¿US Geological Survey¡/A¿. A complete description of the DEM file is located at the USGS site. The reader reads the entire dem file and create a vtkImageData that contains a single scalar component that is the elevation in meters. The spacing is also expressed in meters. A number of get methods provide access to fields on the header.

To create an instance of class vtkDEMReader, simply invoke its constructor as follows

```
obj = vtkDEMReader
```

37.21.2 Methods

The class vtkDEMReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDEMReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDEMReader = obj.NewInstance ()`
- `vtkDEMReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name of Digital Elevation Model (DEM) file
- `string = obj.GetFileName ()` - Specify file name of Digital Elevation Model (DEM) file
- `obj.SetElevationReference (int )` - Specify the elevation origin to use. By default, the elevation origin is equal to ElevationBounds[0]. A more convenient origin is to use sea level (i.e., a value of 0.0).
- `int = obj.GetElevationReferenceMinValue ()` - Specify the elevation origin to use. By default, the elevation origin is equal to ElevationBounds[0]. A more convenient origin is to use sea level (i.e., a value of 0.0).
- `int = obj.GetElevationReferenceMaxValue ()` - Specify the elevation origin to use. By default, the elevation origin is equal to ElevationBounds[0]. A more convenient origin is to use sea level (i.e., a value of 0.0).
- `int = obj.GetElevationReference ()` - Specify the elevation origin to use. By default, the elevation origin is equal to ElevationBounds[0]. A more convenient origin is to use sea level (i.e., a value of 0.0).
- `obj.SetElevationReferenceToSeaLevel ()` - Specify the elevation origin to use. By default, the elevation origin is equal to ElevationBounds[0]. A more convenient origin is to use sea level (i.e., a value of 0.0).
- `obj.SetElevationReferenceToElevationBounds ()` - Specify the elevation origin to use. By default, the elevation origin is equal to ElevationBounds[0]. A more convenient origin is to use sea level (i.e., a value of 0.0).
- `string = obj.GetElevationReferenceAsString (void )` - Specify the elevation origin to use. By default, the elevation origin is equal to ElevationBounds[0]. A more convenient origin is to use sea level (i.e., a value of 0.0).
- `string = obj.GetMapLabel ()` - An ASCII description of the map

- `int = obj.GetDEMLevel ()` - Code 1=DEM-1, 2=DEM_2, ...
- `int = obj.GetElevationPattern ()` - Code 1=regular, 2=random, reserved for future use
- `int = obj.GetGroundSystem ()` - Ground planimetric reference system
- `int = obj.GetGroundZone ()` - Zone in ground planimetric reference system
- `float = obj. GetProjectionParameters ()` - Map Projection parameters. All are zero.
- `int = obj.GetPlaneUnitOfMeasure ()` - Defining unit of measure for ground planimetric coordinates throughout the file. 0 = radians, 1 = feet, 2 = meters, 3 = arc-seconds.
- `int = obj.GetElevationUnitOfMeasure ()` - Defining unit of measure for elevation coordinates throughout the file. 1 = feet, 2 = meters
- `int = obj.GetPolygonSize ()` - Number of sides in the polygon which defines the coverage of the DEM file. Set to 4.
- `float = obj. GetElevationBounds ()` - Minimum and maximum elevation for the DEM. The units in the file are in ElevationUnitOfMeasure. This class converts them to meters.
- `float = obj.GetLocalRotation ()` - Counterclockwise angle (in radians) from the primary axis of the planimetric reference to the primary axis of the DEM local reference system. IGNORED BY THIS IMPLEMENTATION.
- `int = obj.GetAccuracyCode ()` - Accuracy code for elevations. 0=unknown accuracy
- `float = obj. GetSpatialResolution ()` - DEM spatial resolution for x,y,z. Values are expressed in units of resolution. Since elevations are read as integers, this permits fractional elevations.
- `int = obj. GetProfileDimension ()` - The number of rows and columns in the DEM.

37.22 vtkDICOMImageReader

37.22.1 Usage

DICOM (stands for Digital Imaging in COmmunications and Medicine) is a medical image file format widely used to exchange data, provided by various modalities. .SECTION Warnings This reader might eventually handle ACR-NEMA file (predecessor of the DICOM format for medical images). This reader does not handle encapsulated format, only plain raw file are handled. This reader also does not handle multi-frames DICOM datasets. .SECTION Warnings Internally DICOMParser assumes the x,y pixel spacing is stored in 0028,0030 and that z spacing is stored in Slice Thickness (correct only when slice were acquired contiguous): 0018,0050. Which means this is only valid for some rare MR Image Storage

To create an instance of class vtkDICOMImageReader, simply invoke its constructor as follows

```
obj = vtkDICOMImageReader
```

37.22.2 Methods

The class vtkDICOMImageReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDICOMImageReader class.

- `string = obj.GetClassName ()` - Static method for construction.
- `int = obj.IsA (string name)` - Static method for construction.
- `vtkDICOMImageReader = obj.NewInstance ()` - Static method for construction.

- `vtkDICOMImageReader = obj.SafeDownCast (vtkObject o)` - Static method for construction.
- `obj.SetFileName (string fn)` - Set the directory name for the reader to look in for DICOM files. If this method is used, the reader will try to find all the DICOM files in a directory. It will select the subset corresponding to the first series UID it stumbles across and it will try to build an ordered volume from them based on the slice number. The volume building will be upgraded to something more sophisticated in the future.
- `obj.SetDirectoryName (string dn)` - Set the directory name for the reader to look in for DICOM files. If this method is used, the reader will try to find all the DICOM files in a directory. It will select the subset corresponding to the first series UID it stumbles across and it will try to build an ordered volume from them based on the slice number. The volume building will be upgraded to something more sophisticated in the future.
- `string = obj.GetDirectoryName ()` - Returns the directory name.
- `double = obj.GetPixelSpacing ()` - Returns the pixel spacing (in X, Y, Z). Note: if there is only one slice, the Z spacing is set to the slice thickness. If there is more than one slice, it is set to the distance between the first two slices.
- `int = obj.GetWidth ()` - Returns the image width.
- `int = obj.GetHeight ()` - Returns the image height.
- `float = obj.GetImagePositionPatient ()` - Get the (DICOM) x,y,z coordinates of the first pixel in the image (upper left hand corner) of the last image processed by the DICOMParser
- `float = obj.GetImageOrientationPatient ()` - Get the (DICOM) directions cosines. It consist of the components of the first two vectors. The third vector needs to be computed to form an orthonormal basis.
- `int = obj.GetBitsAllocated ()` - Get the number of bits allocated for each pixel in the file.
- `int = obj.GetPixelRepresentation ()` - Get the pixel representation of the last image processed by the DICOMParser. A zero is a unsigned quantity. A one indicates a signed quantity
- `int = obj.GetNumberOfComponents ()` - Get the number of components of the image data for the last image processed.
- `string = obj.GetTransferSyntaxUID ()` - Get the transfer syntax UID for the last image processed.
- `float = obj.GetRescaleSlope ()` - Get the rescale slope for the pixel data.
- `float = obj.GetRescaleOffset ()` - Get the rescale offset for the pixel data.
- `string = obj.GetPatientName ()` - Get the patient name for the last image processed.
- `string = obj.GetStudyUID ()` - Get the study uid for the last image processed.
- `string = obj.GetStudyID ()` - Get the Study ID for the last image processed.
- `float = obj.GetGantryAngle ()` - Get the gantry angle for the last image processed.
- `int = obj.CanReadFile (string fname)`
- `string = obj.GetFileExtensions ()` - Return a descriptive name for the file format that might be useful in a GUI.
- `string = obj.GetDescriptiveName ()`

37.23 vtkEnSight6BinaryReader

37.23.1 Usage

vtkEnSight6BinaryReader is a class to read binary EnSight6 files into vtk. Because the different parts of the EnSight data can be of various data types, this reader produces multiple outputs, one per part in the input file. All variable information is being stored in field data. The descriptions listed in the case file are used as the array names in the field data. For complex vector variables, the description is appended with _r (for the array of real values) and _i (for the array if imaginary values). Complex scalar variables are stored as a single array with 2 components, real and imaginary, listed in that order.

To create an instance of class vtkEnSight6BinaryReader, simply invoke its constructor as follows

```
obj = vtkEnSight6BinaryReader
```

37.23.2 Methods

The class vtkEnSight6BinaryReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEnSight6BinaryReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEnSight6BinaryReader = obj.NewInstance ()`
- `vtkEnSight6BinaryReader = obj.SafeDownCast (vtkObject o)`

37.24 vtkEnSight6Reader

37.24.1 Usage

vtkEnSight6Reader is a class to read EnSight6 files into vtk. Because the different parts of the EnSight data can be of various data types, this reader produces multiple outputs, one per part in the input file. All variable information is being stored in field data. The descriptions listed in the case file are used as the array names in the field data. For complex vector variables, the description is appended with _r (for the array of real values) and _i (for the array if imaginary values). Complex scalar variables are stored as a single array with 2 components, real and imaginary, listed in that order.

To create an instance of class vtkEnSight6Reader, simply invoke its constructor as follows

```
obj = vtkEnSight6Reader
```

37.24.2 Methods

The class vtkEnSight6Reader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEnSight6Reader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEnSight6Reader = obj.NewInstance ()`
- `vtkEnSight6Reader = obj.SafeDownCast (vtkObject o)`

37.25 vtkEnSightGoldBinaryReader

37.25.1 Usage

vtkEnSightGoldBinaryReader is a class to read EnSight Gold files into vtk. Because the different parts of the EnSight data can be of various data types, this reader produces multiple outputs, one per part in the input file. All variable information is being stored in field data. The descriptions listed in the case file are used as the array names in the field data. For complex vector variables, the description is appended with _r (for the array of real values) and _i (for the array if imaginary values). Complex scalar variables are stored as a single array with 2 components, real and imaginary, listed in that order.

To create an instance of class vtkEnSightGoldBinaryReader, simply invoke its constructor as follows

```
obj = vtkEnSightGoldBinaryReader
```

37.25.2 Methods

The class vtkEnSightGoldBinaryReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEnSightGoldBinaryReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEnSightGoldBinaryReader = obj.NewInstance ()`
- `vtkEnSightGoldBinaryReader = obj.SafeDownCast (vtkObject o)`

37.26 vtkEnSightGoldReader

37.26.1 Usage

vtkEnSightGoldReader is a class to read EnSight Gold files into vtk. Because the different parts of the EnSight data can be of various data types, this reader produces multiple outputs, one per part in the input file. All variable information is being stored in field data. The descriptions listed in the case file are used as the array names in the field data. For complex vector variables, the description is appended with _r (for the array of real values) and _i (for the array if imaginary values). Complex scalar variables are stored as a single array with 2 components, real and imaginary, listed in that order.

To create an instance of class vtkEnSightGoldReader, simply invoke its constructor as follows

```
obj = vtkEnSightGoldReader
```

37.26.2 Methods

The class vtkEnSightGoldReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEnSightGoldReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEnSightGoldReader = obj.NewInstance ()`
- `vtkEnSightGoldReader = obj.SafeDownCast (vtkObject o)`

37.27 vtkFacetWriter

37.27.1 Usage

vtkFacetWriter creates an unstructured grid dataset. It reads ASCII files stored in Facet format

The facet format looks like this: FACET FILE ... nparts Part 1 name 0 npoints 0 0 p1x p1y p1z p2x p2y p2z ... 1 Part 1 name ncells npointspercell p1c1 p2c1 p3c1 ... pnc1 materialnum partnum p1c2 p2c2 p3c2 ... pnc2 materialnum partnum ...

To create an instance of class vtkFacetWriter, simply invoke its constructor as follows

```
obj = vtkFacetWriter
```

37.27.2 Methods

The class vtkFacetWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFacetWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFacetWriter = obj.NewInstance ()`
- `vtkFacetWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name of Facet datafile to read
- `string = obj.GetFileName ()` - Specify file name of Facet datafile to read
- `obj.Write ()` - Write data

37.28 vtkFLUENTReader

37.28.1 Usage

vtkFLUENTReader creates an unstructured grid dataset. It reads .cas and .dat files stored in FLUENT native format.

.SECTION Thanks Thanks to Brian W. Dotson & Terry E. Jordan (Department of Energy, National Energy Technology Laboratory) & Douglas McCorkle (Iowa State University) who developed this class. Please address all comments to Brian Dotson (brian.dotson@netl.doe.gov) & Terry Jordan (terry.jordan@sa.netl.doe.gov) & Doug McCorkle (mccdo@iastate.edu)

To create an instance of class vtkFLUENTReader, simply invoke its constructor as follows

```
obj = vtkFLUENTReader
```

37.28.2 Methods

The class vtkFLUENTReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFLUENTReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFLUENTReader = obj.NewInstance ()`

- `vtkFLUENTReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify the file name of the Fluent case file to read.
- `string = obj.GetFileName ()` - Specify the file name of the Fluent case file to read.
- `int = obj.GetNumberOfCells ()` - Get the total number of cells. The number of cells is only valid after a successful read of the data file is performed. Initial value is 0.
- `int = obj.GetNumberOfCellArrays (void )` - Get the number of cell arrays available in the input.
- `string = obj.GetCellArrayName (int index)` - Get the name of the cell array with the given index in the input.
- `int = obj.GetCellArrayStatus (string name)` - Get/Set whether the cell array with the given name is to be read.
- `obj.SetCellArrayStatus (string name, int status)` - Get/Set whether the cell array with the given name is to be read.
- `obj.DisableAllCellArrays ()` - Turn on/off all cell arrays.
- `obj.EnableAllCellArrays ()` - Turn on/off all cell arrays.
- `obj.SetDataByteOrderToBigEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.
- `obj.SetDataByteOrderToLittleEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.
- `int = obj.GetDataByteOrder ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.
- `obj.SetDataByteOrder (int )` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.

- `string = obj.GetDataByteOrderAsString ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.

37.29 vtkGAMBITReader

37.29.1 Usage

vtkGAMBITReader creates an unstructured grid dataset. It reads ASCII files stored in GAMBIT neutral format, with optional data stored at the nodes or at the cells of the model. A cell-based fielddata stores the material id.

To create an instance of class vtkGAMBITReader, simply invoke its constructor as follows

```
obj = vtkGAMBITReader
```

37.29.2 Methods

The class vtkGAMBITReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGAMBITReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGAMBITReader = obj.NewInstance ()`
- `vtkGAMBITReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify the file name of the GAMBIT data file to read.
- `string = obj.GetFileName ()` - Specify the file name of the GAMBIT data file to read.
- `int = obj.GetNumberOfCells ()` - Get the total number of cells. The number of cells is only valid after a successful read of the data file is performed.
- `int = obj.GetNumberOfNodes ()` - Get the total number of nodes. The number of nodes is only valid after a successful read of the data file is performed.
- `int = obj.GetNumberOfNodeFields ()` - Get the number of data components at the nodes and cells.
- `int = obj.GetNumberOfCellFields ()` - Get the number of data components at the nodes and cells.

37.30 vtkGaussianCubeReader

37.30.1 Usage

vtkGaussianCubeReader is a source object that reads ASCII files following the description in http://www.gaussian.com/00000
The FileName must be specified.

.SECTION Thanks Dr. Jean M. Favre who developed and contributed this class.

To create an instance of class vtkGaussianCubeReader, simply invoke its constructor as follows

```
obj = vtkGaussianCubeReader
```

37.30.2 Methods

The class vtkGaussianCubeReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGaussianCubeReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGaussianCubeReader = obj.NewInstance ()`
- `vtkGaussianCubeReader = obj.SafeDownCast (vtkObject o)`
- `vtkTransform = obj.GetTransform ()`
- `obj.SetFileName (string )`
- `string = obj.GetFileName ()`
- `vtkImageData = obj.GetGridOutput ()`

37.31 vtkGenericDataObjectReader

37.31.1 Usage

vtkGenericDataObjectReader is a class that provides instance variables and methods to read any type of data object in Visualization Toolkit (vtk) format. The output type of this class will vary depending upon the type of data file. Convenience methods are provided to return the data as a particular type. (See text for format description details). The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkGenericDataObjectReader, simply invoke its constructor as follows

```
obj = vtkGenericDataObjectReader
```

37.31.2 Methods

The class vtkGenericDataObjectReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericDataObjectReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenericDataObjectReader = obj.NewInstance ()`
- `vtkGenericDataObjectReader = obj.SafeDownCast (vtkObject o)`
- `vtkDataObject = obj.GetOutput ()` - Get the output of this filter
- `vtkDataObject = obj.GetOutput (int idx)` - Get the output of this filter
- `vtkGraph = obj.GetGraphOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)

- `vtkPolyData = obj.GetPolyDataOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkRectilinearGrid = obj.GetRectilinearGridOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkStructuredGrid = obj.GetStructuredGridOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkStructuredPoints = obj.GetStructuredPointsOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkTable = obj.GetTableOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkTree = obj.GetTreeOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `vtkUnstructuredGrid = obj.GetUnstructuredGridOutput ()` - Get the output as various concrete types. This method is typically used when you know exactly what type of data is being read. Otherwise, use the general GetOutput() method. If the wrong type is used NULL is returned. (You must also set the filename of the object prior to getting the output.)
- `int = obj.ReadOutputType ()` - This method can be used to find out the type of output expected without needing to read the whole file.

37.32 vtkGenericDataObjectWriter

37.32.1 Usage

vtkGenericDataObjectWriter is a concrete class that writes data objects to disk. The input to this object is any subclass of vtkDataObject.

To create an instance of class vtkGenericDataObjectWriter, simply invoke its constructor as follows

```
obj = vtkGenericDataObjectWriter
```

37.32.2 Methods

The class vtkGenericDataObjectWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericDataObjectWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkGenericDataObjectWriter = obj.NewInstance ()`
- `vtkGenericDataObjectWriter = obj.SafeDownCast (vtkObject o)`

37.33 vtkGenericEnSightReader

37.33.1 Usage

The class vtkGenericEnSightReader allows the user to read an EnSight data set without a priori knowledge of what type of EnSight data set it is.

To create an instance of class vtkGenericEnSightReader, simply invoke its constructor as follows

```
obj = vtkGenericEnSightReader
```

37.33.2 Methods

The class vtkGenericEnSightReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericEnSightReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenericEnSightReader = obj.NewInstance ()`
- `vtkGenericEnSightReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetCaseFileName (string fileName)` - Set/Get the Case file name.
- `string = obj.GetCaseFileName ()` - Set/Get the Case file name.
- `obj.SetFilePath (string )` - Set/Get the file path.
- `string = obj.GetFilePath ()` - Set/Get the file path.
- `int = obj.GetNumberOfVariables ()` - Get the number of variables listed in the case file.
- `int = obj.GetNumberOfComplexVariables ()` - Get the number of variables listed in the case file.
- `int = obj.GetNumberOfVariables (int type)` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfScalarsPerNode ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfVectorsPerNode ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfTensorsSymmPerNode ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfScalarsPerElement ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfVectorsPerElement ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfTensorsSymmPerElement ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfScalarsPerMeasuredNode ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfVectorsPerMeasuredNode ()` - Get the number of variables of a particular type.

- `int = obj.GetNumberOfComplexScalarsPerNode ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfComplexVectorsPerNode ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfComplexScalarsPerElement ()` - Get the number of variables of a particular type.
- `int = obj.GetNumberOfComplexVectorsPerElement ()` - Get the number of variables of a particular type.
- `string = obj.GetDescription (int n)` - Get the nth description for a non-complex variable.
- `string = obj.GetComplexDescription (int n)` - Get the nth description for a complex variable.
- `string = obj.GetDescription (int n, int type)` - Get the nth description of a particular variable type. Returns NULL if no variable of this type exists in this data set. SCALAR_PER_NODE = 0; VECTOR_PER_NODE = 1; TENSOR_SYMM_PER_NODE = 2; SCALAR_PER_ELEMENT = 3; VECTOR_PER_ELEMENT = 4; TENSOR_SYMM_PER_ELEMENT = 5; SCALAR_PER_MEASURED_NODE = 6; VECTOR_PER_MEASURED_NODE = 7; COMPLEX_SCALAR_PER_NODE = 8; COMPLEX_VECTOR_PER_N 9; COMPLEX_SCALAR_PER_ELEMENT = 10; COMPLEX_VECTOR_PER_ELEMENT = 11
- `int = obj.GetVariableType (int n)` - Get the variable type of variable n.
- `int = obj.GetComplexVariableType (int n)` - Get the variable type of variable n.
- `obj.SetTimeValue (float value)` - Set/Get the time value at which to get the value.
- `float = obj.GetTimeValue ()` - Set/Get the time value at which to get the value.
- `float = obj.GetMinimumTimeValue ()` - Get the minimum or maximum time value for this data set.
- `float = obj.GetMaximumTimeValue ()` - Get the minimum or maximum time value for this data set.
- `vtkDataArrayCollection = obj.GetTimeSets ()` - Get the time values per time set
- `int = obj.DetermineEnSightVersion (int quiet)` - Reads the FORMAT part of the case file to determine whether this is an EnSight6 or EnSightGold data set. Returns an identifier listed in the FileTypes enum or -1 if an error occurred or the file could not be indentified as any EnSight type.
- `obj.ReadAllVariablesOn ()` - Set/get the flag for whether to read all the variables
- `obj.ReadAllVariablesOff ()` - Set/get the flag for whether to read all the variables
- `obj.SetReadAllVariables (int )` - Set/get the flag for whether to read all the variables
- `int = obj.GetReadAllVariables ()` - Set/get the flag for whether to read all the variables
- `vtkDataArraySelection = obj.GetPointDataArraySelection ()` - Get the data array selection tables used to configure which data arrays are loaded by the reader.
- `vtkDataArraySelection = obj.GetCellDataArraySelection ()` - Get the data array selection tables used to configure which data arrays are loaded by the reader.
- `int = obj.GetNumberOfPointArrays ()` - Get the number of point or cell arrays available in the input.
- `int = obj.GetNumberOfCellArrays ()` - Get the number of point or cell arrays available in the input.
- `string = obj.GetPointArrayName (int index)` - Get the name of the point or cell array with the given index in the input.

- `string = obj.GetCellArrayName (int index)` - Get the name of the point or cell array with the given index in the input.
- `int = obj.GetPointArrayStatus (string name)` - Get/Set whether the point or cell array with the given name is to be read.
- `int = obj.GetCellArrayStatus (string name)` - Get/Set whether the point or cell array with the given name is to be read.
- `obj.SetPointArrayStatus (string name, int status)` - Get/Set whether the point or cell array with the given name is to be read.
- `obj.SetCellArrayStatus (string name, int status)` - Get/Set whether the point or cell array with the given name is to be read.
- `obj.SetByteOrderToBigEndian ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `obj.SetByteOrderToLittleEndian ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `obj.SetByteOrder (int )` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `int = obj.GetByteOrder ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `string = obj.GetByteOrderAsString ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `string = obj.GetGeometryFileName ()` - Get the Geometry file name. Made public to allow access from apps requiring detailed info about the Data contents
- `obj.SetParticleCoordinatesByIndex (int )` - The MeasuredGeometryFile should list particle coordinates from 0-¿N-1. If a file is loaded where point Ids are listed from 1-N the Id to points reference will be wrong and the data will be generated incorrectly. Setting ParticleCoordinatesByIndex to true will force all Id's to increment from 0-¿N-1 (relative to their order in the file) and regardless of the actual Id of of the point. Warning, if the Points are listed in non sequential order then setting this flag will reorder them.
- `int = obj.GetParticleCoordinatesByIndex ()` - The MeasuredGeometryFile should list particle coordinates from 0-¿N-1. If a file is loaded where point Ids are listed from 1-N the Id to points reference will be wrong and the data will be generated incorrectly. Setting ParticleCoordinatesByIndex to true will force all Id's to increment from 0-¿N-1 (relative to their order in the file) and regardless of the actual Id of of the point. Warning, if the Points are listed in non sequential order then setting this flag will reorder them.
- `obj.ParticleCoordinatesByIndexOn ()` - The MeasuredGeometryFile should list particle coordinates from 0-¿N-1. If a file is loaded where point Ids are listed from 1-N the Id to points reference will be wrong and the data will be generated incorrectly. Setting ParticleCoordinatesByIndex to true will force all Id's to increment from 0-¿N-1 (relative to their order in the file) and regardless of the actual Id of of the point. Warning, if the Points are listed in non sequential order then setting this flag will reorder them.

- `obj.ParticleCoordinatesByIndexOff ()` - The MeasuredGeometryFile should list particle coordinates from 0-¿N-1. If a file is loaded where point Ids are listed from 1-N the Id to points reference will be wrong and the data will be generated incorrectly. Setting ParticleCoordinatesByIndex to true will force all Id's to increment from 0-¿N-1 (relative to their order in the file) and regardless of the actual Id of of the point. Warning, if the Points are listed in non sequential order then setting this flag will reorder them.

37.34 vtkGenericMovieWriter

37.34.1 Usage

vtkGenericMovieWriter is the abstract base class for several movie writers. The input type is a vtkImageData. The Start() method will open and create the file, the Write() method will output a frame to the file (i.e. the contents of the vtkImageData), End() will finalize and close the file.

To create an instance of class vtkGenericMovieWriter, simply invoke its constructor as follows

```
obj = vtkGenericMovieWriter
```

37.34.2 Methods

The class vtkGenericMovieWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericMovieWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenericMovieWriter = obj.NewInstance ()`
- `vtkGenericMovieWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImageData input)` - Set/Get the input object from the image pipeline.
- `vtkImageData = obj.GetInput ()` - Set/Get the input object from the image pipeline.
- `obj.SetFileName (string )` - Specify file name of avi file.
- `string = obj.GetFileName ()` - Specify file name of avi file.
- `obj.Start ()` - These methods start writing an Movie file, write a frame to the file and then end the writing process.
- `obj.Write ()` - These methods start writing an Movie file, write a frame to the file and then end the writing process.
- `obj.End ()` - These methods start writing an Movie file, write a frame to the file and then end the writing process.
- `int = obj.GetError ()` - Was there an error on the last write performed?

37.35 vtkGESignaReader

37.35.1 Usage

vtkGESignaReader is a source object that reads some GE Signa ximg files It does support reading in pixel spacing, slice spacing and it computes an origin for the image in millimeters. It always produces greyscale unsigned short data and it supports reading in rectangular, packed, compressed, and packed&compressed. It does not read in slice orientation, or position right now. To use it you just need to specify a filename or a file prefix and pattern.

To create an instance of class vtkGESignaReader, simply invoke its constructor as follows

```
obj = vtkGESignaReader
```

37.35.2 Methods

The class vtkGESignaReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGESignaReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGESignaReader = obj.NewInstance ()`
- `vtkGESignaReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string fname)` - Is the given file a GESigna file?
- `string = obj.GetFileExtensions ()` - A descriptive name for this format
- `string = obj.GetDescriptiveName ()`

37.36 vtkGlobFileNames

37.36.1 Usage

vtkGlobFileNames is a utility for finding files and directories that match a given wildcard pattern. Allowed wildcards are *, ?, [...], [!...]. The "*" wildcard matches any substring, the "?" matches any single character, the [...] matches any one of the enclosed characters, e.g. [abc] will match one of a, b, or c, while [0-9] will match any digit, and [!...] will match any single character except for the ones within the brackets. Special treatment is given to "/" (or "\" on Windows) because these are path separators. These are never matched by a wildcard, they are only matched with another file separator.

To create an instance of class vtkGlobFileNames, simply invoke its constructor as follows

```
obj = vtkGlobFileNames
```

37.36.2 Methods

The class vtkGlobFileNames has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGlobFileNames class.

- `string = obj.GetClassName ()` - Return the class name as a string.
- `int = obj.IsA (string name)` - Return the class name as a string.

- `vtkGlobFileNames = obj.NewInstance ()` - Return the class name as a string.
- `vtkGlobFileNames = obj.SafeDownCast (vtkObject o)` - Return the class name as a string.
- `obj.Reset ()` - Reset the glob by clearing the list of output filenames.
- `obj.SetDirectory (string )` - Set the directory in which to perform the glob. If this is not set, then the current directory will be used. Also, if you use a glob pattern that contains absolute path (one that starts with "/" or a drive letter) then that absolute path will be used and Directory will be ignored.
- `string = obj.GetDirectory ()` - Set the directory in which to perform the glob. If this is not set, then the current directory will be used. Also, if you use a glob pattern that contains absolute path (one that starts with "/" or a drive letter) then that absolute path will be used and Directory will be ignored.
- `int = obj.AddFileNames (string pattern)` - Search for all files that match the given expression, sort them, and add them to the output. This method can be called repeatedly to add files matching additional patterns. Returns 1 if successful, otherwise returns zero.
- `obj.SetRecurse (int )` - Recurse into subdirectories.
- `obj.RecurseOn ()` - Recurse into subdirectories.
- `obj.RecurseOff ()` - Recurse into subdirectories.
- `int = obj.GetRecurse ()` - Recurse into subdirectories.
- `int = obj.GetNumberOfFileNames ()` - Return the number of files found.
- `string = obj.GetNthFileName (int index)` - Return the file at the given index, the indexing is 0 based.
- `vtkStringArray = obj.GetFileNames ()` - Get an array that contains all the file names.

37.37 vtkGraphReader

37.37.1 Usage

vtkGraphReader is a source object that reads ASCII or binary vtkGraph data files in vtk format. (see text for format details). The output of this reader is a single vtkGraph data object. The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkGraphReader, simply invoke its constructor as follows

```
obj = vtkGraphReader
```

37.37.2 Methods

The class vtkGraphReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphReader = obj.NewInstance ()`
- `vtkGraphReader = obj.SafeDownCast (vtkObject o)`

- `vtkGraph = obj.GetOutput ()` - Get the output of this reader.
- `vtkGraph = obj.GetOutput (int idx)` - Get the output of this reader.
- `obj.SetOutput (vtkGraph output)` - Get the output of this reader.

37.38 vtkGraphWriter

37.38.1 Usage

vtkGraphWriter is a sink object that writes ASCII or binary vtkGraph data files in vtk format. See text for format details.

To create an instance of class vtkGraphWriter, simply invoke its constructor as follows

```
obj = vtkGraphWriter
```

37.38.2 Methods

The class vtkGraphWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphWriter = obj.NewInstance ()`
- `vtkGraphWriter = obj.SafeDownCast (vtkObject o)`
- `vtkGraph = obj.GetInput ()` - Get the input to this writer.
- `vtkGraph = obj.GetInput (int port)` - Get the input to this writer.

37.39 vtkImageReader

37.39.1 Usage

vtkImageReader provides methods needed to read a region from a file. It supports both transforms and masks on the input data, but as a result is more complicated and slower than its parent class vtkImageReader2.

To create an instance of class vtkImageReader, simply invoke its constructor as follows

```
obj = vtkImageReader
```

37.39.2 Methods

The class vtkImageReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageReader = obj.NewInstance ()`
- `vtkImageReader = obj.SafeDownCast (vtkObject o)`

- `obj.SetDataVOI (int , int , int , int , int , int )` - Set/get the data VOI. You can limit the reader to only read a subset of the data.
- `obj.SetDataVOI (int a[6])` - Set/get the data VOI. You can limit the reader to only read a subset of the data.
- `int = obj. GetDataVOI ()` - Set/get the data VOI. You can limit the reader to only read a subset of the data.
- `vtkTypeUInt64 = obj.GetDataMask ()` - Set/Get the Data mask. The data mask is a simply integer whose bits are treated as a mask to the bits read from disk. That is, the data mask is bitwise-and'ed to the numbers read from disk. This ivar is stored as 64 bits, the largest mask you will need. The mask will be truncated to the data size required to be read (using the least significant bits).
- `obj.SetDataMask (vtkTypeUInt64 )` - Set/Get the Data mask. The data mask is a simply integer whose bits are treated as a mask to the bits read from disk. That is, the data mask is bitwise-and'ed to the numbers read from disk. This ivar is stored as 64 bits, the largest mask you will need. The mask will be truncated to the data size required to be read (using the least significant bits).
- `obj.SetTransform (vtkTransform )` - Set/Get transformation matrix to transform the data from slice space into world space. This matrix must be a permutation matrix. To qualify, the sums of the rows must be + or - 1.
- `vtkTransform = obj.GetTransform ()` - Set/Get transformation matrix to transform the data from slice space into world space. This matrix must be a permutation matrix. To qualify, the sums of the rows must be + or - 1.
- `obj.ComputeInverseTransformedExtent (int inExtent[6], int outExtent[6])`
- `int = obj.OpenAndSeekFile (int extent[6], int slice)`
- `obj.SetScalarArrayName (string )` - Set/get the scalar array name for this data set.
- `string = obj.GetScalarArrayName ()` - Set/get the scalar array name for this data set.

37.40 vtkImageReader2

37.40.1 Usage

vtkImageReader2 is the parent class for vtkImageReader. It is a good super class for streaming readers that do not require a mask or transform on the data. vtkImageReader was implemented before vtkImageReader2, vtkImageReader2 is intended to have a simpler interface.

To create an instance of class vtkImageReader2, simply invoke its constructor as follows

```
obj = vtkImageReader2
```

37.40.2 Methods

The class vtkImageReader2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageReader2 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageReader2 = obj.NewInstance ()`

- `vtkImageReader2 = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name for the image file. If the data is stored in multiple files, then use SetFileNames or SetFilePrefix instead.
- `string = obj.GetFileName ()` - Specify file name for the image file. If the data is stored in multiple files, then use SetFileNames or SetFilePrefix instead.
- `obj.SetFileNames (vtkStringArray )` - Specify a list of file names. Each file must be a single slice, and each slice must be of the same size. The files must be in the correct order. Use SetFileName when reading a volume (multiple slice), since DataExtent will be modified after a SetFileNames call.
- `vtkStringArray = obj.GetFileNames ()` - Specify a list of file names. Each file must be a single slice, and each slice must be of the same size. The files must be in the correct order. Use SetFileName when reading a volume (multiple slice), since DataExtent will be modified after a SetFileNames call.
- `obj.SetFilePrefix (string )` - Specify file prefix for the image file or files. This can be used in place of SetFileName or SetFileNames if the filenames follow a specific naming pattern, but you must explicitly set the DataExtent so that the reader will know what range of slices to load.
- `string = obj.GetFilePrefix ()` - Specify file prefix for the image file or files. This can be used in place of SetFileName or SetFileNames if the filenames follow a specific naming pattern, but you must explicitly set the DataExtent so that the reader will know what range of slices to load.
- `obj.SetFilePattern (string )` - The sprintf-style format string used to build filename from FilePrefix and slice number.
- `string = obj.GetFilePattern ()` - The sprintf-style format string used to build filename from FilePrefix and slice number.
- `obj.SetDataScalarType (int type)` - Set the data type of pixels in the file. If you want the output scalar type to have a different value, set it after this method is called.
- `obj.SetDataScalarTypeToFloat ()` - Set the data type of pixels in the file. If you want the output scalar type to have a different value, set it after this method is called.
- `obj.SetDataScalarTypeToDouble ()` - Set the data type of pixels in the file. If you want the output scalar type to have a different value, set it after this method is called.
- `obj.SetDataScalarTypeToInt ()` - Set the data type of pixels in the file. If you want the output scalar type to have a different value, set it after this method is called.
- `obj.SetDataScalarTypeToUnsignedInt ()` - Set the data type of pixels in the file. If you want the output scalar type to have a different value, set it after this method is called.
- `obj.SetDataScalarTypeToShort ()` - Set the data type of pixels in the file. If you want the output scalar type to have a different value, set it after this method is called.
- `obj.SetDataScalarTypeToUnsignedShort ()` - Set the data type of pixels in the file. If you want the output scalar type to have a different value, set it after this method is called.
- `obj.SetDataScalarTypeToChar ()` - Set the data type of pixels in the file. If you want the output scalar type to have a different value, set it after this method is called.
- `obj.SetDataScalarTypeToSignedChar ()` - Set the data type of pixels in the file. If you want the output scalar type to have a different value, set it after this method is called.
- `obj.SetDataScalarTypeToUnsignedChar ()` - Get the file format. Pixels are this type in the file.
- `int = obj.GetDataScalarType ()` - Get the file format. Pixels are this type in the file.

- `obj.SetNumberOfScalarComponents (int )` - Set/Get the number of scalar components
- `int = obj.GetNumberOfScalarComponents ()` - Set/Get the number of scalar components
- `obj.SetDataExtent (int , int , int , int , int , int )` - Get/Set the extent of the data on disk.
- `obj.SetDataExtent (int  a[6])` - Get/Set the extent of the data on disk.
- `int = obj. GetDataExtent ()` - Get/Set the extent of the data on disk.
- `obj.SetFileDimensionality (int )` - The number of dimensions stored in a file. This defaults to two.
- `int = obj.GetFileDimensionality ()` - Set/Get the spacing of the data in the file.
- `obj.SetDataSpacing (double , double , double )` - Set/Get the spacing of the data in the file.
- `obj.SetDataSpacing (double  a[3])` - Set/Get the spacing of the data in the file.
- `double = obj. GetDataSpacing ()` - Set/Get the spacing of the data in the file.
- `obj.SetDataOrigin (double , double , double )` - Set/Get the origin of the data (location of first pixel in the file).
- `obj.SetDataOrigin (double  a[3])` - Set/Get the origin of the data (location of first pixel in the file).
- `double = obj. GetDataOrigin ()` - Set/Get the origin of the data (location of first pixel in the file).
- `long = obj.GetHeaderSize ()` - Get the size of the header computed by this object.
- `long = obj.GetHeaderSize (long slice)` - Get the size of the header computed by this object.
- `obj.SetHeaderSize (long size)` - If there is a tail on the file, you want to explicitly set the header size.
- `obj.SetDataByteOrderToBigEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.
- `obj.SetDataByteOrderToLittleEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.
- `int = obj.GetDataByteOrder ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.

- `obj.SetDataByteOrder (int )` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.
- `string = obj.GetDataByteOrderAsString ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.
- `obj.SetFileNameSliceOffset (int )` - When reading files which start at an unusual index, this can be added to the slice number when generating the file name (default = 0)
- `int = obj.GetFileNameSliceOffset ()` - When reading files which start at an unusual index, this can be added to the slice number when generating the file name (default = 0)
- `obj.SetFileNameSliceSpacing (int )` - When reading files which have regular, but non contiguous slices (eg filename.1,filename.3,filename.5) a spacing can be specified to skip missing files (default = 1)
- `int = obj.GetFileNameSliceSpacing ()` - When reading files which have regular, but non contiguous slices (eg filename.1,filename.3,filename.5) a spacing can be specified to skip missing files (default = 1)
- `obj.SetSwapBytes (int )` - Set/Get the byte swapping to explicitly swap the bytes of a file.
- `int = obj.GetSwapBytes ()` - Set/Get the byte swapping to explicitly swap the bytes of a file.
- `obj.SwapBytesOn ()` - Set/Get the byte swapping to explicitly swap the bytes of a file.
- `obj.SwapBytesOff ()` - Set/Get the byte swapping to explicitly swap the bytes of a file.
- `int = obj.OpenFile ()`
- `obj.SeekFile (int i, int j, int k)`
- `obj.FileLowerLeftOn ()` - Set/Get whether the data comes from the file starting in the lower left corner or upper left corner.
- `obj.FileLowerLeftOff ()` - Set/Get whether the data comes from the file starting in the lower left corner or upper left corner.
- `int = obj.GetFileLowerLeft ()` - Set/Get whether the data comes from the file starting in the lower left corner or upper left corner.
- `obj.SetFileLowerLeft (int )` - Set/Get whether the data comes from the file starting in the lower left corner or upper left corner.
- `obj.ComputeInternalFileName (int slice)` - Set/Get the internal file name
- `string = obj.GetInternalFileName ()` - Set/Get the internal file name
- `int = obj.CanReadFile (string )` - Get the file extensions for this format. Returns a string with a space separated list of extensions in the format .extension

- `string = obj.GetFileExtensions ()` - Return a descriptive name for the file format that might be useful in a GUI.
- `string = obj.GetDescriptiveName ()` - Return a descriptive name for the file format that might be useful in a GUI.

37.41 vtkImageReader2Collection

37.41.1 Usage

vtkImageReader2Collection is an object that creates and manipulates lists of objects of type vtkImageReader2 and its subclasses.

To create an instance of class vtkImageReader2Collection, simply invoke its constructor as follows

```
obj = vtkImageReader2Collection
```

37.41.2 Methods

The class vtkImageReader2Collection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageReader2Collection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageReader2Collection = obj.NewInstance ()`
- `vtkImageReader2Collection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkImageReader2 )` - Add an image reader to the list.
- `vtkImageReader2 = obj.GetNextItem ()` - Get the next image reader in the list.

37.42 vtkImageReader2Factory

37.42.1 Usage

vtkImageReader2Factory: This class is used to create a vtkImageReader2 object given a path name to a file. It calls CanReadFile on all available readers until one of them returns true. The available reader list comes from three places. In the InitializeReaders function of this class, built-in VTK classes are added to the list, users can call RegisterReader, or users can create a vtkObjectFactory that has CreateObject method that returns a new vtkImageReader2 sub class when given the string "vtkImageReaderObject". This way applications can be extended with new readers via a plugin dll or by calling RegisterReader. Of course all of the readers that are part of the vtk release are made automatically available.

To create an instance of class vtkImageReader2Factory, simply invoke its constructor as follows

```
obj = vtkImageReader2Factory
```

37.42.2 Methods

The class vtkImageReader2Factory has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageReader2Factory class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkImageReader2Factory = obj.NewInstance ()`
- `vtkImageReader2Factory = obj.SafeDownCast (vtkObject o)`

37.43 vtkImageWriter

37.43.1 Usage

vtkImageWriter writes images to files with any data type. The data type of the file is the same scalar type as the input. The dimensionality determines whether the data will be written in one or multiple files. This class is used as the superclass of most image writing classes such as vtkBMPWriter etc. It supports streaming.

To create an instance of class vtkImageWriter, simply invoke its constructor as follows

```
obj = vtkImageWriter
```

37.43.2 Methods

The class vtkImageWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageWriter = obj.NewInstance ()`
- `vtkImageWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name for the image file. You should specify either a FileName or a FilePrefix. Use FilePrefix if the data is stored in multiple files.
- `string = obj.GetFileName ()` - Specify file name for the image file. You should specify either a FileName or a FilePrefix. Use FilePrefix if the data is stored in multiple files.
- `obj.SetFilePrefix (string )` - Specify file prefix for the image file(s).You should specify either a FileName or FilePrefix. Use FilePrefix if the data is stored in multiple files.
- `string = obj.GetFilePrefix ()` - Specify file prefix for the image file(s).You should specify either a FileName or FilePrefix. Use FilePrefix if the data is stored in multiple files.
- `obj.SetFilePattern (string )` - The sprintf format used to build filename from FilePrefix and number.
- `string = obj.GetFilePattern ()` - The sprintf format used to build filename from FilePrefix and number.
- `obj.SetFileDimensionality (int )` - What dimension are the files to be written. Usually this is 2, or 3. If it is 2 and the input is a volume then the volume will be written as a series of 2d slices.
- `int = obj.GetFileDimensionality ()` - What dimension are the files to be written. Usually this is 2, or 3. If it is 2 and the input is a volume then the volume will be written as a series of 2d slices.
- `obj.Write ()` - The main interface which triggers the writer to start.
- `obj.DeleteFiles ()`

37.44 vtkInputStream

37.44.1 Usage

vtkInputStream provides a VTK-style interface wrapping around a standard input stream. The access methods are virtual so that subclasses can transparently provide decoding of an encoded stream. Data lengths for Seek and Read calls refer to the length of the input data. The actual length in the stream may differ for subclasses that implement an encoding scheme.

To create an instance of class vtkInputStream, simply invoke its constructor as follows

```
obj = vtkInputStream
```

37.44.2 Methods

The class vtkInputStream has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInputStream class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInputStream = obj.NewInstance ()`
- `vtkInputStream = obj.SafeDownCast (vtkObject o)`
- `obj.StartReading ()` - Called after the stream position has been set by the caller, but before any Seek or Read calls. The stream position should not be adjusted by the caller until after an EndReading call.
- `int = obj.Seek (long offset)` - Seek to the given offset in the input data. Returns 1 for success, 0 for failure.
- `long = obj.Read (string data, long length)` - Read input data of the given length. Returns amount actually read.
- `long = obj.Read (string data, long length)` - Read input data of the given length. Returns amount actually read.
- `obj.EndReading ()` - Called after all desired calls to Seek and Read have been made. After this call, the caller is free to change the position of the stream. Additional reads should not be done until after another call to StartReading.

37.45 vtkIVWriter

37.45.1 Usage

vtkIVWriter is a concrete subclass of vtkWriter that writes OpenInventor 2.0 files.

To create an instance of class vtkIVWriter, simply invoke its constructor as follows

```
obj = vtkIVWriter
```

37.45.2 Methods

The class vtkIVWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIVWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIVWriter = obj.NewInstance ()`
- `vtkIVWriter = obj.SafeDownCast (vtkObject o)`

37.46 vtkJPEGReader

37.46.1 Usage

vtkJPEGReader is a source object that reads JPEG files. It should be able to read most any JPEG file

To create an instance of class vtkJPEGReader, simply invoke its constructor as follows

```
obj = vtkJPEGReader
```

37.46.2 Methods

The class vtkJPEGReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkJPEGReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkJPEGReader = obj.NewInstance ()`
- `vtkJPEGReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string fname)` - Is the given file a JPEG file?
- `string = obj.GetFileExtensions ()` - Return a descriptive name for the file format that might be useful in a GUI.
- `string = obj.GetDescriptiveName ()` - Return a descriptive name for the file format that might be useful in a GUI.

37.47 vtkJPEGWriter

37.47.1 Usage

vtkJPEGWriter writes JPEG files. It supports 1 and 3 component data of unsigned char. It relies on the IJG's libjpeg. Thanks to IJG for supplying a public jpeg IO library.

To create an instance of class vtkJPEGWriter, simply invoke its constructor as follows

```
obj = vtkJPEGWriter
```

37.47.2 Methods

The class vtkJPEGWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkJPEGWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkJPEGWriter = obj.NewInstance ()`
- `vtkJPEGWriter = obj.SafeDownCast (vtkObject o)`
- `obj.Write ()` - The main interface which triggers the writer to start.
- `obj.SetQuality (int )` - Compression quality. 0 = Low quality, 100 = High quality
- `int = obj.GetQualityMinValue ()` - Compression quality. 0 = Low quality, 100 = High quality
- `int = obj.GetQualityMaxValue ()` - Compression quality. 0 = Low quality, 100 = High quality
- `int = obj.GetQuality ()` - Compression quality. 0 = Low quality, 100 = High quality
- `obj.SetProgressive (int )` - Progressive JPEG generation.
- `int = obj.GetProgressive ()` - Progressive JPEG generation.
- `obj.ProgressiveOn ()` - Progressive JPEG generation.
- `obj.ProgressiveOff ()` - Progressive JPEG generation.
- `obj.SetWriteToMemory (int )` - Write the image to memory (a vtkUnsignedCharArray)
- `int = obj.GetWriteToMemory ()` - Write the image to memory (a vtkUnsignedCharArray)
- `obj.WriteToMemoryOn ()` - Write the image to memory (a vtkUnsignedCharArray)
- `obj.WriteToMemoryOff ()` - Write the image to memory (a vtkUnsignedCharArray)
- `obj.SetResult (vtkUnsignedCharArray )` - When writing to memory this is the result, it will be NULL until the data is written the first time
- `vtkUnsignedCharArray = obj.GetResult ()` - When writing to memory this is the result, it will be NULL until the data is written the first time

37.48 vtkMaterialLibrary

37.48.1 Usage

This class provides the Material XMLs. .SECTION Thanks Shader support in VTK includes key contributions by Gary Templet at Sandia National Labs.

To create an instance of class vtkMaterialLibrary, simply invoke its constructor as follows

```
obj = vtkMaterialLibrary
```

37.48.2 Methods

The class vtkMaterialLibrary has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMaterialLibrary class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMaterialLibrary = obj.NewInstance ()`
- `vtkMaterialLibrary = obj.SafeDownCast (vtkObject o)`

37.49 vtkMCubesReader

37.49.1 Usage

vtkMCubesReader is a source object that reads binary marching cubes files. (Marching cubes is an isosurfacing technique that generates many triangles.) The binary format is supported by W. Lorensen's marching cubes program (and the vtkSliceCubes object). The format repeats point coordinates, so this object will merge the points with a vtkLocator object. You can choose to supply the vtkLocator or use the default.

To create an instance of class vtkMCubesReader, simply invoke its constructor as follows

```
obj = vtkMCubesReader
```

37.49.2 Methods

The class vtkMCubesReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMCubesReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMCubesReader = obj.NewInstance ()`
- `vtkMCubesReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name of marching cubes file.
- `string = obj.GetFileName ()` - Specify file name of marching cubes file.
- `obj.SetLimitsFileName (string )` - Set / get the file name of the marching cubes limits file.
- `string = obj.GetLimitsFileName ()` - Set / get the file name of the marching cubes limits file.
- `obj.SetHeaderSize (int )` - Specify a header size if one exists. The header is skipped and not used at this time.
- `int = obj.GetHeaderSizeMinValue ()` - Specify a header size if one exists. The header is skipped and not used at this time.
- `int = obj.GetHeaderSizeMaxValue ()` - Specify a header size if one exists. The header is skipped and not used at this time.
- `int = obj.GetHeaderSize ()` - Specify a header size if one exists. The header is skipped and not used at this time.

- `obj.SetFlipNormals (int )` - Specify whether to flip normals in opposite direction. Flipping ONLY changes the direction of the normal vector. Contrast this with flipping in vtkPolyDataNormals which flips both the normal and the cell point order.

- `int = obj.GetFlipNormals ()` - Specify whether to flip normals in opposite direction. Flipping ONLY changes the direction of the normal vector. Contrast this with flipping in vtkPolyDataNormals which flips both the normal and the cell point order.

- `obj.FlipNormalsOn ()` - Specify whether to flip normals in opposite direction. Flipping ONLY changes the direction of the normal vector. Contrast this with flipping in vtkPolyDataNormals which flips both the normal and the cell point order.

- `obj.FlipNormalsOff ()` - Specify whether to flip normals in opposite direction. Flipping ONLY changes the direction of the normal vector. Contrast this with flipping in vtkPolyDataNormals which flips both the normal and the cell point order.

- `obj.SetNormals (int )` - Specify whether to read normals.

- `int = obj.GetNormals ()` - Specify whether to read normals.

- `obj.NormalsOn ()` - Specify whether to read normals.

- `obj.NormalsOff ()` - Specify whether to read normals.

- `obj.SetDataByteOrderToBigEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.

- `obj.SetDataByteOrderToLittleEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.

- `int = obj.GetDataByteOrder ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.

- `obj.SetDataByteOrder (int )` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.

- `string = obj.GetDataByteOrderAsString ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.
- `obj.SetSwapBytes (int )` - Turn on/off byte swapping.
- `int = obj.GetSwapBytes ()` - Turn on/off byte swapping.
- `obj.SwapBytesOn ()` - Turn on/off byte swapping.
- `obj.SwapBytesOff ()` - Turn on/off byte swapping.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Set / get a spatial locator for merging points. By default, an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified.
- `long = obj.GetMTime ()` - Return the mtime also considering the locator.

37.50 vtkMCubesWriter

37.50.1 Usage

vtkMCubesWriter is a polydata writer that writes binary marching cubes files. (Marching cubes is an isosurfacing technique that generates many triangles.) The binary format is supported by W. Lorensen's marching cubes program (and the vtkSliceCubes object). Each triangle is represented by three records, with each record consisting of six single precision floating point numbers representing the a triangle vertex coordinate and vertex normal.

To create an instance of class vtkMCubesWriter, simply invoke its constructor as follows

```
obj = vtkMCubesWriter
```

37.50.2 Methods

The class vtkMCubesWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMCubesWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMCubesWriter = obj.NewInstance ()`
- `vtkMCubesWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetLimitsFileName (string )` - Set/get file name of marching cubes limits file.
- `string = obj.GetLimitsFileName ()` - Set/get file name of marching cubes limits file.

37.51 vtkMedicalImageProperties

37.51.1 Usage

vtkMedicalImageProperties is a helper class that can be used by medical image readers and applications to encapsulate medical image/acquisition properties. Later on, this should probably be extended to add any user-defined property.

To create an instance of class vtkMedicalImageProperties, simply invoke its constructor as follows

```
obj = vtkMedicalImageProperties
```

37.51.2 Methods

The class vtkMedicalImageProperties has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMedicalImageProperties class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMedicalImageProperties = obj.NewInstance ()`
- `vtkMedicalImageProperties = obj.SafeDownCast (vtkObject o)`
- `obj.Clear ()` - Convenience method to reset all fields to an emptry string/value
- `obj.SetPatientName (string )` - Patient name For ex: DICOM (0010,0010) = DOE,JOHN
- `string = obj.GetPatientName ()` - Patient name For ex: DICOM (0010,0010) = DOE,JOHN
- `obj.SetPatientID (string )` - Patient ID For ex: DICOM (0010,0020) = 1933197
- `string = obj.GetPatientID ()` - Patient ID For ex: DICOM (0010,0020) = 1933197
- `obj.SetPatientAge (string )` - Patient age Format: nnnD, nnW, nnnM or nnnY (eventually nnD, nnW, nnY) with D (day), M (month), W (week), Y (year) For ex: DICOM (0010,1010) = 031Y
- `string = obj.GetPatientAge ()` - Patient age Format: nnnD, nnW, nnnM or nnnY (eventually nnD, nnW, nnY) with D (day), M (month), W (week), Y (year) For ex: DICOM (0010,1010) = 031Y
- `int = obj.GetPatientAgeYear ()`
- `int = obj.GetPatientAgeMonth ()`
- `int = obj.GetPatientAgeWeek ()`
- `int = obj.GetPatientAgeDay ()`
- `obj.SetPatientSex (string )` - Patient sex For ex: DICOM (0010,0040) = M
- `string = obj.GetPatientSex ()` - Patient sex For ex: DICOM (0010,0040) = M
- `obj.SetPatientBirthDate (string )` - Patient birth date Format: yyyymmdd For ex: DICOM (0010,0030) = 19680427
- `string = obj.GetPatientBirthDate ()` - Patient birth date Format: yyyymmdd For ex: DICOM (0010,0030) = 19680427
- `int = obj.GetPatientBirthDateYear ()`
- `int = obj.GetPatientBirthDateMonth ()`

- `int = obj.GetPatientBirthDateDay ()`
- `obj.SetStudyDate (string )` - Study Date Format: yyyymmdd For ex: DICOM (0008,0020) = 20030617
- `string = obj.GetStudyDate ()` - Study Date Format: yyyymmdd For ex: DICOM (0008,0020) = 20030617
- `obj.SetAcquisitionDate (string )` - Acquisition Date Format: yyyymmdd For ex: DICOM (0008,0022) = 20030617
- `string = obj.GetAcquisitionDate ()` - Acquisition Date Format: yyyymmdd For ex: DICOM (0008,0022) = 20030617
- `int = obj.GetAcquisitionDateYear ()`
- `int = obj.GetAcquisitionDateMonth ()`
- `int = obj.GetAcquisitionDateDay ()`
- `obj.SetStudyTime (string )` - Study Time Format: hhmmss.frac (any trailing component(s) can be ommited) For ex: DICOM (0008,0030) = 162552.0705 or 230012, or 0012
- `string = obj.GetStudyTime ()` - Study Time Format: hhmmss.frac (any trailing component(s) can be ommited) For ex: DICOM (0008,0030) = 162552.0705 or 230012, or 0012
- `obj.SetAcquisitionTime (string )` - Acquisition time Format: hhmmss.frac (any trailing component(s) can be ommited) For ex: DICOM (0008,0032) = 162552.0705 or 230012, or 0012
- `string = obj.GetAcquisitionTime ()` - Acquisition time Format: hhmmss.frac (any trailing component(s) can be ommited) For ex: DICOM (0008,0032) = 162552.0705 or 230012, or 0012
- `obj.SetImageDate (string )` - Image Date aka Content Date Format: yyyymmdd For ex: DICOM (0008,0023) = 20030617
- `string = obj.GetImageDate ()` - Image Date aka Content Date Format: yyyymmdd For ex: DICOM (0008,0023) = 20030617
- `int = obj.GetImageDateYear ()`
- `int = obj.GetImageDateMonth ()`
- `int = obj.GetImageDateDay ()`
- `obj.SetImageTime (string )` - Image Time Format: hhmmss.frac (any trailing component(s) can be ommited) For ex: DICOM (0008,0033) = 162552.0705 or 230012, or 0012
- `string = obj.GetImageTime ()` - Image Time Format: hhmmss.frac (any trailing component(s) can be ommited) For ex: DICOM (0008,0033) = 162552.0705 or 230012, or 0012
- `obj.SetImageNumber (string )` - Image number For ex: DICOM (0020,0013) = 1
- `string = obj.GetImageNumber ()` - Image number For ex: DICOM (0020,0013) = 1
- `obj.SetSeriesNumber (string )` - Series number For ex: DICOM (0020,0011) = 902
- `string = obj.GetSeriesNumber ()` - Series number For ex: DICOM (0020,0011) = 902
- `obj.SetSeriesDescription (string )` - Series Description User provided description of the Series For ex: DICOM (0008,103e) = SCOUT
- `string = obj.GetSeriesDescription ()` - Series Description User provided description of the Series For ex: DICOM (0008,103e) = SCOUT

- `obj.SetStudyID (string )` - Study ID For ex: DICOM (0020,0010) = 37481
- `string = obj.GetStudyID ()` - Study ID For ex: DICOM (0020,0010) = 37481
- `obj.SetStudyDescription (string )` - Study description For ex: DICOM (0008,1030) = BRAIN/C-SP/FACIAL
- `string = obj.GetStudyDescription ()` - Study description For ex: DICOM (0008,1030) = BRAIN/C-SP/FACIAL
- `obj.SetModality (string )` - Modality For ex: DICOM (0008,0060)= CT
- `string = obj.GetModality ()` - Modality For ex: DICOM (0008,0060)= CT
- `obj.SetManufacturer (string )` - Manufacturer For ex: DICOM (0008,0070) = Siemens
- `string = obj.GetManufacturer ()` - Manufacturer For ex: DICOM (0008,0070) = Siemens
- `obj.SetManufacturerModelName (string )` - Manufacturer's Model Name For ex: DICOM (0008,1090) = LightSpeed QX/i
- `string = obj.GetManufacturerModelName ()` - Manufacturer's Model Name For ex: DICOM (0008,1090) = LightSpeed QX/i
- `obj.SetStationName (string )` - Station Name For ex: DICOM (0008,1010) = LSPD_OC8
- `string = obj.GetStationName ()` - Station Name For ex: DICOM (0008,1010) = LSPD_OC8
- `obj.SetInstitutionName (string )` - Institution Name For ex: DICOM (0008,0080) = FooCity Medical Center
- `string = obj.GetInstitutionName ()` - Institution Name For ex: DICOM (0008,0080) = FooCity Medical Center
- `obj.SetConvolutionKernel (string )` - Convolution Kernel (or algorithm used to reconstruct the data) For ex: DICOM (0018,1210) = Bone
- `string = obj.GetConvolutionKernel ()` - Convolution Kernel (or algorithm used to reconstruct the data) For ex: DICOM (0018,1210) = Bone
- `obj.SetSliceThickness (string )` - Slice Thickness (Nominal reconstructed slice thickness, in mm) For ex: DICOM (0018,0050) = 0.273438
- `string = obj.GetSliceThickness ()` - Slice Thickness (Nominal reconstructed slice thickness, in mm) For ex: DICOM (0018,0050) = 0.273438
- `double = obj.GetSliceThicknessAsDouble ()` - Slice Thickness (Nominal reconstructed slice thickness, in mm) For ex: DICOM (0018,0050) = 0.273438
- `obj.SetKVP (string )` - Peak kilo voltage output of the (x-ray) generator used For ex: DICOM (0018,0060) = 120
- `string = obj.GetKVP ()` - Peak kilo voltage output of the (x-ray) generator used For ex: DICOM (0018,0060) = 120
- `obj.SetGantryTilt (string )` - Gantry/Detector tilt (Nominal angle of tilt in degrees of the scanning gantry.) For ex: DICOM (0018,1120) = 15
- `string = obj.GetGantryTilt ()` - Gantry/Detector tilt (Nominal angle of tilt in degrees of the scanning gantry.) For ex: DICOM (0018,1120) = 15

- `double = obj.GetGantryTiltAsDouble ()` - Gantry/Detector tilt (Nominal angle of tilt in degrees of the scanning gantry.) For ex: DICOM (0018,1120) = 15
- `obj.SetEchoTime (string )` - Echo Time (Time in ms between the middle of the excitation pulse and the peak of the echo produced) For ex: DICOM (0018,0081) = 105
- `string = obj.GetEchoTime ()` - Echo Time (Time in ms between the middle of the excitation pulse and the peak of the echo produced) For ex: DICOM (0018,0081) = 105
- `obj.SetEchoTrainLength (string )` - Echo Train Length (Number of lines in k-space acquired per excitation per image) For ex: DICOM (0018,0091) = 35
- `string = obj.GetEchoTrainLength ()` - Echo Train Length (Number of lines in k-space acquired per excitation per image) For ex: DICOM (0018,0091) = 35
- `obj.SetRepetitionTime (string )` - Repetition Time The period of time in msec between the beginning of a pulse sequence and the beginning of the succeeding (essentially identical) pulse sequence. For ex: DICOM (0018,0080) = 2040
- `string = obj.GetRepetitionTime ()` - Repetition Time The period of time in msec between the beginning of a pulse sequence and the beginning of the succeeding (essentially identical) pulse sequence. For ex: DICOM (0018,0080) = 2040
- `obj.SetExposureTime (string )` - Exposure time (time of x-ray exposure in msec) For ex: DICOM (0018,1150) = 5
- `string = obj.GetExposureTime ()` - Exposure time (time of x-ray exposure in msec) For ex: DICOM (0018,1150) = 5
- `obj.SetXRayTubeCurrent (string )` - X-ray tube current (in mA) For ex: DICOM (0018,1151) = 400
- `string = obj.GetXRayTubeCurrent ()` - X-ray tube current (in mA) For ex: DICOM (0018,1151) = 400
- `obj.SetExposure (string )` - Exposure (The exposure expressed in mAs, for example calculated from Exposure Time and X-ray Tube Current) For ex: DICOM (0018,1152) = 114
- `string = obj.GetExposure ()` - Exposure (The exposure expressed in mAs, for example calculated from Exposure Time and X-ray Tube Current) For ex: DICOM (0018,1152) = 114
- `obj.SetDirectionCosine (double , double , double , double , double , double )` - Get the direction cosine (default to 1,0,0,0,1,0)
- `obj.SetDirectionCosine (double  a[6])` - Get the direction cosine (default to 1,0,0,0,1,0)
- `double = obj. GetDirectionCosine ()` - Get the direction cosine (default to 1,0,0,0,1,0)
- `obj.AddUserDefinedValue (string name, string value)`
- `string = obj.GetUserDefinedValue (string name)`
- `int = obj.GetNumberOfUserDefinedValues ()`
- `string = obj.GetUserDefinedNameByIndex (int idx)`
- `string = obj.GetUserDefinedValueByIndex (int idx)`
- `obj.RemoveAllUserDefinedValues ()`

- `int = obj.AddWindowLevelPreset (double w, double l)` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `obj.RemoveWindowLevelPreset (double w, double l)` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `obj.RemoveAllWindowLevelPresets ()` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `int = obj.GetNumberOfWindowLevelPresets ()` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `int = obj.HasWindowLevelPreset (double w, double l)` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `int = obj.GetWindowLevelPresetIndex (double w, double l)` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `int = obj.GetNthWindowLevelPreset (int idx, double w, double l)` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `double = obj.GetNthWindowLevelPreset (int idx)` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `obj.SetNthWindowLevelPresetComment (int idx, string comment)` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `string = obj.GetNthWindowLevelPresetComment (int idx)` - Add/Remove/Query the window/level presets that may have been associated to a medical image. Window is also known as 'width', level is also known as 'center'. The same window/level pair can not be added twice. As a convenience, a comment (aka Explanation) can be associated to a preset. For ex:

```
DICOM Window Center (0028,1050) = 00045\000470
DICOM Window Width  (0028,1051) = 0106\03412
DICOM Window Center Width Explanation (0028,1055) = WINDOW1\WINDOW2
```

- `string = obj.GetInstanceUIDFromSliceID (int volumeidx, int sliceid)` - Mapping from a sliceidx within a volumeidx into a DICOM Instance UID Some DICOM reader can populate this structure so that later on from a slice index in a vtkImageData volume we can backtrack and find out which 2d slice it was coming from
- `obj.SetInstanceUIDFromSliceID (int volumeidx, int sliceid, string uid)` - Mapping from a sliceidx within a volumeidx into a DICOM Instance UID Some DICOM reader can populate this structure so that later on from a slice index in a vtkImageData volume we can backtrack and find out which 2d slice it was coming from
- `int = obj.GetOrientationType (int volumeidx)`

- `obj.SetOrientationType (int volumeidx, int orientation)`
- `obj.DeepCopy (vtkMedicalImageProperties p)` - Copy the contents of p to this instance.

37.52 vtkMedicalImageReader2

37.52.1 Usage

vtkMedicalImageReader2 is a parent class for medical image readers. It provides a place to store patient information that may be stored in the image header.

To create an instance of class vtkMedicalImageReader2, simply invoke its constructor as follows

```
obj = vtkMedicalImageReader2
```

37.52.2 Methods

The class vtkMedicalImageReader2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMedicalImageReader2 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMedicalImageReader2 = obj.NewInstance ()`
- `vtkMedicalImageReader2 = obj.SafeDownCast (vtkObject o)`
- `vtkMedicalImageProperties = obj.GetMedicalImageProperties ()` - Get the medical image properties object
- `obj.SetPatientName (string )` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `string = obj.GetPatientName ()` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `obj.SetPatientID (string )` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `string = obj.GetPatientID ()` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `obj.SetDate (string )` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `string = obj.GetDate ()` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `obj.SetSeries (string )` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `string = obj.GetSeries ()` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `obj.SetStudy (string )` - For backward compatibility, propagate calls to the MedicalImageProperties object.

- `string = obj.GetStudy ()` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `obj.SetImageNumber (string )` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `string = obj.GetImageNumber ()` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `obj.SetModality (string )` - For backward compatibility, propagate calls to the MedicalImageProperties object.
- `string = obj.GetModality ()` - For backward compatibility, propagate calls to the MedicalImageProperties object.

37.53 vtkMetaImageReader

37.53.1 Usage

One of the formats for which a reader is already available in the toolkit is the MetaImage file format. This is a fairly simple yet powerful format consisting of a text header and a binary data section. The following instructions describe how you can write a MetaImage header for the data that you download from the BrainWeb page.

The minimal structure of the MetaImage header is the following:

NDims = 3 DimSize = 181 217 181 ElementType = MET_UCHAR ElementSpacing = 1.0 1.0 1.0 ElementByteOrderMSB = False ElementDataFile = brainweb1.raw

* NDims indicate that this is a 3D image. ITK can handle images of arbitrary dimension. * DimSize indicates the size of the volume in pixels along each direction. * ElementType indicate the primitive type used for pixels. In this case is "unsigned char", implying that the data is digitized in 8 bits / pixel. * ElementSpacing indicates the physical separation between the center of one pixel and the center of the next pixel along each direction in space. The units used are millimeters. * ElementByteOrderMSB indicates is the data is encoded in little or big endian order. You might want to play with this value when moving data between different computer platforms. * ElementDataFile is the name of the file containing the raw binary data of the image. This file must be in the same directory as the header.

MetaImage headers are expected to have extension: ".mha" or ".mhd"

Once you write this header text file, it should be possible to read the image into your ITK based application using the itk::FileIOToImageFilter class.

To create an instance of class vtkMetaImageReader, simply invoke its constructor as follows

```
obj = vtkMetaImageReader
```

37.53.2 Methods

The class vtkMetaImageReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMetaImageReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMetaImageReader = obj.NewInstance ()`
- `vtkMetaImageReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileExtensions ()`

- `string = obj.GetDescriptiveName ()`
- `int = obj.GetWidth ()`
- `int = obj.GetHeight ()`
- `int = obj.GetNumberOfComponents ()`
- `int = obj.GetPixelRepresentation ()`
- `int = obj.GetDataByteOrder (void )`
- `double = obj.GetRescaleSlope ()`
- `double = obj.GetRescaleOffset ()`
- `int = obj.GetBitsAllocated ()`
- `string = obj.GetDistanceUnits ()`
- `string = obj.GetAnatomicalOrientation ()`
- `double = obj.GetGantryAngle ()`
- `string = obj.GetPatientName ()`
- `string = obj.GetPatientID ()`
- `string = obj.GetDate ()`
- `string = obj.GetSeries ()`
- `string = obj.GetImageNumber ()`
- `string = obj.GetModality ()`
- `string = obj.GetStudyID ()`
- `string = obj.GetStudyUID ()`
- `string = obj.GetTransferSyntaxUID ()`
- `int = obj.CanReadFile (string name)` - Test whether the file with the given name can be read by this reader.

37.54 vtkMetaImageWriter

37.54.1 Usage

One of the formats for which a reader is already available in the toolkit is the MetaImage file format. This is a fairly simple yet powerful format consisting of a text header and a binary data section. The following instructions describe how you can write a MetaImage header for the data that you download from the BrainWeb page.

The minimal structure of the MetaImage header is the following:

NDims = 3 DimSize = 181 217 181 ElementType = MET_UCHAR ElementSpacing = 1.0 1.0 1.0 ElementByteOrderMSB = False ElementDataFile = brainweb1.raw

* NDims indicate that this is a 3D image. ITK can handle images of arbitrary dimension. * DimSize indicates the size of the volume in pixels along each direction. * ElementType indicate the primitive type used for pixels. In this case is "unsigned char", implying that the data is digitized in 8 bits / pixel. * ElementSpacing indicates the physical separation between the center of one pixel and the center of the next pixel along each direction in space. The units used are millimeters. * ElementByteOrderMSB indicates is

the data is encoded in little or big endian order. You might want to play with this value when moving data between different computer platforms. * ElementDataFile is the name of the file containing the raw binary data of the image. This file must be in the same directory as the header.

MetaImage headers are expected to have extension: ".mha" or ".mhd"

Once you write this header text file, it should be possible to read the image into your ITK based application using the itk::FileIOToImageFilter class.

To create an instance of class vtkMetaImageWriter, simply invoke its constructor as follows

```
obj = vtkMetaImageWriter
```

37.54.2 Methods

The class vtkMetaImageWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMetaImageWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMetaImageWriter = obj.NewInstance ()`
- `vtkMetaImageWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string fname)` - Specify file name of meta file
- `string = obj.GetFileName ()` - Specify the file name of the raw image data.
- `obj.SetRAWFileName (string fname)` - Specify the file name of the raw image data.
- `string = obj.GetRAWFileName ()` - Specify the file name of the raw image data.
- `obj.SetCompression (bool compress)`
- `bool = obj.GetCompression (void )`
- `obj.Write ()`

37.55 vtkMFIXReader

37.55.1 Usage

vtkMFIXReader creates an unstructured grid dataset. It reads a restart file and a set of sp files. The restart file contains the mesh information. MFIX meshes are either cylindrical or rectilinear, but this reader will convert them to an unstructured grid. The sp files contain transient data for the cells. Each sp file has one or more variables stored inside it.

To create an instance of class vtkMFIXReader, simply invoke its constructor as follows

```
obj = vtkMFIXReader
```

37.55.2 Methods

The class vtkMFIXReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMFIXReader class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkMFIXReader = obj.NewInstance ()`
- `vtkMFIXReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify the file name of the MFIX Restart data file to read.
- `string = obj.GetFileName ()` - Specify the file name of the MFIX Restart data file to read.
- `int = obj.GetNumberOfCells ()` - Get the total number of cells. The number of cells is only valid after a successful read of the data file is performed.
- `int = obj.GetNumberOfPoints ()` - Get the total number of nodes. The number of nodes is only valid after a successful read of the data file is performed.
- `int = obj.GetNumberOfCellFields ()` - Get the number of data components at the nodes and cells.
- `obj.SetTimeStep (int )` - Which TimeStep to read.
- `int = obj.GetTimeStep ()` - Which TimeStep to read.
- `int = obj.GetNumberOfTimeSteps ()` - Returns the number of timesteps.
- `int = obj. GetTimeStepRange ()` - Which TimeStepRange to read
- `obj.SetTimeStepRange (int , int )` - Which TimeStepRange to read
- `obj.SetTimeStepRange (int  a[2])` - Which TimeStepRange to read
- `int = obj.GetNumberOfCellArrays (void )`
- `string = obj.GetCellArrayName (int index)` - Get the name of the cell array with the given index in the input.
- `int = obj.GetCellArrayStatus (string name)` - Get/Set whether the cell array with the given name is to be read.
- `obj.SetCellArrayStatus (string name, int status)` - Get/Set whether the cell array with the given name is to be read.
- `obj.DisableAllCellArrays ()` - Turn on/off all cell arrays.
- `obj.EnableAllCellArrays ()` - Turn on/off all cell arrays.
- `obj.GetCellDataRange (int cellComp, int index, float min, float max)` - Get the range of cell data.

37.56 vtkMINCImageAttributes

37.56.1 Usage

This class provides methods to access all of the information contained in the MINC header. If you read a MINC file into VTK and then write it out again, you can use writer-¿SetImageAttributes(reader-¿GetImageAttributes) to ensure that all of the medical information contained in the file is tranferred from the reader to the writer. If you want to change any of the header information, you must use ShallowCopy to make a copy of the reader's attributes and then modify only the copy.

To create an instance of class vtkMINCImageAttributes, simply invoke its constructor as follows

```
obj = vtkMINCImageAttributes
```

37.56.2 Methods

The class vtkMINCImageAttributes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMINCImageAttributes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMINCImageAttributes = obj.NewInstance ()`
- `vtkMINCImageAttributes = obj.SafeDownCast (vtkObject o)`
- `obj.Reset ()` - Reset all the attributes in preparation for loading new information.
- `obj.SetName (string )` - Get the name of the image, not including the path or the extension. This is only needed for printing the header and there is usually no need to set it.
- `string = obj.GetName ()` - Get the name of the image, not including the path or the extension. This is only needed for printing the header and there is usually no need to set it.
- `obj.SetDataType (int )` - Get the image data type, as stored on disk. This information is useful if the file was converted to floating-point when it was loaded. When writing a file from float or double image data, you can use this method to prescribe the output type.
- `int = obj.GetDataType ()` - Get the image data type, as stored on disk. This information is useful if the file was converted to floating-point when it was loaded. When writing a file from float or double image data, you can use this method to prescribe the output type.
- `obj.AddDimension (string dimension)` - Add the names of up to five dimensions. The ordering of these dimensions will determine the dimension order of the file. If no DimensionNames are set, the writer will set the dimension order of the file to be the same as the dimension order in memory.
- `obj.AddDimension (string dimension, vtkIdType length)` - Add the names of up to five dimensions. The ordering of these dimensions will determine the dimension order of the file. If no DimensionNames are set, the writer will set the dimension order of the file to be the same as the dimension order in memory.
- `vtkStringArray = obj.GetDimensionNames ()` - Get the dimension names. The dimension names are same order as written in the file, starting with the slowest-varying dimension. Use this method to get the array if you need to change "space" dimensions to "frequency" after performing a Fourier transform.
- `vtkIdTypeArray = obj.GetDimensionLengths ()` - Get the lengths of all the dimensions. The dimension lengths are informative, the vtkMINCImageWriter does not look at these values but instead uses the dimension sizes of its input.
- `vtkStringArray = obj.GetVariableNames ()` - Get the names of all the variables.
- `vtkStringArray = obj.GetAttributeNames (string variable)` - List the attribute names for a variable. Set the variable to the empty string to get a list of the global attributes.
- `obj.SetImageMin (vtkDoubleArray imageMin)` - Get the image min and max arrays. These are set by the reader, but they aren't used by the writer except to compute the full real data range of the original file.
- `obj.SetImageMax (vtkDoubleArray imageMax)` - Get the image min and max arrays. These are set by the reader, but they aren't used by the writer except to compute the full real data range of the original file.

- `vtkDoubleArray = obj.GetImageMin ()` - Get the image min and max arrays. These are set by the reader, but they aren't used by the writer except to compute the full real data range of the original file.
- `vtkDoubleArray = obj.GetImageMax ()` - Get the image min and max arrays. These are set by the reader, but they aren't used by the writer except to compute the full real data range of the original file.
- `int = obj.GetNumberOfImageMinMaxDimensions ()` - Get the number of ImageMinMax dimensions.
- `obj.SetNumberOfImageMinMaxDimensions (int )` - Get the number of ImageMinMax dimensions.
- `int = obj.HasAttribute (string variable, string attribute)` - Check to see if a particular attribute exists.
- `obj.SetAttributeValueAsArray (string variable, string attribute, vtkDataArray array)` - Set attribute values for a variable as a vtkDataArray. Set the variable to the empty string to access global attributes.
- `vtkDataArray = obj.GetAttributeValueAsArray (string variable, string attribute)` - Set attribute values for a variable as a vtkDataArray. Set the variable to the empty string to access global attributes.
- `obj.SetAttributeValueAsString (string variable, string attribute, string value)` - Set an attribute value as a string. Set the variable to the empty string to access global attributes. If you specify a variable that does not exist, it will be created.
- `string = obj.GetAttributeValueAsString (string variable, string attribute)` - Set an attribute value as a string. Set the variable to the empty string to access global attributes. If you specify a variable that does not exist, it will be created.
- `obj.SetAttributeValueAsInt (string variable, string attribute, int value)` - Set an attribute value as an int. Set the variable to the empty string to access global attributes. If you specify a variable that does not exist, it will be created.
- `int = obj.GetAttributeValueAsInt (string variable, string attribute)` - Set an attribute value as an int. Set the variable to the empty string to access global attributes. If you specify a variable that does not exist, it will be created.
- `obj.SetAttributeValueAsDouble (string variable, string attribute, double value)` - Set an attribute value as a double. Set the variable to the empty string to access global attributes. If you specify a variable that does not exist, it will be created.
- `double = obj.GetAttributeValueAsDouble (string variable, string attribute)` - Set an attribute value as a double. Set the variable to the empty string to access global attributes. If you specify a variable that does not exist, it will be created.
- `int = obj.ValidateAttribute (string varname, string attname, vtkDataArray array)` - Validate a particular attribute. This involves checking that the attribute is a MINC standard attribute, and checking whether it can be set (as opposed to being set automatically from the image information). The return values is 0 if the attribute is set automatically and therefore should not be copied from here, 1 if this attribute is valid and should be set, and 2 if the attribute is non-standard.
- `obj.ShallowCopy (vtkMINCImageAttributes source)` - Do a shallow copy. This will copy all the attributes from the source. It is much more efficient than a DeepCopy would be, since it only copies pointers to the attribute values instead of copying the arrays themselves. You must use this method to make a copy if you want to modify any MINC attributes from a MINCReader before you pass them to a MINCWriter.

- `obj.FindValidRange (double range[2])` - Find the valid range of the data from the information stored in the attributes.
- `obj.FindImageRange (double range[2])` - Find the image range of the data from the information stored in the attributes.
- `obj.PrintFileHeader ()` - A diagnostic function. Print the header of the file in the same format as ncdump or mincheader.

37.57 vtkMINCImageReader

37.57.1 Usage

MINC is a NetCDF-based medical image file format that was developed at the Montreal Neurological Institute in 1992. This class will read a MINC file into VTK, rearranging the data to match the VTK x, y, and z dimensions, and optionally rescaling real-valued data to VTK_FLOAT if RescaleRealValuesOn() is set. If RescaleRealValues is off, then the data will be stored in its original data type and the GetRescaleSlope(), GetRescaleIntercept() method can be used to retrieve global rescaling parameters. If the original file had a time dimension, the SetTimeStep() method can be used to specify a time step to read. All of the original header information can be accessed though the GetImageAttributes() method.

To create an instance of class vtkMINCImageReader, simply invoke its constructor as follows

```
obj = vtkMINCImageReader
```

37.57.2 Methods

The class vtkMINCImageReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMINCImageReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMINCImageReader = obj.NewInstance ()`
- `vtkMINCImageReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string name)` - Set the file name.
- `string = obj.GetFileExtensions ()` - Get the name of this file format.
- `string = obj.GetDescriptiveName ()` - Test whether the specified file can be read.
- `int = obj.CanReadFile (string name)` - Test whether the specified file can be read.
- `vtkMatrix4x4 = obj.GetDirectionCosines ()` - Get a matrix that describes the orientation of the data. The three columns of the matrix are the direction cosines for the x, y and z dimensions respectively.
- `double = obj.GetRescaleSlope ()` - Get the slope and intercept for rescaling the scalar values to real data values. To convert scalar values to real values, use the equation y = x*RescaleSlope + RescaleIntercept.
- `double = obj.GetRescaleIntercept ()` - Get the slope and intercept for rescaling the scalar values to real data values. To convert scalar values to real values, use the equation y = x*RescaleSlope + RescaleIntercept.

- `obj.SetRescaleRealValues (int )` - Rescale real data values to float. If this is done, the RescaleSlope and RescaleIntercept will be set to 1 and 0 respectively. This is off by default.
- `obj.RescaleRealValuesOn ()` - Rescale real data values to float. If this is done, the RescaleSlope and RescaleIntercept will be set to 1 and 0 respectively. This is off by default.
- `obj.RescaleRealValuesOff ()` - Rescale real data values to float. If this is done, the RescaleSlope and RescaleIntercept will be set to 1 and 0 respectively. This is off by default.
- `int = obj.GetRescaleRealValues ()` - Rescale real data values to float. If this is done, the RescaleSlope and RescaleIntercept will be set to 1 and 0 respectively. This is off by default.
- `double = obj.GetDataRange ()` - Get the scalar range of the output from the information in the file header. This is more efficient that computing the scalar range, but in some cases the MINC file stores an incorrect valid_range and the DataRange will be incorrect.
- `obj.GetDataRange (double range[2])` - Get the scalar range of the output from the information in the file header. This is more efficient that computing the scalar range, but in some cases the MINC file stores an incorrect valid_range and the DataRange will be incorrect.
- `int = obj.GetNumberOfTimeSteps ()` - Get the number of time steps in the file.
- `obj.SetTimeStep (int )` - Set the time step to read.
- `int = obj.GetTimeStep ()` - Set the time step to read.
- `vtkMINCImageAttributes = obj.GetImageAttributes ()` - Get the image attributes, which contain patient information and other useful metadata.

37.58 vtkMINCImageWriter

37.58.1 Usage

MINC is a NetCDF-based medical image file format that was developed at the Montreal Neurological Institute in 1992. The data is written slice-by-slice, and this writer is therefore suitable for streaming MINC data that is larger than the memory size through VTK. This writer can also produce files with up to 4 dimensions, where the fourth dimension is provided by using AddInput() to specify multiple input data sets. If you want to set header information for the file, you must supply a vtkMINCImageAttributes

To create an instance of class vtkMINCImageWriter, simply invoke its constructor as follows

```
obj = vtkMINCImageWriter
```

37.58.2 Methods

The class vtkMINCImageWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMINCImageWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMINCImageWriter = obj.NewInstance ()`
- `vtkMINCImageWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileExtensions ()` - Get the name of this file format.
- `string = obj.GetDescriptiveName ()` - Set the file name.

- `obj.SetFileName (string name)` - Set the file name.

- `obj.Write ()` - Write the data. This will attempt to stream the data slice-by-slice through the pipeline and out to the file, unless the whole extent of the input has already been updated.

- `obj.SetDirectionCosines (vtkMatrix4x4 matrix)` - Set a matrix that describes the orientation of the data. The three columns of this matrix should give the unit-vector directions for the VTK x, y and z dimensions respectively. The writer will use this information to determine how to map the VTK dimensions to the canonical MINC dimensions, and if necessary, the writer will re-order one or more dimensions back-to-front to ensure that no MINC dimension ends up with a direction cosines vector whose dot product with the canonical unit vector for that dimension is negative.

- `vtkMatrix4x4 = obj.GetDirectionCosines ()` - Set a matrix that describes the orientation of the data. The three columns of this matrix should give the unit-vector directions for the VTK x, y and z dimensions respectively. The writer will use this information to determine how to map the VTK dimensions to the canonical MINC dimensions, and if necessary, the writer will re-order one or more dimensions back-to-front to ensure that no MINC dimension ends up with a direction cosines vector whose dot product with the canonical unit vector for that dimension is negative.

- `obj.SetRescaleSlope (double )` - Set the slope and intercept for rescaling the intensities. The default values are zero, which indicates to the reader that no rescaling is to be performed.

- `double = obj.GetRescaleSlope ()` - Set the slope and intercept for rescaling the intensities. The default values are zero, which indicates to the reader that no rescaling is to be performed.

- `obj.SetRescaleIntercept (double )` - Set the slope and intercept for rescaling the intensities. The default values are zero, which indicates to the reader that no rescaling is to be performed.

- `double = obj.GetRescaleIntercept ()` - Set the slope and intercept for rescaling the intensities. The default values are zero, which indicates to the reader that no rescaling is to be performed.

- `obj.SetImageAttributes (vtkMINCImageAttributes attributes)` - Set the image attributes, which contain patient information and other useful metadata.

- `vtkMINCImageAttributes = obj.GetImageAttributes ()` - Set the image attributes, which contain patient information and other useful metadata.

- `obj.SetStrictValidation (int )` - Set whether to validate that all variable attributes that have been set are ones that are listed in the MINC standard.

- `obj.StrictValidationOn ()` - Set whether to validate that all variable attributes that have been set are ones that are listed in the MINC standard.

- `obj.StrictValidationOff ()` - Set whether to validate that all variable attributes that have been set are ones that are listed in the MINC standard.

- `int = obj.GetStrictValidation ()` - Set whether to validate that all variable attributes that have been set are ones that are listed in the MINC standard.

- `obj.SetHistoryAddition (string )` - Set a string value to append to the history of the file. This string should describe, briefly, how the file was processed.

- `string = obj.GetHistoryAddition ()` - Set a string value to append to the history of the file. This string should describe, briefly, how the file was processed.

37.59 vtkMoleculeReaderBase

37.59.1 Usage

vtkMoleculeReaderBase is a source object that reads Molecule files The FileName must be specified .SECTION Thanks Dr. Jean M. Favre who developed and contributed this class

To create an instance of class vtkMoleculeReaderBase, simply invoke its constructor as follows

```
obj = vtkMoleculeReaderBase
```

37.59.2 Methods

The class vtkMoleculeReaderBase has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMoleculeReaderBase class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMoleculeReaderBase = obj.NewInstance ()`
- `vtkMoleculeReaderBase = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )`
- `string = obj.GetFileName ()`
- `obj.SetBScale (double )`
- `double = obj.GetBScale ()`
- `obj.SetHBScale (double )`
- `double = obj.GetHBScale ()`
- `int = obj.GetNumberOfAtoms ()`

37.60 vtkMultiBlockPLOT3DReader

37.60.1 Usage

vtkMultiBlockPLOT3DReader is a reader object that reads PLOT3D formatted files and generates structured grid(s) on output. PLOT3D is a computer graphics program designed to visualize the grids and solutions of computational fluid dynamics. Please see the "PLOT3D User's Manual" available from NASA Ames Research Center, Moffett Field CA.

PLOT3D files consist of a grid file (also known as XYZ file), an optional solution file (also known as a Q file), and an optional function file that contains user created data (currently unsupported). The Q file contains solution information as follows: the four parameters free stream mach number (Fsmach), angle of attack (Alpha), Reynolds number (Re), and total integration time (Time). This information is stored in an array called Properties in the FieldData of each output (tuple 0: fsmach, tuple 1: alpha, tuple 2: re, tuple 3: time). In addition, the solution file contains the flow density (scalar), flow momentum (vector), and flow energy (scalar).

The reader can generate additional scalars and vectors (or "functions") from this information. To use vtkMultiBlockPLOT3DReader, you must specify the particular function number for the scalar and vector you want to visualize. This implementation of the reader provides the following functions. The scalar functions are: -1 - don't read or compute any scalars 100 - density 110 - pressure 120 - temperature 130 -

enthalpy 140 - internal energy 144 - kinetic energy 153 - velocity magnitude 163 - stagnation energy 170 - entropy 184 - swirl.

The vector functions are: -1 - don't read or compute any vectors 200 - velocity 201 - vorticity 202 - momentum 210 - pressure gradient.

(Other functions are described in the PLOT3D spec, but only those listed are implemented here.) Note that by default, this reader creates the density scalar (100) and momentum vector (202) as output. (These are just read in from the solution file.) Please note that the validity of computation is a function of this class's gas constants (R, Gamma) and the equations used. They may not be suitable for your computational domain.

Additionally, you can read other data and associate it as a vtkDataArray into the output's point attribute data. Use the method AddFunction() to list all the functions that you'd like to read. AddFunction() accepts an integer parameter that defines the function number.

To create an instance of class vtkMultiBlockPLOT3DReader, simply invoke its constructor as follows

```
obj = vtkMultiBlockPLOT3DReader
```

37.60.2 Methods

The class vtkMultiBlockPLOT3DReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiBlockPLOT3DReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiBlockPLOT3DReader = obj.NewInstance ()`
- `vtkMultiBlockPLOT3DReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string name)` - Set/Get the PLOT3D geometry filename.
- `string = obj.GetFileName ()` - Set/Get the PLOT3D geometry filename.
- `obj.SetXYZFileName (string )` - Set/Get the PLOT3D geometry filename.
- `string = obj.GetXYZFileName ()` - Set/Get the PLOT3D geometry filename.
- `obj.SetQFileName (string )` - Set/Get the PLOT3D solution filename.
- `string = obj.GetQFileName ()` - Set/Get the PLOT3D solution filename.
- `int = obj.GetNumberOfBlocks ()` - This returns the number of outputs this reader will produce. This number is equal to the number of grids in the current file. This method has to be called before getting any output if the number of outputs will be greater than 1 (the first output is always the same). Note that every time this method is invoked, the header file is opened and part of the header is read.
- `int = obj.GetNumberOfGrids ()` - Is the file to be read written in binary format (as opposed to ascii).
- `obj.SetBinaryFile (int )` - Is the file to be read written in binary format (as opposed to ascii).
- `int = obj.GetBinaryFile ()` - Is the file to be read written in binary format (as opposed to ascii).
- `obj.BinaryFileOn ()` - Is the file to be read written in binary format (as opposed to ascii).
- `obj.BinaryFileOff ()` - Is the file to be read written in binary format (as opposed to ascii).

- `obj.SetMultiGrid (int )` - Does the file to be read contain information about number of grids. In some PLOT3D files, the first value contains the number of grids (even if there is only 1). If reading such a file, set this to true.
- `int = obj.GetMultiGrid ()` - Does the file to be read contain information about number of grids. In some PLOT3D files, the first value contains the number of grids (even if there is only 1). If reading such a file, set this to true.
- `obj.MultiGridOn ()` - Does the file to be read contain information about number of grids. In some PLOT3D files, the first value contains the number of grids (even if there is only 1). If reading such a file, set this to true.
- `obj.MultiGridOff ()` - Does the file to be read contain information about number of grids. In some PLOT3D files, the first value contains the number of grids (even if there is only 1). If reading such a file, set this to true.
- `obj.SetHasByteCount (int )` - Were the arrays written with leading and trailing byte counts ? Usually, files written by a fortran program will contain these byte counts whereas the ones written by C/C++ won't.
- `int = obj.GetHasByteCount ()` - Were the arrays written with leading and trailing byte counts ? Usually, files written by a fortran program will contain these byte counts whereas the ones written by C/C++ won't.
- `obj.HasByteCountOn ()` - Were the arrays written with leading and trailing byte counts ? Usually, files written by a fortran program will contain these byte counts whereas the ones written by C/C++ won't.
- `obj.HasByteCountOff ()` - Were the arrays written with leading and trailing byte counts ? Usually, files written by a fortran program will contain these byte counts whereas the ones written by C/C++ won't.
- `obj.SetIBlanking (int )` - Is there iblanking (point visibility) information in the file. If there is iblanking arrays, these will be read and assigned to the PointVisibility array of the output.
- `int = obj.GetIBlanking ()` - Is there iblanking (point visibility) information in the file. If there is iblanking arrays, these will be read and assigned to the PointVisibility array of the output.
- `obj.IBlankingOn ()` - Is there iblanking (point visibility) information in the file. If there is iblanking arrays, these will be read and assigned to the PointVisibility array of the output.
- `obj.IBlankingOff ()` - Is there iblanking (point visibility) information in the file. If there is iblanking arrays, these will be read and assigned to the PointVisibility array of the output.
- `obj.SetTwoDimensionalGeometry (int )` - If only two-dimensional data was written to the file, turn this on.
- `int = obj.GetTwoDimensionalGeometry ()` - If only two-dimensional data was written to the file, turn this on.
- `obj.TwoDimensionalGeometryOn ()` - If only two-dimensional data was written to the file, turn this on.
- `obj.TwoDimensionalGeometryOff ()` - If only two-dimensional data was written to the file, turn this on.
- `obj.SetForceRead (int )` - Try to read a binary file even if the file length seems to be inconsistent with the header information. Use this with caution, if the file length is not the same as calculated from the header. either the file is corrupt or the settings are wrong.

- `int = obj.GetForceRead ()` - Try to read a binary file even if the file length seems to be inconsistent with the header information. Use this with caution, if the file length is not the same as calculated from the header. either the file is corrupt or the settings are wrong.
- `obj.ForceReadOn ()` - Try to read a binary file even if the file length seems to be inconsistent with the header information. Use this with caution, if the file length is not the same as calculated from the header. either the file is corrupt or the settings are wrong.
- `obj.ForceReadOff ()` - Try to read a binary file even if the file length seems to be inconsistent with the header information. Use this with caution, if the file length is not the same as calculated from the header. either the file is corrupt or the settings are wrong.
- `obj.SetByteOrderToBigEndian ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `obj.SetByteOrderToLittleEndian ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `obj.SetByteOrder (int )` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `int = obj.GetByteOrder ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `string = obj.GetByteOrderAsString ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `obj.SetR (double )` - Set/Get the gas constant. Default is 1.0.
- `double = obj.GetR ()` - Set/Get the gas constant. Default is 1.0.
- `obj.SetGamma (double )` - Set/Get the ratio of specific heats. Default is 1.4.
- `double = obj.GetGamma ()` - Set/Get the ratio of specific heats. Default is 1.4.
- `obj.SetUvinf (double )` - Set/Get the x-component of the free-stream velocity. Default is 1.0.
- `double = obj.GetUvinf ()` - Set/Get the x-component of the free-stream velocity. Default is 1.0.
- `obj.SetVvinf (double )` - Set/Get the y-component of the free-stream velocity. Default is 1.0.
- `double = obj.GetVvinf ()` - Set/Get the y-component of the free-stream velocity. Default is 1.0.
- `obj.SetWvinf (double )` - Set/Get the z-component of the free-stream velocity. Default is 1.0.
- `double = obj.GetWvinf ()` - Set/Get the z-component of the free-stream velocity. Default is 1.0.
- `obj.SetScalarFunctionNumber (int num)` - Specify the scalar function to extract. If ==(-1), then no scalar function is extracted.
- `int = obj.GetScalarFunctionNumber ()` - Specify the scalar function to extract. If ==(-1), then no scalar function is extracted.
- `obj.SetVectorFunctionNumber (int num)` - Specify the vector function to extract. If ==(-1), then no vector function is extracted.

- `int = obj.GetVectorFunctionNumber ()` - Specify the vector function to extract. If ==(-1), then no vector function is extracted.
- `obj.AddFunction (int functionNumber)` - Specify additional functions to read. These are placed into the point data as data arrays. Later on they can be used by labeling them as scalars, etc.
- `obj.RemoveFunction (int )` - Specify additional functions to read. These are placed into the point data as data arrays. Later on they can be used by labeling them as scalars, etc.
- `obj.RemoveAllFunctions ()` - Specify additional functions to read. These are placed into the point data as data arrays. Later on they can be used by labeling them as scalars, etc.
- `int = obj.CanReadBinaryFile (string fname)` - Return 1 if the reader can read the given file name. Only meaningful for binary files.

37.61 vtkNetCDFCFReader

37.61.1 Usage

Reads netCDF files that follow the CF convention. Details on this convention can be found at ¡http://cf-pcmdi.llnl.gov/¿.

To create an instance of class vtkNetCDFCFReader, simply invoke its constructor as follows

```
obj = vtkNetCDFCFReader
```

37.61.2 Methods

The class vtkNetCDFCFReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkNetCDFCFReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkNetCDFCFReader = obj.NewInstance ()`
- `vtkNetCDFCFReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetSphericalCoordinates ()` - If on (the default), then 3D data with latitude/longitude dimensions will be read in as curvilinear data shaped like spherical coordinates. If false, then the data will always be read in Cartesian coordinates.
- `obj.SetSphericalCoordinates (int )` - If on (the default), then 3D data with latitude/longitude dimensions will be read in as curvilinear data shaped like spherical coordinates. If false, then the data will always be read in Cartesian coordinates.
- `obj.SphericalCoordinatesOn ()` - If on (the default), then 3D data with latitude/longitude dimensions will be read in as curvilinear data shaped like spherical coordinates. If false, then the data will always be read in Cartesian coordinates.
- `obj.SphericalCoordinatesOff ()` - If on (the default), then 3D data with latitude/longitude dimensions will be read in as curvilinear data shaped like spherical coordinates. If false, then the data will always be read in Cartesian coordinates.

- `double = obj.GetVerticalScale ()` - The scale and bias of the vertical component of spherical coordinates. It is common to write the vertical component with respect to something other than the center of the sphere (for example, the surface). In this case, it might be necessary to scale and/or bias the vertical height. The height will become height*scale + bias. Keep in mind that if the positive attribute of the vertical dimension is down, then the height is negated. By default the scale is 1 and the bias is 0 (that is, no change). The scaling will be adjusted if it results in invalid (negative) vertical values.

- `obj.SetVerticalScale (double )` - The scale and bias of the vertical component of spherical coordinates. It is common to write the vertical component with respect to something other than the center of the sphere (for example, the surface). In this case, it might be necessary to scale and/or bias the vertical height. The height will become height*scale + bias. Keep in mind that if the positive attribute of the vertical dimension is down, then the height is negated. By default the scale is 1 and the bias is 0 (that is, no change). The scaling will be adjusted if it results in invalid (negative) vertical values.

- `double = obj.GetVerticalBias ()` - The scale and bias of the vertical component of spherical coordinates. It is common to write the vertical component with respect to something other than the center of the sphere (for example, the surface). In this case, it might be necessary to scale and/or bias the vertical height. The height will become height*scale + bias. Keep in mind that if the positive attribute of the vertical dimension is down, then the height is negated. By default the scale is 1 and the bias is 0 (that is, no change). The scaling will be adjusted if it results in invalid (negative) vertical values.

- `obj.SetVerticalBias (double )` - The scale and bias of the vertical component of spherical coordinates. It is common to write the vertical component with respect to something other than the center of the sphere (for example, the surface). In this case, it might be necessary to scale and/or bias the vertical height. The height will become height*scale + bias. Keep in mind that if the positive attribute of the vertical dimension is down, then the height is negated. By default the scale is 1 and the bias is 0 (that is, no change). The scaling will be adjusted if it results in invalid (negative) vertical values.

37.62 vtkNetCDFPOPReader

37.62.1 Usage

vtkNetCDFPOPReader is a source object that reads NetCDF files. It should be able to read most any NetCDF file that wants to output rectilinear grid

To create an instance of class vtkNetCDFPOPReader, simply invoke its constructor as follows

```
obj = vtkNetCDFPOPReader
```

37.62.2 Methods

The class vtkNetCDFPOPReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkNetCDFPOPReader class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkNetCDFPOPReader = obj.NewInstance ()`

- `vtkNetCDFPOPReader = obj.SafeDownCast (vtkObject o)`

- `obj.SetFilename (string )`

- `string = obj.GetFilename ()`

- obj.SetWholeExtent (int , int , int , int , int , int)
- obj.SetWholeExtent (int a[6])
- int = obj. GetWholeExtent ()
- obj.SetSubExtent (int , int , int , int , int , int)
- obj.SetSubExtent (int a[6])
- int = obj. GetSubExtent ()
- obj.SetOrigin (double , double , double)
- obj.SetOrigin (double a[3])
- double = obj. GetOrigin ()
- obj.SetSpacing (double , double , double)
- obj.SetSpacing (double a[3])
- double = obj. GetSpacing ()
- obj.SetStride (int , int , int)
- obj.SetStride (int a[3])
- int = obj. GetStride ()
- obj.SetBlockReadSize (int)
- int = obj.GetBlockReadSize ()
- int = obj.GetNumberOfVariableArrays () - Variable array selection.
- string = obj.GetVariableArrayName (int idx) - Variable array selection.
- int = obj.GetVariableArrayStatus (string name) - Variable array selection.
- obj.SetVariableArrayStatus (string name, int status) - Variable array selection.

37.63 vtkNetCDFReader

37.63.1 Usage

A superclass for reading netCDF files. Subclass add conventions to the reader. This class just outputs data into a multi block data set with a vtkImageData at each block. A block is created for each variable except that variables with matching dimensions will be placed in the same block.

To create an instance of class vtkNetCDFReader, simply invoke its constructor as follows

```
obj = vtkNetCDFReader
```

37.63.2 Methods

The class vtkNetCDFReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkNetCDFReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkNetCDFReader = obj.NewInstance ()`
- `vtkNetCDFReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string filename)`
- `string = obj.GetFileName ()`
- `int = obj.UpdateMetaData ()` - Update the meta data from the current file. Automatically called during the RequestInformation pipeline update stage.
- `int = obj.GetNumberOfVariableArrays ()` - Variable array selection.
- `string = obj.GetVariableArrayName (int idx)` - Variable array selection.
- `int = obj.GetVariableArrayStatus (string name)` - Variable array selection.
- `obj.SetVariableArrayStatus (string name, int status)` - Variable array selection.
- `vtkStringArray = obj.GetVariableDimensions ()` - Returns an array with string encodings for the dimensions used in each of the variables. The indices in the returned array correspond to those used in the GetVariableArrayName method. Two arrays with the same dimensions will have the same encoded string returned by this method.
- `obj.SetDimensions (string dimensions)` - Loads the grid with the given dimensions. The dimensions are encoded in a string that conforms to the same format as returned by GetVariableDimensions and GetAllDimensions. This method is really a convenience method for SetVariableArrayStatus. It turns on all variables that have the given dimensions and turns off all other variables.
- `vtkStringArray = obj.GetAllDimensions ()` - Returns an array with string encodings for the dimension combinations used in the variables. The result is the same as GetVariableDimensions except that each entry in the array is unique (a set of dimensions is only given once even if it occurs for multiple variables) and the order is meaningless.
- `int = obj.GetReplaceFillValueWithNan ()` - If on, any float or double variable read that has a _FillValue attribute will have that fill value replaced with a not-a-number (NaN) value. The advantage of setting these to NaN values is that, if implemented properly by the system and careful math operations are used, they can implicitly be ignored by calculations like finding the range of the values. That said, this option should be used with caution as VTK does not fully support NaN values and therefore odd calculations may occur. By default this is off.
- `obj.SetReplaceFillValueWithNan (int )` - If on, any float or double variable read that has a _FillValue attribute will have that fill value replaced with a not-a-number (NaN) value. The advantage of setting these to NaN values is that, if implemented properly by the system and careful math operations are used, they can implicitly be ignored by calculations like finding the range of the values. That said, this option should be used with caution as VTK does not fully support NaN values and therefore odd calculations may occur. By default this is off.

- `obj.ReplaceFillValueWithNanOn ()` - If on, any float or double variable read that has a _FillValue attribute will have that fill value replaced with a not-a-number (NaN) value. The advantage of setting these to NaN values is that, if implemented properly by the system and careful math operations are used, they can implicitly be ignored by calculations like finding the range of the values. That said, this option should be used with caution as VTK does not fully support NaN values and therefore odd calculations may occur. By default this is off.

- `obj.ReplaceFillValueWithNanOff ()` - If on, any float or double variable read that has a _FillValue attribute will have that fill value replaced with a not-a-number (NaN) value. The advantage of setting these to NaN values is that, if implemented properly by the system and careful math operations are used, they can implicitly be ignored by calculations like finding the range of the values. That said, this option should be used with caution as VTK does not fully support NaN values and therefore odd calculations may occur. By default this is off.

37.64 vtkOBJReader

37.64.1 Usage

vtkOBJReader is a source object that reads Wavefront .obj files. The output of this source object is polygonal data.

To create an instance of class vtkOBJReader, simply invoke its constructor as follows

```
obj = vtkOBJReader
```

37.64.2 Methods

The class vtkOBJReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOBJReader class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkOBJReader = obj.NewInstance ()`

- `vtkOBJReader = obj.SafeDownCast (vtkObject o)`

- `obj.SetFileName (string )` - Specify file name of Wavefront .obj file.

- `string = obj.GetFileName ()` - Specify file name of Wavefront .obj file.

37.65 vtkOpenFOAMReader

37.65.1 Usage

vtkOpenFOAMReader creates a multiblock dataset. It reads mesh information and time dependent data. The polyMesh folders contain mesh information. The time folders contain transient data for the cells. Each folder can contain any number of data files.

To create an instance of class vtkOpenFOAMReader, simply invoke its constructor as follows

```
obj = vtkOpenFOAMReader
```

37.65.2 Methods

The class vtkOpenFOAMReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenFOAMReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenFOAMReader = obj.NewInstance ()`
- `vtkOpenFOAMReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string )` - Determine if the file can be readed with this reader.
- `obj.SetFileName (string )` - Set/Get the filename.
- `string = obj.GetFileName ()` - Set/Get the filename.
- `int = obj.GetNumberOfCellArrays (void )` - Get/Set whether the cell array with the given name is to be read.
- `int = obj.GetCellArrayStatus (string name)` - Get/Set whether the cell array with the given name is to be read.
- `obj.SetCellArrayStatus (string name, int status)` - Get the name of the cell array with the given index in the input.
- `string = obj.GetCellArrayName (int index)` - Turn on/off all cell arrays.
- `obj.DisableAllCellArrays ()` - Turn on/off all cell arrays.
- `obj.EnableAllCellArrays ()` - Get the number of point arrays available in the input.
- `int = obj.GetNumberOfPointArrays (void )` - Get/Set whether the point array with the given name is to be read.
- `int = obj.GetPointArrayStatus (string name)` - Get/Set whether the point array with the given name is to be read.
- `obj.SetPointArrayStatus (string name, int status)` - Get the name of the point array with the given index in the input.
- `string = obj.GetPointArrayName (int index)` - Turn on/off all point arrays.
- `obj.DisableAllPointArrays ()` - Turn on/off all point arrays.
- `obj.EnableAllPointArrays ()` - Get the number of Lagrangian arrays available in the input.
- `int = obj.GetNumberOfLagrangianArrays (void )` - Get/Set whether the Lagrangian array with the given name is to be read.
- `int = obj.GetLagrangianArrayStatus (string name)` - Get/Set whether the Lagrangian array with the given name is to be read.
- `obj.SetLagrangianArrayStatus (string name, int status)` - Get the name of the Lagrangian array with the given index in the input.
- `string = obj.GetLagrangianArrayName (int index)` - Turn on/off all Lagrangian arrays.
- `obj.DisableAllLagrangianArrays ()` - Turn on/off all Lagrangian arrays.

- `obj.EnableAllLagrangianArrays ()` - Get the number of Patches (inlcuding Internal Mesh) available in the input.
- `int = obj.GetNumberOfPatchArrays (void )` - Get/Set whether the Patch with the given name is to be read.
- `int = obj.GetPatchArrayStatus (string name)` - Get/Set whether the Patch with the given name is to be read.
- `obj.SetPatchArrayStatus (string name, int status)` - Get the name of the Patch with the given index in the input.
- `string = obj.GetPatchArrayName (int index)` - Turn on/off all Patches including the Internal Mesh.
- `obj.DisableAllPatchArrays ()` - Turn on/off all Patches including the Internal Mesh.
- `obj.EnableAllPatchArrays ()` - Set/Get whether to create cell-to-point translated data for cell-type data
- `obj.SetCreateCellToPoint (int )` - Set/Get whether to create cell-to-point translated data for cell-type data
- `int = obj.GetCreateCellToPoint ()` - Set/Get whether to create cell-to-point translated data for cell-type data
- `obj.CreateCellToPointOn ()` - Set/Get whether to create cell-to-point translated data for cell-type data
- `obj.CreateCellToPointOff ()` - Set/Get whether to create cell-to-point translated data for cell-type data
- `obj.SetCacheMesh (int )` - Set/Get whether mesh is to be cached.
- `int = obj.GetCacheMesh ()` - Set/Get whether mesh is to be cached.
- `obj.CacheMeshOn ()` - Set/Get whether mesh is to be cached.
- `obj.CacheMeshOff ()` - Set/Get whether mesh is to be cached.
- `obj.SetDecomposePolyhedra (int )` - Set/Get whether polyhedra are to be decomposed.
- `int = obj.GetDecomposePolyhedra ()` - Set/Get whether polyhedra are to be decomposed.
- `obj.DecomposePolyhedraOn ()` - Set/Get whether polyhedra are to be decomposed.
- `obj.DecomposePolyhedraOff ()` - Set/Get whether polyhedra are to be decomposed.
- `obj.SetPositionsIsIn13Format (int )` - Set/Get whether the lagrangian/positions is in OF 1.3 format
- `int = obj.GetPositionsIsIn13Format ()` - Set/Get whether the lagrangian/positions is in OF 1.3 format
- `obj.PositionsIsIn13FormatOn ()` - Set/Get whether the lagrangian/positions is in OF 1.3 format
- `obj.PositionsIsIn13FormatOff ()` - Set/Get whether the lagrangian/positions is in OF 1.3 format
- `obj.SetListTimeStepsByControlDict (int )` - Determine if time directories are to be listed according to controlDict
- `int = obj.GetListTimeStepsByControlDict ()` - Determine if time directories are to be listed according to controlDict

- `obj.ListTimeStepsByControlDictOn ()` - Determine if time directories are to be listed according to controlDict
- `obj.ListTimeStepsByControlDictOff ()` - Determine if time directories are to be listed according to controlDict
- `obj.SetAddDimensionsToArrayNames (int )` - Add dimensions to array names
- `int = obj.GetAddDimensionsToArrayNames ()` - Add dimensions to array names
- `obj.AddDimensionsToArrayNamesOn ()` - Add dimensions to array names
- `obj.AddDimensionsToArrayNamesOff ()` - Add dimensions to array names
- `obj.SetReadZones (int )` - Set/Get whether zones will be read.
- `int = obj.GetReadZones ()` - Set/Get whether zones will be read.
- `obj.ReadZonesOn ()` - Set/Get whether zones will be read.
- `obj.ReadZonesOff ()` - Set/Get whether zones will be read.
- `obj.SetRefresh ()`
- `obj.SetParent (vtkOpenFOAMReader parent)`
- `bool = obj.SetTimeValue (double )`
- `vtkDoubleArray = obj.GetTimeValues ()`
- `int = obj.MakeMetaDataAtTimeStep (bool )`

37.66 vtkOutputStream

37.66.1 Usage

vtkOutputStream provides a VTK-style interface wrapping around a standard output stream. The access methods are virtual so that subclasses can transparently provide encoding of the output. Data lengths for Write calls refer to the length of the data in memory. The actual length in the stream may differ for subclasses that implement an encoding scheme.

To create an instance of class vtkOutputStream, simply invoke its constructor as follows

```
obj = vtkOutputStream
```

37.66.2 Methods

The class vtkOutputStream has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOutputStream class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOutputStream = obj.NewInstance ()`
- `vtkOutputStream = obj.SafeDownCast (vtkObject o)`
- `int = obj.StartWriting ()` - Called after the stream position has been set by the caller, but before any Write calls. The stream position should not be adjusted by the caller until after an EndWriting call.

- `int = obj.Write (string data, long length)` - Write output data of the given length.
- `int = obj.Write (string data, long length)` - Write output data of the given length.
- `int = obj.EndWriting ()` - Called after all desired calls to Write have been made. After this call, the caller is free to change the position of the stream. Additional writes should not be done until after another call to StartWriting.

37.67 vtkParticleReader

37.67.1 Usage

vtkParticleReader reads either a binary or a text file of particles. Each particle can have associated with it an optional scalar value. So the format is: x, y, z, scalar (all floats or doubles). The text file can consist of a comma delimited set of values. In most cases vtkParticleReader can automatically determine whether the file is text or binary. The data can be either float or double. Progress updates are provided. With respect to binary files, random access into the file to read pieces is supported.

To create an instance of class vtkParticleReader, simply invoke its constructor as follows

```
obj = vtkParticleReader
```

37.67.2 Methods

The class vtkParticleReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParticleReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParticleReader = obj.NewInstance ()`
- `vtkParticleReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name.
- `string = obj.GetFileName ()` - Specify file name.
- `obj.SetDataByteOrderToBigEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.
- `obj.SetDataByteOrderToLittleEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.

- `int = obj.GetDataByteOrder ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.

- `obj.SetDataByteOrder (int )` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.

- `string = obj.GetDataByteOrderAsString ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian. Not used when reading text files.

- `obj.SetSwapBytes (int )` - Set/Get the byte swapping to explicitly swap the bytes of a file. Not used when reading text files.

- `int = obj.GetSwapBytes ()` - Set/Get the byte swapping to explicitly swap the bytes of a file. Not used when reading text files.

- `obj.SwapBytesOn ()` - Set/Get the byte swapping to explicitly swap the bytes of a file. Not used when reading text files.

- `obj.SwapBytesOff ()` - Set/Get the byte swapping to explicitly swap the bytes of a file. Not used when reading text files.

- `obj.SetHasScalar (int )` - Default: 1. If 1 then each particle has a value associated with it.

- `int = obj.GetHasScalar ()` - Default: 1. If 1 then each particle has a value associated with it.

- `obj.HasScalarOn ()` - Default: 1. If 1 then each particle has a value associated with it.

- `obj.HasScalarOff ()` - Default: 1. If 1 then each particle has a value associated with it.

- `obj.SetFileType (int )` - Get/Set the file type. The options are: - FILE_TYPE_IS_UNKNOWN (default) the class will attempt to determine the file type. If this fails then you should set the file type yourself. - FILE_TYPE_IS_TEXT the file type is text. - FILE_TYPE_IS_BINARY the file type is binary.

- `int = obj.GetFileTypeMinValue ()` - Get/Set the file type. The options are: - FILE_TYPE_IS_UNKNOWN (default) the class will attempt to determine the file type. If this fails then you should set the file type yourself. - FILE_TYPE_IS_TEXT the file type is text. - FILE_TYPE_IS_BINARY the file type is binary.

- `int = obj.GetFileTypeMaxValue ()` - Get/Set the file type. The options are: - FILE_TYPE_IS_UNKNOWN (default) the class will attempt to determine the file type. If this fails then you should set the file type yourself. - FILE_TYPE_IS_TEXT the file type is text. - FILE_TYPE_IS_BINARY the file type is binary.
- `int = obj.GetFileType ()` - Get/Set the file type. The options are: - FILE_TYPE_IS_UNKNOWN (default) the class will attempt to determine the file type. If this fails then you should set the file type yourself. - FILE_TYPE_IS_TEXT the file type is text. - FILE_TYPE_IS_BINARY the file type is binary.
- `obj.SetFileTypeToUnknown ()` - Get/Set the file type. The options are: - FILE_TYPE_IS_UNKNOWN (default) the class will attempt to determine the file type. If this fails then you should set the file type yourself. - FILE_TYPE_IS_TEXT the file type is text. - FILE_TYPE_IS_BINARY the file type is binary.
- `obj.SetFileTypeToText ()` - Get/Set the file type. The options are: - FILE_TYPE_IS_UNKNOWN (default) the class will attempt to determine the file type. If this fails then you should set the file type yourself. - FILE_TYPE_IS_TEXT the file type is text. - FILE_TYPE_IS_BINARY the file type is binary.
- `obj.SetFileTypeToBinary ()` - Get/Set the data type. The options are: - VTK_FLOAT (default) single precision floating point. - VTK_DOUBLE double precision floating point.
- `obj.SetDataType (int )` - Get/Set the data type. The options are: - VTK_FLOAT (default) single precision floating point. - VTK_DOUBLE double precision floating point.
- `int = obj.GetDataTypeMinValue ()` - Get/Set the data type. The options are: - VTK_FLOAT (default) single precision floating point. - VTK_DOUBLE double precision floating point.
- `int = obj.GetDataTypeMaxValue ()` - Get/Set the data type. The options are: - VTK_FLOAT (default) single precision floating point. - VTK_DOUBLE double precision floating point.
- `int = obj.GetDataType ()` - Get/Set the data type. The options are: - VTK_FLOAT (default) single precision floating point. - VTK_DOUBLE double precision floating point.
- `obj.SetDataTypeToFloat ()` - Get/Set the data type. The options are: - VTK_FLOAT (default) single precision floating point. - VTK_DOUBLE double precision floating point.
- `obj.SetDataTypeToDouble ()`

37.68 vtkPDBReader

37.68.1 Usage

vtkPDBReader is a source object that reads Molecule files The FileName must be specified
.SECTION Thanks Dr. Jean M. Favre who developed and contributed this class
To create an instance of class vtkPDBReader, simply invoke its constructor as follows

```
obj = vtkPDBReader
```

37.68.2 Methods

The class vtkPDBReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPDBReader class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkPDBReader = obj.NewInstance ()`
- `vtkPDBReader = obj.SafeDownCast (vtkObject o)`

37.69 vtkPLOT3DReader

37.69.1 Usage

vtkPLOT3DReader is a reader object that reads PLOT3D formatted files and generates structured grid(s) on output. PLOT3D is a computer graphics program designed to visualize the grids and solutions of computational fluid dynamics. Please see the "PLOT3D User's Manual" available from NASA Ames Research Center, Moffett Field CA.

PLOT3D files consist of a grid file (also known as XYZ file), an optional solution file (also known as a Q file), and an optional function file that contains user created data (currently unsupported). The Q file contains solution information as follows: the four parameters free stream mach number (Fsmach), angle of attack (Alpha), Reynolds number (Re), and total integration time (Time). This information is stored in an array called Properties in the FieldData of each output (tuple 0: fsmach, tuple 1: alpha, tuple 2: re, tuple 3: time). In addition, the solution file contains the flow density (scalar), flow momentum (vector), and flow energy (scalar).

The reader can generate additional scalars and vectors (or "functions") from this information. To use vtkPLOT3DReader, you must specify the particular function number for the scalar and vector you want to visualize. This implementation of the reader provides the following functions. The scalar functions are: -1 - don't read or compute any scalars 100 - density 110 - pressure 120 - temperature 130 - enthalpy 140 - internal energy 144 - kinetic energy 153 - velocity magnitude 163 - stagnation energy 170 - entropy 184 - swirl.

The vector functions are: -1 - don't read or compute any vectors 200 - velocity 201 - vorticity 202 - momentum 210 - pressure gradient.

(Other functions are described in the PLOT3D spec, but only those listed are implemented here.) Note that by default, this reader creates the density scalar (100) and momentum vector (202) as output. (These are just read in from the solution file.) Please note that the validity of computation is a function of this class's gas constants (R, Gamma) and the equations used. They may not be suitable for your computational domain.

Additionally, you can read other data and associate it as a vtkDataArray into the output's point attribute data. Use the method AddFunction() to list all the functions that you'd like to read. AddFunction() accepts an integer parameter that defines the function number.

To create an instance of class vtkPLOT3DReader, simply invoke its constructor as follows

```
obj = vtkPLOT3DReader
```

37.69.2 Methods

The class vtkPLOT3DReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPLOT3DReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPLOT3DReader = obj.NewInstance ()`
- `vtkPLOT3DReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string name)` - Set/Get the PLOT3D geometry filename.

- `string = obj.GetFileName ()` - Set/Get the PLOT3D geometry filename.
- `obj.SetXYZFileName (string )` - Set/Get the PLOT3D geometry filename.
- `string = obj.GetXYZFileName ()` - Set/Get the PLOT3D geometry filename.
- `obj.SetQFileName (string )` - Set/Get the PLOT3D solution filename.
- `string = obj.GetQFileName ()` - Set/Get the PLOT3D solution filename.
- `obj.SetFunctionFileName (string )` - Set/Get the PLOT3D Function Filename (optional)
- `string = obj.GetFunctionFileName ()` - Set/Get the PLOT3D Function Filename (optional)
- `int = obj.GetNumberOfOutputs ()` - This returns the number of outputs this reader will produce. This number is equal to the number of grids in the current file. This method has to be called before getting any output if the number of outputs will be greater than 1 (the first output is always the same). Note that every time this method is invoked, the header file is opened and part of the header is read.
- `int = obj.GetNumberOfGrids ()` - Replace an output.
- `obj.SetOutput (int idx, vtkStructuredGrid output)` - Replace an output.
- `obj.SetBinaryFile (int )` - Is the file to be read written in binary format (as opposed to ascii).
- `int = obj.GetBinaryFile ()` - Is the file to be read written in binary format (as opposed to ascii).
- `obj.BinaryFileOn ()` - Is the file to be read written in binary format (as opposed to ascii).
- `obj.BinaryFileOff ()` - Is the file to be read written in binary format (as opposed to ascii).
- `obj.SetMultiGrid (int )` - Does the file to be read contain information about number of grids. In some PLOT3D files, the first value contains the number of grids (even if there is only 1). If reading such a file, set this to true.
- `int = obj.GetMultiGrid ()` - Does the file to be read contain information about number of grids. In some PLOT3D files, the first value contains the number of grids (even if there is only 1). If reading such a file, set this to true.
- `obj.MultiGridOn ()` - Does the file to be read contain information about number of grids. In some PLOT3D files, the first value contains the number of grids (even if there is only 1). If reading such a file, set this to true.
- `obj.MultiGridOff ()` - Does the file to be read contain information about number of grids. In some PLOT3D files, the first value contains the number of grids (even if there is only 1). If reading such a file, set this to true.
- `obj.SetHasByteCount (int )` - Were the arrays written with leading and trailing byte counts ? Usually, files written by a fortran program will contain these byte counts whereas the ones written by C/C++ won't.
- `int = obj.GetHasByteCount ()` - Were the arrays written with leading and trailing byte counts ? Usually, files written by a fortran program will contain these byte counts whereas the ones written by C/C++ won't.
- `obj.HasByteCountOn ()` - Were the arrays written with leading and trailing byte counts ? Usually, files written by a fortran program will contain these byte counts whereas the ones written by C/C++ won't.
- `obj.HasByteCountOff ()` - Were the arrays written with leading and trailing byte counts ? Usually, files written by a fortran program will contain these byte counts whereas the ones written by C/C++ won't.

- `obj.SetIBlanking (int )` - Is there iblanking (point visibility) information in the file. If there is iblanking arrays, these will be read and assigned to the PointVisibility array of the output.
- `int = obj.GetIBlanking ()` - Is there iblanking (point visibility) information in the file. If there is iblanking arrays, these will be read and assigned to the PointVisibility array of the output.
- `obj.IBlankingOn ()` - Is there iblanking (point visibility) information in the file. If there is iblanking arrays, these will be read and assigned to the PointVisibility array of the output.
- `obj.IBlankingOff ()` - Is there iblanking (point visibility) information in the file. If there is iblanking arrays, these will be read and assigned to the PointVisibility array of the output.
- `obj.SetTwoDimensionalGeometry (int )` - If only two-dimensional data was written to the file, turn this on.
- `int = obj.GetTwoDimensionalGeometry ()` - If only two-dimensional data was written to the file, turn this on.
- `obj.TwoDimensionalGeometryOn ()` - If only two-dimensional data was written to the file, turn this on.
- `obj.TwoDimensionalGeometryOff ()` - If only two-dimensional data was written to the file, turn this on.
- `obj.SetForceRead (int )` - Try to read a binary file even if the file length seems to be inconsistent with the header information. Use this with caution, if the file length is not the same as calculated from the header. either the file is corrupt or the settings are wrong.
- `int = obj.GetForceRead ()` - Try to read a binary file even if the file length seems to be inconsistent with the header information. Use this with caution, if the file length is not the same as calculated from the header. either the file is corrupt or the settings are wrong.
- `obj.ForceReadOn ()` - Try to read a binary file even if the file length seems to be inconsistent with the header information. Use this with caution, if the file length is not the same as calculated from the header. either the file is corrupt or the settings are wrong.
- `obj.ForceReadOff ()` - Try to read a binary file even if the file length seems to be inconsistent with the header information. Use this with caution, if the file length is not the same as calculated from the header. either the file is corrupt or the settings are wrong.
- `obj.SetDoNotReduceNumberOfOutputs (int )` - If this is on, the reader will never reduce the number of outputs after reading a file with n grids and producing n outputs. If the file read afterwards contains fewer grids, the extra outputs will be empty. This option can be used by application which rely on the initial number of outputs not shrinking.
- `int = obj.GetDoNotReduceNumberOfOutputs ()` - If this is on, the reader will never reduce the number of outputs after reading a file with n grids and producing n outputs. If the file read afterwards contains fewer grids, the extra outputs will be empty. This option can be used by application which rely on the initial number of outputs not shrinking.
- `obj.DoNotReduceNumberOfOutputsOn ()` - If this is on, the reader will never reduce the number of outputs after reading a file with n grids and producing n outputs. If the file read afterwards contains fewer grids, the extra outputs will be empty. This option can be used by application which rely on the initial number of outputs not shrinking.
- `obj.DoNotReduceNumberOfOutputsOff ()` - If this is on, the reader will never reduce the number of outputs after reading a file with n grids and producing n outputs. If the file read afterwards contains fewer grids, the extra outputs will be empty. This option can be used by application which rely on the initial number of outputs not shrinking.

- `obj.SetByteOrderToBigEndian ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `obj.SetByteOrderToLittleEndian ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `obj.SetByteOrder (int )` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `int = obj.GetByteOrder ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `string = obj.GetByteOrderAsString ()` - Set the byte order of the file (remember, more Unix workstations write big endian whereas PCs write little endian). Default is big endian (since most older PLOT3D files were written by workstations).
- `obj.SetR (double )` - Set/Get the gas constant. Default is 1.0.
- `double = obj.GetR ()` - Set/Get the gas constant. Default is 1.0.
- `obj.SetGamma (double )` - Set/Get the ratio of specific heats. Default is 1.4.
- `double = obj.GetGamma ()` - Set/Get the ratio of specific heats. Default is 1.4.
- `obj.SetUvinf (double )` - Set/Get the x-component of the free-stream velocity. Default is 1.0.
- `double = obj.GetUvinf ()` - Set/Get the x-component of the free-stream velocity. Default is 1.0.
- `obj.SetVvinf (double )` - Set/Get the y-component of the free-stream velocity. Default is 1.0.
- `double = obj.GetVvinf ()` - Set/Get the y-component of the free-stream velocity. Default is 1.0.
- `obj.SetWvinf (double )` - Set/Get the z-component of the free-stream velocity. Default is 1.0.
- `double = obj.GetWvinf ()` - Set/Get the z-component of the free-stream velocity. Default is 1.0.
- `obj.SetScalarFunctionNumber (int num)` - Specify the scalar function to extract. If ==(-1), then no scalar function is extracted.
- `int = obj.GetScalarFunctionNumber ()` - Specify the scalar function to extract. If ==(-1), then no scalar function is extracted.
- `obj.SetVectorFunctionNumber (int num)` - Specify the vector function to extract. If ==(-1), then no vector function is extracted.
- `int = obj.GetVectorFunctionNumber ()` - Specify the vector function to extract. If ==(-1), then no vector function is extracted.
- `obj.AddFunction (int functionNumber)` - Specify additional functions to read. These are placed into the point data as data arrays. Later on they can be used by labeling them as scalars, etc.
- `obj.RemoveFunction (int )` - Specify additional functions to read. These are placed into the point data as data arrays. Later on they can be used by labeling them as scalars, etc.
- `obj.RemoveAllFunctions ()` - Specify additional functions to read. These are placed into the point data as data arrays. Later on they can be used by labeling them as scalars, etc.
- `int = obj.CanReadBinaryFile (string fname)` - Return 1 if the reader can read the given file name. Only meaningful for binary files.

37.70 vtkPLYReader

37.70.1 Usage

vtkPLYReader is a source object that reads polygonal data in Stanford University PLY file format (see http://graphics.stanford.edu/data/3Dscanrep). It requires that the elements "vertex" and "face" are defined. The "vertex" element must have the properties "x", "y", and "z". The "face" element must have the property "vertex_indices" defined. Optionally, if the "face" element has the properties "intensity" and/or the triplet "red", "green", and "blue"; these are read and added as scalars to the output data.

To create an instance of class vtkPLYReader, simply invoke its constructor as follows

```
obj = vtkPLYReader
```

37.70.2 Methods

The class vtkPLYReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPLYReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPLYReader = obj.NewInstance ()`
- `vtkPLYReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name of stereo lithography file.
- `string = obj.GetFileName ()` - Specify file name of stereo lithography file.

37.71 vtkPLYWriter

37.71.1 Usage

vtkPLYWriter writes polygonal data in Stanford University PLY format (see http://graphics.stanford.edu/data/3Dscanrep/). The data can be written in either binary (little or big endian) or ASCII representation. As for PointData and CellData, vtkPLYWriter cannot handle normals or vectors. It only handles RGB PointData and CellData. You need to set the name of the array (using SetName for the array and SetArrayName for the writer). If the array is not a vtkUnsignedCharArray with 3 components, you need to specify a vtkLookupTable to map the scalars to RGB.

To create an instance of class vtkPLYWriter, simply invoke its constructor as follows

```
obj = vtkPLYWriter
```

37.71.2 Methods

The class vtkPLYWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPLYWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPLYWriter = obj.NewInstance ()`

- `vtkPLYWriter = obj.SafeDownCast (vtkObject o)`

- `obj.SetDataByteOrder (int )` - If the file type is binary, then the user can specify which byte order to use (little versus big endian).

- `int = obj.GetDataByteOrderMinValue ()` - If the file type is binary, then the user can specify which byte order to use (little versus big endian).

- `int = obj.GetDataByteOrderMaxValue ()` - If the file type is binary, then the user can specify which byte order to use (little versus big endian).

- `int = obj.GetDataByteOrder ()` - If the file type is binary, then the user can specify which byte order to use (little versus big endian).

- `obj.SetDataByteOrderToBigEndian ()` - If the file type is binary, then the user can specify which byte order to use (little versus big endian).

- `obj.SetDataByteOrderToLittleEndian ()` - These methods enable the user to control how to add color into the PLY output file. The default behavior is as follows. The user provides the name of an array and a component number. If the type of the array is three components, unsigned char, then the data is written as three separate "red", "green" and "blue" properties. If the type is not unsigned char, and a lookup table is provided, then the array/component are mapped through the table to generate three separate "red", "green" and "blue" properties in the PLY file. The user can also set the ColorMode to specify a uniform color for the whole part (on a vertex colors, face colors, or both. (Note: vertex colors or cell colors may be written, depending on where the named array is found. If points and cells have the arrays with the same name, then both colors will be written.)

- `obj.SetColorMode (int )` - These methods enable the user to control how to add color into the PLY output file. The default behavior is as follows. The user provides the name of an array and a component number. If the type of the array is three components, unsigned char, then the data is written as three separate "red", "green" and "blue" properties. If the type is not unsigned char, and a lookup table is provided, then the array/component are mapped through the table to generate three separate "red", "green" and "blue" properties in the PLY file. The user can also set the ColorMode to specify a uniform color for the whole part (on a vertex colors, face colors, or both. (Note: vertex colors or cell colors may be written, depending on where the named array is found. If points and cells have the arrays with the same name, then both colors will be written.)

- `int = obj.GetColorMode ()` - These methods enable the user to control how to add color into the PLY output file. The default behavior is as follows. The user provides the name of an array and a component number. If the type of the array is three components, unsigned char, then the data is written as three separate "red", "green" and "blue" properties. If the type is not unsigned char, and a lookup table is provided, then the array/component are mapped through the table to generate three separate "red", "green" and "blue" properties in the PLY file. The user can also set the ColorMode to specify a uniform color for the whole part (on a vertex colors, face colors, or both. (Note: vertex colors or cell colors may be written, depending on where the named array is found. If points and cells have the arrays with the same name, then both colors will be written.)

- `obj.SetColorModeToDefault ()` - These methods enable the user to control how to add color into the PLY output file. The default behavior is as follows. The user provides the name of an array and a component number. If the type of the array is three components, unsigned char, then the data is written as three separate "red", "green" and "blue" properties. If the type is not unsigned char, and a lookup table is provided, then the array/component are mapped through the table to generate three separate "red", "green" and "blue" properties in the PLY file. The user can also set the ColorMode to specify a uniform color for the whole part (on a vertex colors, face colors, or both. (Note: vertex colors or cell colors may be written, depending on where the named array is found. If points and cells have the arrays with the same name, then both colors will be written.)

- `obj.SetColorModeToUniformCellColor ()` - These methods enable the user to control how to add color into the PLY output file. The default behavior is as follows. The user provides the name of an array and a component number. If the type of the array is three components, unsigned char, then the data is written as three separate "red", "green" and "blue" properties. If the type is not unsigned char, and a lookup table is provided, then the array/component are mapped through the table to generate three separate "red", "green" and "blue" properties in the PLY file. The user can also set the ColorMode to specify a uniform color for the whole part (on a vertex colors, face colors, or both. (Note: vertex colors or cell colors may be written, depending on where the named array is found. If points and cells have the arrays with the same name, then both colors will be written.)
- `obj.SetColorModeToUniformPointColor ()` - These methods enable the user to control how to add color into the PLY output file. The default behavior is as follows. The user provides the name of an array and a component number. If the type of the array is three components, unsigned char, then the data is written as three separate "red", "green" and "blue" properties. If the type is not unsigned char, and a lookup table is provided, then the array/component are mapped through the table to generate three separate "red", "green" and "blue" properties in the PLY file. The user can also set the ColorMode to specify a uniform color for the whole part (on a vertex colors, face colors, or both. (Note: vertex colors or cell colors may be written, depending on where the named array is found. If points and cells have the arrays with the same name, then both colors will be written.)
- `obj.SetColorModeToUniformColor ()` - These methods enable the user to control how to add color into the PLY output file. The default behavior is as follows. The user provides the name of an array and a component number. If the type of the array is three components, unsigned char, then the data is written as three separate "red", "green" and "blue" properties. If the type is not unsigned char, and a lookup table is provided, then the array/component are mapped through the table to generate three separate "red", "green" and "blue" properties in the PLY file. The user can also set the ColorMode to specify a uniform color for the whole part (on a vertex colors, face colors, or both. (Note: vertex colors or cell colors may be written, depending on where the named array is found. If points and cells have the arrays with the same name, then both colors will be written.)
- `obj.SetColorModeToOff ()` - Specify the array name to use to color the data.
- `obj.SetArrayName (string )` - Specify the array name to use to color the data.
- `string = obj.GetArrayName ()` - Specify the array name to use to color the data.
- `obj.SetComponent (int )` - Specify the array component to use to color the data.
- `int = obj.GetComponentMinValue ()` - Specify the array component to use to color the data.
- `int = obj.GetComponentMaxValue ()` - Specify the array component to use to color the data.
- `int = obj.GetComponent ()` - Specify the array component to use to color the data.
- `obj.SetLookupTable (vtkScalarsToColors )` - A lookup table can be specified in order to convert data arrays to RGBA colors.
- `vtkScalarsToColors = obj.GetLookupTable ()` - A lookup table can be specified in order to convert data arrays to RGBA colors.
- `obj.SetColor (char , char , char )` - Set the color to use when using a uniform color (either point or cells, or both). The color is specified as a triplet of three unsigned chars between (0,255). This only takes effect when the ColorMode is set to uniform point, uniform cell, or uniform color.
- `obj.SetColor (char a[3])` - Set the color to use when using a uniform color (either point or cells, or both). The color is specified as a triplet of three unsigned chars between (0,255). This only takes effect when the ColorMode is set to uniform point, uniform cell, or uniform color.
- `char = obj. GetColor ()` - Set the color to use when using a uniform color (either point or cells, or both). The color is specified as a triplet of three unsigned chars between (0,255). This only takes effect when the ColorMode is set to uniform point, uniform cell, or uniform color.

37.72 vtkPNGReader

37.72.1 Usage

vtkPNGReader is a source object that reads PNG files. It should be able to read most any PNG file
To create an instance of class vtkPNGReader, simply invoke its constructor as follows

```
obj = vtkPNGReader
```

37.72.2 Methods

The class vtkPNGReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPNGReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPNGReader = obj.NewInstance ()`
- `vtkPNGReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string fname)` - Is the given file a PNG file?
- `string = obj.GetFileExtensions ()` - Return a descriptive name for the file format that might be useful in a GUI.
- `string = obj.GetDescriptiveName ()`

37.73 vtkPNGWriter

37.73.1 Usage

vtkPNGWriter writes PNG files. It supports 1 to 4 component data of unsigned char or unsigned short
To create an instance of class vtkPNGWriter, simply invoke its constructor as follows

```
obj = vtkPNGWriter
```

37.73.2 Methods

The class vtkPNGWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPNGWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPNGWriter = obj.NewInstance ()`
- `vtkPNGWriter = obj.SafeDownCast (vtkObject o)`
- `obj.Write ()` - The main interface which triggers the writer to start.
- `obj.SetWriteToMemory (int )` - Write the image to memory (a vtkUnsignedCharArray)
- `int = obj.GetWriteToMemory ()` - Write the image to memory (a vtkUnsignedCharArray)

- `obj.WriteToMemoryOn ()` - Write the image to memory (a vtkUnsignedCharArray)
- `obj.WriteToMemoryOff ()` - Write the image to memory (a vtkUnsignedCharArray)
- `obj.SetResult (vtkUnsignedCharArray )` - When writing to memory this is the result, it will be NULL until the data is written the first time
- `vtkUnsignedCharArray = obj.GetResult ()` - When writing to memory this is the result, it will be NULL until the data is written the first time

37.74 vtkPNMReader

37.74.1 Usage

vtkPNMReader is a source object that reads pnm (portable anymap) files. This includes .pbm (bitmap), .pgm (grayscale), and .ppm (pixmap) files. (Currently this object only reads binary versions of these files.)

PNMReader creates structured point datasets. The dimension of the dataset depends upon the number of files read. Reading a single file results in a 2D image, while reading more than one file results in a 3D volume.

To read a volume, files must be of the form "FileName.¡number¿" (e.g., foo.ppm.0, foo.ppm.1, ...). You must also specify the DataExtent. The fifth and sixth values of the DataExtent specify the beginning and ending files to read.

To create an instance of class vtkPNMReader, simply invoke its constructor as follows

```
obj = vtkPNMReader
```

37.74.2 Methods

The class vtkPNMReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPNMReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPNMReader = obj.NewInstance ()`
- `vtkPNMReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string fname)`
- `string = obj.GetFileExtensions ()` - PNM
- `string = obj.GetDescriptiveName ()`

37.75 vtkPNMWriter

37.75.1 Usage

vtkPNMWriter writes PNM file. The data type of the file is unsigned char regardless of the input type.

To create an instance of class vtkPNMWriter, simply invoke its constructor as follows

```
obj = vtkPNMWriter
```

37.75.2 Methods

The class vtkPNMWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPNMWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPNMWriter = obj.NewInstance ()`
- `vtkPNMWriter = obj.SafeDownCast (vtkObject o)`

37.76 vtkPolyDataReader

37.76.1 Usage

vtkPolyDataReader is a source object that reads ASCII or binary polygonal data files in vtk format (see text for format details). The output of this reader is a single vtkPolyData data object. The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkPolyDataReader, simply invoke its constructor as follows

```
obj = vtkPolyDataReader
```

37.76.2 Methods

The class vtkPolyDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataReader = obj.NewInstance ()`
- `vtkPolyDataReader = obj.SafeDownCast (vtkObject o)`
- `vtkPolyData = obj.GetOutput ()` - Get the output of this reader.
- `vtkPolyData = obj.GetOutput (int idx)` - Get the output of this reader.
- `obj.SetOutput (vtkPolyData output)` - Get the output of this reader.

37.77 vtkPolyDataWriter

37.77.1 Usage

vtkPolyDataWriter is a source object that writes ASCII or binary polygonal data files in vtk format. See text for format details.

To create an instance of class vtkPolyDataWriter, simply invoke its constructor as follows

```
obj = vtkPolyDataWriter
```

37.77.2 Methods

The class vtkPolyDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataWriter = obj.NewInstance ()`
- `vtkPolyDataWriter = obj.SafeDownCast (vtkObject o)`
- `vtkPolyData = obj.GetInput ()` - Get the input to this writer.
- `vtkPolyData = obj.GetInput (int port)` - Get the input to this writer.

37.78 vtkPostScriptWriter

37.78.1 Usage

vtkPostScriptWriter writes an image as a PostScript file using some reasonable scalings and centered on the page which is assumed to be about 8.5 by 11 inches. This is based loosely off of the code from pnmtops.c. Right now there aren't any real options.

To create an instance of class vtkPostScriptWriter, simply invoke its constructor as follows

```
obj = vtkPostScriptWriter
```

37.78.2 Methods

The class vtkPostScriptWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPostScriptWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPostScriptWriter = obj.NewInstance ()`
- `vtkPostScriptWriter = obj.SafeDownCast (vtkObject o)`

37.79 vtkRectilinearGridReader

37.79.1 Usage

vtkRectilinearGridReader is a source object that reads ASCII or binary rectilinear grid data files in vtk format (see text for format details). The output of this reader is a single vtkRectilinearGrid data object. The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkRectilinearGridReader, simply invoke its constructor as follows

```
obj = vtkRectilinearGridReader
```

37.79.2 Methods

The class vtkRectilinearGridReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGridReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGridReader = obj.NewInstance ()`
- `vtkRectilinearGridReader = obj.SafeDownCast (vtkObject o)`
- `vtkRectilinearGrid = obj.GetOutput ()` - Get and set the output of this reader.
- `vtkRectilinearGrid = obj.GetOutput (int idx)` - Get and set the output of this reader.
- `obj.SetOutput (vtkRectilinearGrid output)` - Get and set the output of this reader.
- `int = obj.ReadMetaData (vtkInformation outInfo)` - Read the meta information from the file. This needs to be public to it can be accessed by vtkDataSetReader.

37.80 vtkRectilinearGridWriter

37.80.1 Usage

vtkRectilinearGridWriter is a source object that writes ASCII or binary rectilinear grid data files in vtk format. See text for format details.

To create an instance of class vtkRectilinearGridWriter, simply invoke its constructor as follows

```
obj = vtkRectilinearGridWriter
```

37.80.2 Methods

The class vtkRectilinearGridWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGridWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGridWriter = obj.NewInstance ()`
- `vtkRectilinearGridWriter = obj.SafeDownCast (vtkObject o)`
- `vtkRectilinearGrid = obj.GetInput ()` - Get the input to this writer.
- `vtkRectilinearGrid = obj.GetInput (int port)` - Get the input to this writer.

37.81 vtkRowQuery

37.81.1 Usage

The abstract superclass of query classes that return row-oriented (table) results. A subclass will provide database-specific query parameters and implement the vtkRowQuery API to return query results:

Execute() - Execute the query. No results need to be retrieved at this point, unless you are performing caching.

GetNumberOfFields() - After Execute() is performed, returns the number of fields in the query results.

GetFieldName() - The name of the field at an index.

GetFieldType() - The data type of the field at an index.

NextRow() - Advances the query results by one row, and returns whether there are more rows left in the query.

DataValue() - Extract a single data value from the current row.

.SECTION Thanks Thanks to Andrew Wilson from Sandia National Laboratories for his work on the database classes.

To create an instance of class vtkRowQuery, simply invoke its constructor as follows

```
obj = vtkRowQuery
```

37.81.2 Methods

The class vtkRowQuery has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRowQuery class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRowQuery = obj.NewInstance ()`
- `vtkRowQuery = obj.SafeDownCast (vtkObject o)`
- `bool = obj.Execute ()` - Execute the query. This must be performed before any field name or data access functions are used.
- `int = obj.GetNumberOfFields ()` - The number of fields in the query result.
- `string = obj.GetFieldName (int i)` - Return the name of the specified query field.
- `int = obj.GetFieldType (int i)` - Return the type of the field, using the constants defined in vtkType.h.
- `int = obj.GetFieldIndex (string name)` - Return the index of the specified query field. Uses GetNumberOfFields() and GetFieldName() to match field name.
- `bool = obj.NextRow ()` - Advance row, return false if past end.
- `bool = obj.IsActive ()` - Return true if the query is active (i.e. execution was successful and results are ready to be fetched). Returns false on error or inactive query.
- `bool = obj.HasError ()` - Returns true if an error is set, otherwise false.
- `string = obj.GetLastErrorText ()` - Get the last error text from the query
- `obj.SetCaseSensitiveFieldNames (bool )` - Many databases do not preserve case in field names. This can cause GetFieldIndex to fail if you search for a field named someFieldName when the database actually stores it as SOMEFIELDNAME. This ivar controls whether GetFieldIndex() expects field names to be case-sensitive. The default is OFF, i.e. case is not preserved.

- `bool = obj.GetCaseSensitiveFieldNames ()` - Many databases do not preserve case in field names. This can cause GetFieldIndex to fail if you search for a field named someFieldName when the database actually stores it as SOMEFIELDNAME. This ivar controls whether GetFieldIndex() expects field names to be case-sensitive. The default is OFF, i.e. case is not preserved.
- `obj.CaseSensitiveFieldNamesOn ()` - Many databases do not preserve case in field names. This can cause GetFieldIndex to fail if you search for a field named someFieldName when the database actually stores it as SOMEFIELDNAME. This ivar controls whether GetFieldIndex() expects field names to be case-sensitive. The default is OFF, i.e. case is not preserved.
- `obj.CaseSensitiveFieldNamesOff ()` - Many databases do not preserve case in field names. This can cause GetFieldIndex to fail if you search for a field named someFieldName when the database actually stores it as SOMEFIELDNAME. This ivar controls whether GetFieldIndex() expects field names to be case-sensitive. The default is OFF, i.e. case is not preserved.

37.82 vtkRowQueryToTable

37.82.1 Usage

vtkRowQueryToTable creates a vtkTable with the results of an arbitrary SQL query. To use this filter, you first need an instance of a vtkSQLDatabase subclass. You may use the database class to obtain a vtkRowQuery instance. Set that query on this filter to extract the query as a table.

.SECTION Thanks Thanks to Andrew Wilson from Sandia National Laboratories for his work on the database classes.

To create an instance of class vtkRowQueryToTable, simply invoke its constructor as follows

```
obj = vtkRowQueryToTable
```

37.82.2 Methods

The class vtkRowQueryToTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRowQueryToTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRowQueryToTable = obj.NewInstance ()`
- `vtkRowQueryToTable = obj.SafeDownCast (vtkObject o)`
- `obj.SetQuery (vtkRowQuery query)` - The query to execute.
- `vtkRowQuery = obj.GetQuery ()` - The query to execute.
- `long = obj.GetMTime ()` - Update the modified time based on the query.

37.83 vtkRTXMLPolyDataReader

37.83.1 Usage

vtkRTXMLPolyDataReader reads the VTK XML PolyData file format in real time.

To create an instance of class vtkRTXMLPolyDataReader, simply invoke its constructor as follows

```
obj = vtkRTXMLPolyDataReader
```

37.83.2 Methods

The class vtkRTXMLPolyDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRTXMLPolyDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRTXMLPolyDataReader = obj.NewInstance ()`
- `vtkRTXMLPolyDataReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetLocation (string dataLocation)`
- `string = obj.GetDataLocation ()`
- `obj.UpdateToNextFile ()` - Reader will read in the next available data file The filename is this-¿NextFileName maintained internally
- `int = obj.NewDataAvailable ()` - check if there is new data file available in the given DataLocation
- `obj.ResetReader ()` - ResetReader check the data directory specified in this-¿DataLocation, and reset the Internal data structure specifically: this-¿Internal-¿ProcessedFileList for monitoring the arriving new data files if SetDataLocation(char*) is set by the user, this ResetReader() should also be invoked.
- `string = obj.GetNextFileName ()` - Return the name of the next available data file assume NewDataAvailable() return VTK_OK

37.84 vtkSESAMEReader

37.84.1 Usage

vtkSESAMEReader is a source object that reads SESAME files. Currently supported tables include 301, 304, 502, 503, 504, 505, 602

SESAMEReader creates rectilinear grid datasets. The dimension of the dataset depends upon the number of densities and temperatures in the table. Values at certain temperatures and densities are stored as scalars.

To create an instance of class vtkSESAMEReader, simply invoke its constructor as follows

```
obj = vtkSESAMEReader
```

37.84.2 Methods

The class vtkSESAMEReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSESAMEReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSESAMEReader = obj.NewInstance ()`
- `vtkSESAMEReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string file)` - Set the filename to read

- `string = obj.GetFileName ()` - Get the filename to read
- `int = obj.IsValidFile ()` - Return whether this is a valid file
- `int = obj.GetNumberOfTableIds ()` - Get the number of tables in this file
- `vtkIntArray = obj.GetTableIdsAsArray ()` - Returns the table ids in a data array.
- `obj.SetTable (int tableId)` - Set the table to read in
- `int = obj.GetTable ()` - Get the table to read in
- `int = obj.GetNumberOfTableArrayNames ()` - Get the number of arrays for the table to read
- `int = obj.GetNumberOfTableArrays ()` - Get the names of arrays for the table to read
- `string = obj.GetTableArrayName (int index)` - Get the names of arrays for the table to read
- `obj.SetTableArrayStatus (string name, int flag)` - Set whether to read a table array
- `int = obj.GetTableArrayStatus (string name)` - Set whether to read a table array

37.85 vtkShaderCodeLibrary

37.85.1 Usage

This class provides the hardware shader code. .SECTION Thanks Shader support in VTK includes key contributions by Gary Templet at Sandia National Labs.

To create an instance of class vtkShaderCodeLibrary, simply invoke its constructor as follows

```
obj = vtkShaderCodeLibrary
```

37.85.2 Methods

The class vtkShaderCodeLibrary has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkShaderCodeLibrary class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkShaderCodeLibrary = obj.NewInstance ()`
- `vtkShaderCodeLibrary = obj.SafeDownCast (vtkObject o)`

37.86 vtkSimplePointsReader

37.86.1 Usage

vtkSimplePointsReader is a source object that reads a list of points from a file. Each point is specified by three floating-point values in ASCII format. There is one point per line of the file. A vertex cell is created for each point in the output. This reader is meant as an example of how to write a reader in VTK.

To create an instance of class vtkSimplePointsReader, simply invoke its constructor as follows

```
obj = vtkSimplePointsReader
```

37.86.2 Methods

The class vtkSimplePointsReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSimplePointsReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSimplePointsReader = obj.NewInstance ()`
- `vtkSimplePointsReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Set/Get the name of the file from which to read points.
- `string = obj.GetFileName ()` - Set/Get the name of the file from which to read points.

37.87 vtkSLACParticleReader

37.87.1 Usage

A reader for a data format used by Omega3p, Tau3p, and several other tools used at the Standford Linear Accelerator Center (SLAC). The underlying format uses netCDF to store arrays, but also imposes some conventions to store a list of particles in 3D space.

This reader supports pieces, but in actuality only loads anything in piece 0. All other pieces are empty.

To create an instance of class vtkSLACParticleReader, simply invoke its constructor as follows

```
obj = vtkSLACParticleReader
```

37.87.2 Methods

The class vtkSLACParticleReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSLACParticleReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSLACParticleReader = obj.NewInstance ()`
- `vtkSLACParticleReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetFileName ()`
- `obj.SetFileName (string )`

37.88 vtkSLACReader

37.88.1 Usage

A reader for a data format used by Omega3p, Tau3p, and several other tools used at the Standford Linear Accelerator Center (SLAC). The underlying format uses netCDF to store arrays, but also imposes several conventions to form an unstructured grid of elements.

To create an instance of class vtkSLACReader, simply invoke its constructor as follows

```
obj = vtkSLACReader
```

37.88.2 Methods

The class vtkSLACReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSLACReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSLACReader = obj.NewInstance ()`
- `vtkSLACReader = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetMeshFileName ()`
- `obj.SetMeshFileName (string )`
- `obj.AddModeFileName (string fname)` - There may be one mode file (usually for actual modes) or multiple mode files (which usually actually represent time series). These methods set and clear the list of mode files (which can be a single mode file).
- `obj.RemoveAllModeFileNames ()` - There may be one mode file (usually for actual modes) or multiple mode files (which usually actually represent time series). These methods set and clear the list of mode files (which can be a single mode file).
- `int = obj.GetNumberOfModeFileNames ()` - There may be one mode file (usually for actual modes) or multiple mode files (which usually actually represent time series). These methods set and clear the list of mode files (which can be a single mode file).
- `string = obj.GetModeFileName (int idx)` - There may be one mode file (usually for actual modes) or multiple mode files (which usually actually represent time series). These methods set and clear the list of mode files (which can be a single mode file).
- `int = obj.GetReadInternalVolume ()` - If on, reads the internal volume of the data set. Set to off by default.
- `obj.SetReadInternalVolume (int )` - If on, reads the internal volume of the data set. Set to off by default.
- `obj.ReadInternalVolumeOn ()` - If on, reads the internal volume of the data set. Set to off by default.
- `obj.ReadInternalVolumeOff ()` - If on, reads the internal volume of the data set. Set to off by default.
- `int = obj.GetReadExternalSurface ()` - If on, reads the external surfaces of the data set. Set to on by default.
- `obj.SetReadExternalSurface (int )` - If on, reads the external surfaces of the data set. Set to on by default.
- `obj.ReadExternalSurfaceOn ()` - If on, reads the external surfaces of the data set. Set to on by default.
- `obj.ReadExternalSurfaceOff ()` - If on, reads the external surfaces of the data set. Set to on by default.
- `int = obj.GetReadMidpoints ()` - If on, reads midpoint information for external surfaces and builds quadratic surface triangles. Set to on by default.

- `obj.SetReadMidpoints (int )` - If on, reads midpoint information for external surfaces and builds quadratic surface triangles. Set to on by default.
- `obj.ReadMidpointsOn ()` - If on, reads midpoint information for external surfaces and builds quadratic surface triangles. Set to on by default.
- `obj.ReadMidpointsOff ()` - If on, reads midpoint information for external surfaces and builds quadratic surface triangles. Set to on by default.
- `int = obj.GetNumberOfVariableArrays ()` - Variable array selection.
- `string = obj.GetVariableArrayName (int idx)` - Variable array selection.
- `int = obj.GetVariableArrayStatus (string name)` - Variable array selection.
- `obj.SetVariableArrayStatus (string name, int status)` - Variable array selection.

37.89 vtkSLCReader

37.89.1 Usage

vtkSLCReader reads an SLC file and creates a structured point dataset. The size of the volume and the data spacing is set from the SLC file header.

To create an instance of class vtkSLCReader, simply invoke its constructor as follows

```
obj = vtkSLCReader
```

37.89.2 Methods

The class vtkSLCReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSLCReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSLCReader = obj.NewInstance ()`
- `vtkSLCReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Set/Get the name of the file to read.
- `string = obj.GetFileName ()` - Set/Get the name of the file to read.
- `int = obj.GetError ()` - Was there an error on the last read performed?
- `int = obj.CanReadFile (string fname)` - Is the given file an SLC file?
- `string = obj.GetFileExtensions ()` - SLC
- `string = obj.GetDescriptiveName ()`

37.90 vtkSortFileNames

37.90.1 Usage

vtkSortFileNames will take a list of filenames (e.g. from a file load dialog) and sort them into one or more series. If the input list of filenames contains any directories, these can be removed before sorting using the SkipDirectories flag. This class should be used where information about the series groupings can be determined by the filenames, but it might not be successful in cases where the information about the series groupings is stored in the files themselves (e.g DICOM).

To create an instance of class vtkSortFileNames, simply invoke its constructor as follows

```
obj = vtkSortFileNames
```

37.90.2 Methods

The class vtkSortFileNames has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSortFileNames class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSortFileNames = obj.NewInstance ()`
- `vtkSortFileNames = obj.SafeDownCast (vtkObject o)`
- `obj.SetGrouping (int )` - Sort the file names into groups, according to similarity in filename name and path. Files in different directories, or with different extensions, or which do not fit into the same numbered series will be placed into different groups. This is off by default.
- `int = obj.GetGrouping ()` - Sort the file names into groups, according to similarity in filename name and path. Files in different directories, or with different extensions, or which do not fit into the same numbered series will be placed into different groups. This is off by default.
- `obj.GroupingOn ()` - Sort the file names into groups, according to similarity in filename name and path. Files in different directories, or with different extensions, or which do not fit into the same numbered series will be placed into different groups. This is off by default.
- `obj.GroupingOff ()` - Sort the file names into groups, according to similarity in filename name and path. Files in different directories, or with different extensions, or which do not fit into the same numbered series will be placed into different groups. This is off by default.
- `obj.SetNumericSort (int )` - Sort the files numerically, rather than lexicographically. For filenames that contain numbers, this means the order will be [”file8.dat”, ”file9.dat”, ”file10.dat”] instead of the usual alphabetic sorting order [”file10.dat” ”file8.dat”, ”file9.dat”]. NumericSort is off by default.
- `int = obj.GetNumericSort ()` - Sort the files numerically, rather than lexicographically. For filenames that contain numbers, this means the order will be [”file8.dat”, ”file9.dat”, ”file10.dat”] instead of the usual alphabetic sorting order [”file10.dat” ”file8.dat”, ”file9.dat”]. NumericSort is off by default.
- `obj.NumericSortOn ()` - Sort the files numerically, rather than lexicographically. For filenames that contain numbers, this means the order will be [”file8.dat”, ”file9.dat”, ”file10.dat”] instead of the usual alphabetic sorting order [”file10.dat” ”file8.dat”, ”file9.dat”]. NumericSort is off by default.
- `obj.NumericSortOff ()` - Sort the files numerically, rather than lexicographically. For filenames that contain numbers, this means the order will be [”file8.dat”, ”file9.dat”, ”file10.dat”] instead of the usual alphabetic sorting order [”file10.dat” ”file8.dat”, ”file9.dat”]. NumericSort is off by default.

- `obj.SetIgnoreCase (int )` - Ignore case when sorting. This flag is honored by both the sorting and the grouping. This is off by default.
- `int = obj.GetIgnoreCase ()` - Ignore case when sorting. This flag is honored by both the sorting and the grouping. This is off by default.
- `obj.IgnoreCaseOn ()` - Ignore case when sorting. This flag is honored by both the sorting and the grouping. This is off by default.
- `obj.IgnoreCaseOff ()` - Ignore case when sorting. This flag is honored by both the sorting and the grouping. This is off by default.
- `obj.SetSkipDirectories (int )` - Skip directories. If this flag is set, any input item that is a directory rather than a file will not be included in the output. This is off by default.
- `int = obj.GetSkipDirectories ()` - Skip directories. If this flag is set, any input item that is a directory rather than a file will not be included in the output. This is off by default.
- `obj.SkipDirectoriesOn ()` - Skip directories. If this flag is set, any input item that is a directory rather than a file will not be included in the output. This is off by default.
- `obj.SkipDirectoriesOff ()` - Skip directories. If this flag is set, any input item that is a directory rather than a file will not be included in the output. This is off by default.
- `obj.SetInputFileNames (vtkStringArray input)` - Set a list of file names to group and sort.
- `vtkStringArray = obj.GetInputFileNames ()` - Set a list of file names to group and sort.
- `vtkStringArray = obj.GetFileNames ()` - Get the full list of sorted filenames.
- `int = obj.GetNumberOfGroups ()` - Get the number of groups that the names were split into, if grouping is on. The filenames are automatically split into groups, where the filenames in each group will be identical except for their series numbers. If grouping is not on, this method will return zero.
- `vtkStringArray = obj.GetNthGroup (int i)` - Get the Nth group of file names. This method should only be used if grouping is on. If grouping is off, it will always return null.
- `obj.Update ()` - Update the output filenames from the input filenames. This method is called automatically by GetFileNames() and GetNumberOfGroups() if the input names have changed.

37.91 vtkSQLDatabase

37.91.1 Usage

Abstract base class for all SQL database connection classes. Manages a connection to the database, and is responsible for creating instances of the associated vtkSQLQuery objects associated with this class in order to perform execute queries on the database. To allow connections to a new type of database, create both a subclass of this class and vtkSQLQuery, and implement the required functions:

Open() - open the database connection, if possible. Close() - close the connection. GetQueryInstance() - create and return an instance of the vtkSQLQuery subclass associated with the database type.

The subclass should also provide API to set connection parameters.

This class also provides the function EffectSchema to transform a database schema into a SQL database.

.SECTION Thanks Thanks to Andrew Wilson from Sandia National Laboratories for his work on the database classes and for the SQLite example. Thanks to David Thompson and Philippe Pebay from Sandia National Laboratories for implementing this class.

To create an instance of class vtkSQLDatabase, simply invoke its constructor as follows

```
obj = vtkSQLDatabase
```

37.91.2 Methods

The class vtkSQLDatabase has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSQLDatabase class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSQLDatabase = obj.NewInstance ()`
- `vtkSQLDatabase = obj.SafeDownCast (vtkObject o)`
- `bool = obj.Open (string password)` - Open a new connection to the database. You need to set up any database parameters before calling this function. For database connections that do not require a password, pass an empty string. Returns true is the database was opened sucessfully, and false otherwise.
- `obj.Close ()` - Close the connection to the database.
- `bool = obj.IsOpen ()` - Return whether the database has an open connection.
- `vtkSQLQuery = obj.GetQueryInstance ()` - Return an empty query on this database.
- `bool = obj.HasError ()` - Did the last operation generate an error
- `string = obj.GetLastErrorText ()` - Get the last error text from the database I'm using const so that people do NOT use the standard vtkGetStringMacro in their implementation, because 99will not be the correct thing to do...
- `string = obj.GetDatabaseType ()` - Get the type of the database (e.g. mysql, psql,..).
- `vtkStringArray = obj.GetTables ()` - Get the list of tables from the database.
- `vtkStringArray = obj.GetRecord (string table)` - Get the list of fields for a particular table.
- `bool = obj.IsSupported (int )` - Get the URL of the database.
- `vtkStdString = obj.GetURL ()` - Get the URL of the database.
- `vtkStdString = obj.GetTablePreamble (bool )` - Return the SQL string with the syntax to create a column inside a "CREATE TABLE" SQL statement. NB: this method implements the following minimally-portable syntax: ¡column name¿ ¡column type¿ ¡column attributes¿ It must be overwritten for those SQL backends which have a different syntax such as, e.g., MySQL.
- `vtkStdString = obj.GetColumnSpecification (vtkSQLDatabaseSchema schema, int tblHandle, int colHandl` - Return the SQL string with the syntax to create a column inside a "CREATE TABLE" SQL statement. NB: this method implements the following minimally-portable syntax: ¡column name¿ ¡column type¿ ¡column attributes¿ It must be overwritten for those SQL backends which have a different syntax such as, e.g., MySQL.
- `vtkStdString = obj.GetTriggerSpecification (vtkSQLDatabaseSchema schema, int tblHandle, int trgHand` - Return the SQL string with the syntax to create a trigger using a "CREATE TRIGGER" SQL statement. NB1: support is contingent on VTK_FEATURE_TRIGGERS being recognized as a supported feature. Not all backends (e.g., SQLite) support it. NB2: this method implements the following minimally-portable syntax: ¡trigger name¿ BEFORE — AFTER ¡event¿ ON ¡table name¿ FOR EACH ROW ¡trigger action¿ It must be overwritten for those SQL backends which have a different syntax such as, e.g., PostgreSQL.
- `bool = obj.EffectSchema (vtkSQLDatabaseSchema , bool dropIfExistsfalse)` - Effect a database schema.

37.92 vtkSQLDatabaseSchema

37.92.1 Usage

A class to create a SQL database schema

.SECTION Thanks Thanks to Philippe Pebay and David Thompson from Sandia National Laboratories for implementing this class.

To create an instance of class vtkSQLDatabaseSchema, simply invoke its constructor as follows

```
obj = vtkSQLDatabaseSchema
```

37.92.2 Methods

The class vtkSQLDatabaseSchema has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSQLDatabaseSchema class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSQLDatabaseSchema = obj.NewInstance ()`
- `vtkSQLDatabaseSchema = obj.SafeDownCast (vtkObject o)`
- `int = obj.AddPreamble (string preName, string preAction, string preBackendVTK\_SQL\_ALLBACKENDS)`
- `int = obj.AddTable (string tblName)` - Add a table to the schema
- `int = obj.AddColumnToTable (int tblHandle, int colType, string colName, int colSize, string colAttr`
- `int = obj.AddColumnToTable (string tblName, int colType, string colName, int colSize, string colAtt`
- `int = obj.AddIndexToTable (int tblHandle, int idxType, string idxName)`
- `int = obj.AddIndexToTable (string tblName, int idxType, string idxName)`
- `int = obj.AddColumnToIndex (int tblHandle, int idxHandle, int colHandle)`
- `int = obj.AddColumnToIndex (string tblName, string idxName, string colName)`
- `int = obj.AddTriggerToTable (int tblHandle, int trgType, string trgName, string trgAction, string t`
- `int = obj.AddTriggerToTable (string tblName, int trgType, string trgName, string trgAction, string` - Given a preamble name, get its handle.
- `int = obj.GetPreambleHandleFromName (string preName)` - Given a preamble name, get its handle.
- `string = obj.GetPreambleNameFromHandle (int preHandle)` - Given a preamble handle, get its name.
- `string = obj.GetPreambleActionFromHandle (int preHandle)` - Given a preamble handle, get its action.
- `string = obj.GetPreambleBackendFromHandle (int preHandle)` - Given a preamble handle, get its backend.
- `int = obj.GetTableHandleFromName (string tblName)` - Given a table name, get its handle.
- `string = obj.GetTableNameFromHandle (int tblHandle)` - Given a table hanlde, get its name.

- `int = obj.GetIndexHandleFromName (string tblName, string idxName)` - Given the names of a table and an index, get the handle of the index in this table.
- `string = obj.GetIndexNameFromHandle (int tblHandle, int idxHandle)` - Given the handles of a table and an index, get the name of the index.
- `int = obj.GetIndexTypeFromHandle (int tblHandle, int idxHandle)` - Given the handles of a table and an index, get the type of the index.
- `string = obj.GetIndexColumnNameFromHandle (int tblHandle, int idxHandle, int cnmHandle)` - Given the handles of a table, an index, and a column name, get the column name.
- `int = obj.GetColumnHandleFromName (string tblName, string colName)` - Given the names of a table and a column, get the handle of the column in this table.
- `string = obj.GetColumnNameFromHandle (int tblHandle, int colHandle)` - Given the handles of a table and a column, get the name of the column.
- `int = obj.GetColumnTypeFromHandle (int tblHandle, int colHandle)` - Given the handles of a table and a column, get the type of the column.
- `int = obj.GetColumnSizeFromHandle (int tblHandle, int colHandle)` - Given the handles of a table and a column, get the size of the column.
- `string = obj.GetColumnAttributesFromHandle (int tblHandle, int colHandle)` - Given the handles of a table and a column, get the attributes of the column.
- `int = obj.GetTriggerHandleFromName (string tblName, string trgName)` - Given the names of a trigger and a table, get the handle of the trigger in this table.
- `string = obj.GetTriggerNameFromHandle (int tblHandle, int trgHandle)` - Given the handles of a table and a trigger, get the name of the trigger.
- `int = obj.GetTriggerTypeFromHandle (int tblHandle, int trgHandle)` - Given the handles of a table and a trigger, get the type of the trigger.
- `string = obj.GetTriggerActionFromHandle (int tblHandle, int trgHandle)` - Given the handles of a table and a trigger, get the action of the trigger.
- `string = obj.GetTriggerBackendFromHandle (int tblHandle, int trgHandle)` - Given the handles of a table and a trigger, get the backend of the trigger.
- `obj.Reset ()` - Reset the schema to its initial, empty state.
- `int = obj.GetNumberOfPreambles ()` - Get the number of preambles.
- `int = obj.GetNumberOfTables ()` - Get the number of tables.
- `int = obj.GetNumberOfColumnsInTable (int tblHandle)` - Get the number of columns in a particular table .
- `int = obj.GetNumberOfIndicesInTable (int tblHandle)` - Get the number of indices in a particular table .
- `int = obj.GetNumberOfColumnNamesInIndex (int tblHandle, int idxHandle)` - Get the number of column names associated to a particular index in a particular table .
- `int = obj.GetNumberOfTriggersInTable (int tblHandle)` - Get the number of trigger in a particular table .
- `obj.SetName (string )` - Set/Get the name of the schema.
- `string = obj.GetName ()` - Set/Get the name of the schema.

37.93 vtkSQLiteDatabase

37.93.1 Usage

SQLite (http://www.sqlite.org) is a public-domain SQL database written in C++. It's small, fast, and can be easily embedded inside other applications. Its databases are stored in files.

This class provides a VTK interface to SQLite. You do not need to download any external libraries: we include a copy of SQLite 3.3.16 in VTK/Utilities/vtksqlite.

If you want to open a database that stays in memory and never gets written to disk, pass in the URL 'sqlite://:memory:'; otherwise, specifiy the file path by passing the URL 'sqlite://¡file_path¿'.

.SECTION Thanks Thanks to Andrew Wilson and Philippe Pebay from Sandia National Laboratories for implementing this class.

To create an instance of class vtkSQLiteDatabase, simply invoke its constructor as follows

```
obj = vtkSQLiteDatabase
```

37.93.2 Methods

The class vtkSQLiteDatabase has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSQLiteDatabase class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSQLiteDatabase = obj.NewInstance ()`
- `vtkSQLiteDatabase = obj.SafeDownCast (vtkObject o)`
- `bool = obj.Open (string password)` - Open a new connection to the database. You need to set the filename before calling this function. Returns true if the database was opened successfully; false otherwise. - USE_EXISTING (default) - Fail if the file does not exist. - USE_EXISTING_OR_CREATE - Create a new file if necessary. - CREATE_OR_CLEAR - Create new or clear existing file. - CREATE - Create new, fail if file exists.
- `bool = obj.Open (string password, int mode)` - Open a new connection to the database. You need to set the filename before calling this function. Returns true if the database was opened successfully; false otherwise. - USE_EXISTING (default) - Fail if the file does not exist. - USE_EXISTING_OR_CREATE - Create a new file if necessary. - CREATE_OR_CLEAR - Create new or clear existing file. - CREATE - Create new, fail if file exists.
- `obj.Close ()` - Close the connection to the database.
- `bool = obj.IsOpen ()` - Return whether the database has an open connection
- `vtkSQLQuery = obj.GetQueryInstance ()` - Return an empty query on this database.
- `vtkStringArray = obj.GetTables ()` - Get the list of tables from the database
- `vtkStringArray = obj.GetRecord (string table)` - Get the list of fields for a particular table
- `bool = obj.IsSupported (int feature)` - Return whether a feature is supported by the database.
- `bool = obj.HasError ()` - Did the last operation generate an error
- `string = obj.GetLastErrorText ()` - Get the last error text from the database
- `string = obj.GetDatabaseType ()` - String representing database type (e.g. "sqlite").

- `string = obj.GetDatabaseFileName ()` - String representing the database filename.
- `obj.SetDatabaseFileName (string )` - String representing the database filename.
- `vtkStdString = obj.GetURL ()` - Get the URL of the database.
- `vtkStdString = obj.GetColumnSpecification (vtkSQLDatabaseSchema schema, int tblHandle, int colHandl` - Return the SQL string with the syntax to create a column inside a "CREATE TABLE" SQL statement. NB: this method implements the SQLite-specific syntax: ¡column name¿ ¡column type¿ ¡column attributes¿

37.94 vtkSQLiteQuery

37.94.1 Usage

This is an implementation of vtkSQLQuery for SQLite databases. See the documentation for vtkSQLQuery for information about what the methods do.

.SECTION Bugs

Sometimes Execute() will return false (meaning an error) but GetLastErrorText() winds up null. I am not certain why this is happening.

.SECTION Thanks Thanks to Andrew Wilson from Sandia National Laboratories for implementing this class.

To create an instance of class vtkSQLiteQuery, simply invoke its constructor as follows

```
obj = vtkSQLiteQuery
```

37.94.2 Methods

The class vtkSQLiteQuery has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSQLiteQuery class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSQLiteQuery = obj.NewInstance ()`
- `vtkSQLiteQuery = obj.SafeDownCast (vtkObject o)`
- `bool = obj.SetQuery (string query)` - Set the SQL query string. This must be performed before Execute() or BindParameter() can be called.
- `bool = obj.Execute ()` - Execute the query. This must be performed before any field name or data access functions are used.
- `int = obj.GetNumberOfFields ()` - The number of fields in the query result.
- `string = obj.GetFieldName (int i)` - Return the name of the specified query field.
- `int = obj.GetFieldType (int i)` - Return the type of the field, using the constants defined in vtkType.h.
- `bool = obj.NextRow ()` - Advance row, return false if past end.
- `bool = obj.HasError ()` - Return true if there is an error on the current query.
- `bool = obj.BeginTransaction ()` - Begin, abort (roll back), or commit a transaction.

- `bool = obj.RollbackTransaction ()` - Begin, abort (roll back), or commit a transaction.
- `bool = obj.CommitTransaction ()` - Begin, abort (roll back), or commit a transaction.
- `string = obj.GetLastErrorText ()` - Get the last error text from the query
- `bool = obj.BindParameter (int index, int value)` - The following methods bind a parameter value to a placeholder in the SQL string. See the documentation for vtkSQLQuery for further explanation. The driver makes internal copies of string and BLOB parameters so you don't need to worry about keeping them in scope until the query finishes executing.
- `bool = obj.BindParameter (int index, float value)` - The following methods bind a parameter value to a placeholder in the SQL string. See the documentation for vtkSQLQuery for further explanation. The driver makes internal copies of string and BLOB parameters so you don't need to worry about keeping them in scope until the query finishes executing.
- `bool = obj.BindParameter (int index, double value)` - The following methods bind a parameter value to a placeholder in the SQL string. See the documentation for vtkSQLQuery for further explanation. The driver makes internal copies of string and BLOB parameters so you don't need to worry about keeping them in scope until the query finishes executing.
- `bool = obj.BindParameter (int index, string stringValue)` - Bind a string value – string must be null-terminated
- `bool = obj.ClearParameterBindings ()` - Bind a blob value. Not all databases support blobs as a data type. Check vtkSQLDatabase::IsSupported(VTK_SQL_FEATURE_BLOB) to make sure.

37.95 vtkSQLQuery

37.95.1 Usage

The abstract superclass of SQL query classes. Instances of subclasses of vtkSQLQuery are created using the GetQueryInstance() function in vtkSQLDatabase. To implement a query connection for a new database type, subclass both vtkSQLDatabase and vtkSQLQuery, and implement the required functions. For the query class, this involves the following:

Execute() - Execute the query on the database. No results need to be retrieved at this point, unless you are performing caching.

GetNumberOfFields() - After Execute() is performed, returns the number of fields in the query results.

GetFieldName() - The name of the field at an index.

GetFieldType() - The data type of the field at an index.

NextRow() - Advances the query results by one row, and returns whether there are more rows left in the query.

DataValue() - Extract a single data value from the current row.

Begin/Rollback/CommitTransaction() - These methods are optional but recommended if the database supports transactions.

.SECTION Thanks Thanks to Andrew Wilson from Sandia National Laboratories for his work on the database classes.

To create an instance of class vtkSQLQuery, simply invoke its constructor as follows

```
obj = vtkSQLQuery
```

37.95.2 Methods

The class vtkSQLQuery has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSQLQuery class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSQLQuery = obj.NewInstance ()`
- `vtkSQLQuery = obj.SafeDownCast (vtkObject o)`
- `bool = obj.SetQuery (string query)` - The query string to be executed. Since some databases will process the query string as soon as it's set, this method returns a boolean to indicate success or failure.
- `string = obj.GetQuery ()` - The query string to be executed. Since some databases will process the query string as soon as it's set, this method returns a boolean to indicate success or failure.
- `bool = obj.IsActive ()` - Execute the query. This must be performed before any field name or data access functions are used.
- `bool = obj.Execute ()` - Execute the query. This must be performed before any field name or data access functions are used.
- `bool = obj.BeginTransaction ()` - Begin, commit, or roll back a transaction. If the underlying database does not support transactions these calls will do nothing.
- `bool = obj.CommitTransaction ()` - Begin, commit, or roll back a transaction. If the underlying database does not support transactions these calls will do nothing.
- `bool = obj.RollbackTransaction ()` - Return the database associated with the query.
- `vtkSQLDatabase = obj.GetDatabase ()` - Return the database associated with the query.
- `bool = obj.BindParameter (int index, int value)`
- `bool = obj.BindParameter (int index, float value)`
- `bool = obj.BindParameter (int index, double value)`
- `bool = obj.BindParameter (int index, string stringValue)` - Bind a string value – string must be null-terminated
- `bool = obj.ClearParameterBindings ()` - Reset all parameter bindings to NULL.
- `string = obj.EscapeString (string src, bool addSurroundingQuotes)` - Escape a string for inclusion into an SQL query. This method exists to provide a wrappable version of the method that takes and returns vtkStdString objects. You are responsible for calling delete [] on the character array returned by this method. This method simply calls the vtkStdString variant and thus need not be re-implemented by subclasses.

37.96 vtkSTLReader

37.96.1 Usage

vtkSTLReader is a source object that reads ASCII or binary stereo lithography files (.stl files). The FileName must be specified to vtkSTLReader. The object automatically detects whether the file is ASCII or binary.

.stl files are quite inefficient since they duplicate vertex definitions. By setting the Merging boolean you can control whether the point data is merged after reading. Merging is performed by default, however, merging requires a large amount of temporary storage since a 3D hash table must be constructed.

To create an instance of class vtkSTLReader, simply invoke its constructor as follows

```
obj = vtkSTLReader
```

37.96.2 Methods

The class vtkSTLReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSTLReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSTLReader = obj.NewInstance ()`
- `vtkSTLReader = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Overload standard modified time function. If locator is modified, then this object is modified as well.
- `obj.SetFileName (string )` - Specify file name of stereo lithography file.
- `string = obj.GetFileName ()` - Specify file name of stereo lithography file.
- `obj.SetMerging (int )` - Turn on/off merging of points/triangles.
- `int = obj.GetMerging ()` - Turn on/off merging of points/triangles.
- `obj.MergingOn ()` - Turn on/off merging of points/triangles.
- `obj.MergingOff ()` - Turn on/off merging of points/triangles.
- `obj.SetScalarTags (int )` - Turn on/off tagging of solids with scalars.
- `int = obj.GetScalarTags ()` - Turn on/off tagging of solids with scalars.
- `obj.ScalarTagsOn ()` - Turn on/off tagging of solids with scalars.
- `obj.ScalarTagsOff ()` - Turn on/off tagging of solids with scalars.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Specify a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Specify a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified.

37.97 vtkSTLWriter

37.97.1 Usage

vtkSTLWriter writes stereo lithography (.stl) files in either ASCII or binary form. Stereo lithography files only contain triangles. If polygons with more than 3 vertices are present, only the first 3 vertices are written. Use vtkTriangleFilter to convert polygons to triangles.

To create an instance of class vtkSTLWriter, simply invoke its constructor as follows

```
obj = vtkSTLWriter
```

37.97.2 Methods

The class vtkSTLWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSTLWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSTLWriter = obj.NewInstance ()`
- `vtkSTLWriter = obj.SafeDownCast (vtkObject o)`

37.98 vtkStructuredGridReader

37.98.1 Usage

vtkStructuredGridReader is a source object that reads ASCII or binary structured grid data files in vtk format. (see text for format details). The output of this reader is a single vtkStructuredGrid data object. The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkStructuredGridReader, simply invoke its constructor as follows

```
obj = vtkStructuredGridReader
```

37.98.2 Methods

The class vtkStructuredGridReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGridReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredGridReader = obj.NewInstance ()`
- `vtkStructuredGridReader = obj.SafeDownCast (vtkObject o)`
- `vtkStructuredGrid = obj.GetOutput ()` - Get the output of this reader.
- `vtkStructuredGrid = obj.GetOutput (int idx)` - Get the output of this reader.
- `obj.SetOutput (vtkStructuredGrid output)` - Get the output of this reader.
- `int = obj.ReadMetaData (vtkInformation outInfo)` - Read the meta information from the file. This needs to be public to it can be accessed by vtkDataSetReader.

37.99 vtkStructuredGridWriter

37.99.1 Usage

vtkStructuredGridWriter is a source object that writes ASCII or binary structured grid data files in vtk format. See text for format details.

To create an instance of class vtkStructuredGridWriter, simply invoke its constructor as follows

```
obj = vtkStructuredGridWriter
```

37.99.2 Methods

The class vtkStructuredGridWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredGridWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredGridWriter = obj.NewInstance ()`
- `vtkStructuredGridWriter = obj.SafeDownCast (vtkObject o)`
- `vtkStructuredGrid = obj.GetInput ()` - Get the input to this writer.
- `vtkStructuredGrid = obj.GetInput (int port)` - Get the input to this writer.

37.100 vtkStructuredPointsReader

37.100.1 Usage

vtkStructuredPointsReader is a source object that reads ASCII or binary structured points data files in vtk format (see text for format details). The output of this reader is a single vtkStructuredPoints data object. The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkStructuredPointsReader, simply invoke its constructor as follows

```
obj = vtkStructuredPointsReader
```

37.100.2 Methods

The class vtkStructuredPointsReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredPointsReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredPointsReader = obj.NewInstance ()`
- `vtkStructuredPointsReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetOutput (vtkStructuredPoints output)` - Set/Get the output of this reader.
- `vtkStructuredPoints = obj.GetOutput (int idx)` - Set/Get the output of this reader.
- `vtkStructuredPoints = obj.GetOutput ()` - Set/Get the output of this reader.
- `int = obj.ReadMetaData (vtkInformation outInfo)` - Read the meta information from the file. This needs to be public to it can be accessed by vtkDataSetReader.

37.101 vtkStructuredPointsWriter

37.101.1 Usage

vtkStructuredPointsWriter is a source object that writes ASCII or binary structured points data in vtk file format. See text for format details.

To create an instance of class vtkStructuredPointsWriter, simply invoke its constructor as follows

```
obj = vtkStructuredPointsWriter
```

37.101.2 Methods

The class vtkStructuredPointsWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStructuredPointsWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStructuredPointsWriter = obj.NewInstance ()`
- `vtkStructuredPointsWriter = obj.SafeDownCast (vtkObject o)`
- `vtkImageData = obj.GetInput ()` - Get the input to this writer.
- `vtkImageData = obj.GetInput (int port)` - Get the input to this writer.

37.102 vtkTableReader

37.102.1 Usage

vtkTableReader is a source object that reads ASCII or binary vtkTable data files in vtk format. (see text for format details). The output of this reader is a single vtkTable data object. The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkTableReader, simply invoke its constructor as follows

```
obj = vtkTableReader
```

37.102.2 Methods

The class vtkTableReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableReader = obj.NewInstance ()`
- `vtkTableReader = obj.SafeDownCast (vtkObject o)`
- `vtkTable = obj.GetOutput ()` - Get the output of this reader.
- `vtkTable = obj.GetOutput (int idx)` - Get the output of this reader.
- `obj.SetOutput (vtkTable output)` - Get the output of this reader.

37.103 vtkTableWriter

37.103.1 Usage

vtkTableWriter is a sink object that writes ASCII or binary vtkTable data files in vtk format. See text for format details.

To create an instance of class vtkTableWriter, simply invoke its constructor as follows

```
obj = vtkTableWriter
```

37.103.2 Methods

The class vtkTableWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTableWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTableWriter = obj.NewInstance ()`
- `vtkTableWriter = obj.SafeDownCast (vtkObject o)`
- `vtkTable = obj.GetInput ()` - Get the input to this writer.
- `vtkTable = obj.GetInput (int port)` - Get the input to this writer.

37.104 vtkTecplotReader

37.104.1 Usage

vtkTecplotReader parses an ASCII Tecplot file to get a vtkMultiBlockDataSet object made up of several vtkDataSet objects, of which each is of type either vtkStructuredGrid or vtkUnstructuredGrid. Each vtkDataSet object maintains the geometry, topology, and some associated attributes describing physical properties.

Tecplot treats 3D coordinates (only one or two coordinates might be explicitly specified in a file) as varaibles too, whose names (e.g., 'X' / 'x' / 'I', 'Y' / 'y' / 'J', 'Z' / 'z' / 'K') are provided in the variables list (the 'VARIABLES' section). These names are then followed in the list by those of other traditional variables or attributes (node- based and / or cell-based data with the mode specified via token 'VAR LOCATION', to be extracted to create vtkPointData and / or vtkCellData). Each zone described afterwards (in the 'ZONE's section) provides the specific values of the aforementioned variables (including 3D coordinates), in the same order as indicated by the variable-names list, through either POINT-packing (i.e., tuple-based storage) or BLOCK-packing (component-based storage). In particular, the first / description line of each zone tells the type of all the constituent cells as the connectivity / topology information. In other words, the entire dataset is made up of multiple zones (blocks), of which each maintains a set of cells of the same type ('BRICK', 'TRIANGLE', 'QUADRILATERAL', 'TETRAHEDRON', and 'POINT' in Tecplot terms). In addition, the description line of each zone specifies the zone name, dimensionality information (size of each dimension for a structured zone), number of nodes, and number of cells. Information about the file format is available at http://download.tecplot.com/360/dataformat.pdf.

To create an instance of class vtkTecplotReader, simply invoke its constructor as follows

```
obj = vtkTecplotReader
```

37.104.2 Methods

The class vtkTecplotReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTecplotReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTecplotReader = obj.NewInstance ()`
- `vtkTecplotReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfVariables ()` - Get the number of all variables (including 3D coordinates).
- `obj.SetFileName (string fileName)` - Specify a Tecplot ASCII file for data loading.
- `string = obj.GetDataTitle ()` - Get the Tecplot data title.
- `int = obj.GetNumberOfBlocks ()` - Get the number of blocks (i.e., zones in Tecplot terms).
- `string = obj.GetBlockName (int blockIdx)` - Get the name of a block specified by a zero-based index. NULL is returned for an invalid block index.
- `int = obj.GetNumberOfDataAttributes ()` - Get the number of standard data attributes (node-based and cell-based), excluding 3D coordinates.
- `string = obj.GetDataAttributeName (int attrIndx)` - Get the name of a zero-based data attribute (not 3D coordinates). NULL is returned for an invalid attribute index.
- `int = obj.IsDataAttributeCellBased (string attrName)` - Get the type (0 for node-based and 1 for cell-based) of a specified data attribute (not 3D coordinates). -1 is returned for an invalid attribute name.
- `int = obj.IsDataAttributeCellBased (int attrIndx)` - Get the type (0 for node-based and 1 for cell-based) of a specified data attribute (not 3D coordinates). -1 is returned for an invalid attribute index.
- `int = obj.GetNumberOfDataArrays ()` - Get the number of all data attributes (point data and cell data).
- `string = obj.GetDataArrayName (int arrayIdx)` - Get the name of a data array specified by the zero-based index (arrayIdx).
- `int = obj.GetDataArrayStatus (string arayName)` - Get the status of a specific data array (0: un-selected; 1: selected).
- `obj.SetDataArrayStatus (string arayName, int bChecked)` - Set the status of a specific data array (0: de-select; 1: select) specified by the name.

37.105 vtkTIFFReader

37.105.1 Usage

vtkTIFFReader is a source object that reads TIFF files. It should be able to read almost any TIFF file
To create an instance of class vtkTIFFReader, simply invoke its constructor as follows

```
obj = vtkTIFFReader
```

37.105.2 Methods

The class vtkTIFFReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTIFFReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTIFFReader = obj.NewInstance ()`
- `vtkTIFFReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string fname)` - Is the given file name a tiff file file?
- `string = obj.GetFileExtensions ()` - Return a descriptive name for the file format that might be useful in a GUI.
- `string = obj.GetDescriptiveName ()` - Auxilary methods used by the reader internally.
- `obj.InitializeColors ()` - Auxilary methods used by the reader internally.
- `obj.SetOrientationType (int orientationType)` - Set orientation type ORIENTATION_TOPLEFT 1 (row 0 top, col 0 lhs) ORIENTATION_TOPRIGHT 2 (row 0 top, col 0 rhs) ORIENTATION_BOTRIGHT 3 (row 0 bottom, col 0 rhs) ORIENTATION_BOTLEFT 4 (row 0 bottom, col 0 lhs) ORIENTATION_LEFTTOP 5 (row 0 lhs, col 0 top) ORIENTATION_RIGHTTOP 6 (row 0 rhs, col 0 top) ORIENTATION_RIGHTBOT 7 (row 0 rhs, col 0 bottom) ORIENTATION_LEFTBOT 8 (row 0 lhs, col 0 bottom) User need to explicitely include vtk_tiff.h header to have access to those #define
- `int = obj.GetOrientationType ()` - Set orientation type ORIENTATION_TOPLEFT 1 (row 0 top, col 0 lhs) ORIENTATION_TOPRIGHT 2 (row 0 top, col 0 rhs) ORIENTATION_BOTRIGHT 3 (row 0 bottom, col 0 rhs) ORIENTATION_BOTLEFT 4 (row 0 bottom, col 0 lhs) ORIENTATION_LEFTTOP 5 (row 0 lhs, col 0 top) ORIENTATION_RIGHTTOP 6 (row 0 rhs, col 0 top) ORIENTATION_RIGHTBOT 7 (row 0 rhs, col 0 bottom) ORIENTATION_LEFTBOT 8 (row 0 lhs, col 0 bottom) User need to explicitely include vtk_tiff.h header to have access to those #define
- `bool = obj.GetOrientationTypeSpecifiedFlag ()` - Get method to check if orientation type is specified
- `obj.SetOriginSpecifiedFlag (bool )` - Set/get methods to see if manual Origin/Spacing have been set.
- `bool = obj.GetOriginSpecifiedFlag ()` - Set/get methods to see if manual Origin/Spacing have been set.
- `obj.OriginSpecifiedFlagOn ()` - Set/get methods to see if manual Origin/Spacing have been set.
- `obj.OriginSpecifiedFlagOff ()` - Set/get methods to see if manual Origin/Spacing have been set.
- `obj.SetSpacingSpecifiedFlag (bool )` -
- `bool = obj.GetSpacingSpecifiedFlag ()` -
- `obj.SpacingSpecifiedFlagOn ()` -
- `obj.SpacingSpecifiedFlagOff ()` -

37.106 vtkTIFFWriter

37.106.1 Usage

vtkTIFFWriter writes image data as a TIFF data file. Data can be written uncompressed or compressed. Several forms of compression are supported including packed bits, JPEG, deflation, and LZW. (Note: LZW compression is currently under patent in the US and is disabled until the patent expires. However, the mechanism for supporting this compression is available for those with a valid license or to whom the patent does not apply.)

To create an instance of class vtkTIFFWriter, simply invoke its constructor as follows

```
obj = vtkTIFFWriter
```

37.106.2 Methods

The class vtkTIFFWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTIFFWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTIFFWriter = obj.NewInstance ()`
- `vtkTIFFWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetCompression (int )` - Set compression type. Sinze LZW compression is patented outside US, the additional work steps have to be taken in order to use that compression.
- `int = obj.GetCompressionMinValue ()` - Set compression type. Sinze LZW compression is patented outside US, the additional work steps have to be taken in order to use that compression.
- `int = obj.GetCompressionMaxValue ()` - Set compression type. Sinze LZW compression is patented outside US, the additional work steps have to be taken in order to use that compression.
- `int = obj.GetCompression ()` - Set compression type. Sinze LZW compression is patented outside US, the additional work steps have to be taken in order to use that compression.
- `obj.SetCompressionToNoCompression ()` - Set compression type. Sinze LZW compression is patented outside US, the additional work steps have to be taken in order to use that compression.
- `obj.SetCompressionToPackBits ()` - Set compression type. Sinze LZW compression is patented outside US, the additional work steps have to be taken in order to use that compression.
- `obj.SetCompressionToJPEG ()` - Set compression type. Sinze LZW compression is patented outside US, the additional work steps have to be taken in order to use that compression.
- `obj.SetCompressionToDeflate ()` - Set compression type. Sinze LZW compression is patented outside US, the additional work steps have to be taken in order to use that compression.
- `obj.SetCompressionToLZW ()`

37.107 vtkTreeReader

37.107.1 Usage

vtkTreeReader is a source object that reads ASCII or binary vtkTree data files in vtk format. (see text for format details). The output of this reader is a single vtkTree data object. The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkTreeReader, simply invoke its constructor as follows

```
obj = vtkTreeReader
```

37.107.2 Methods

The class vtkTreeReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeReader = obj.NewInstance ()`
- `vtkTreeReader = obj.SafeDownCast (vtkObject o)`
- `vtkTree = obj.GetOutput ()` - Get the output of this reader.
- `vtkTree = obj.GetOutput (int idx)` - Get the output of this reader.
- `obj.SetOutput (vtkTree output)` - Get the output of this reader.

37.108 vtkTreeWriter

37.108.1 Usage

vtkTreeWriter is a sink object that writes ASCII or binary vtkTree data files in vtk format. See text for format details.

To create an instance of class vtkTreeWriter, simply invoke its constructor as follows

```
obj = vtkTreeWriter
```

37.108.2 Methods

The class vtkTreeWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeWriter = obj.NewInstance ()`
- `vtkTreeWriter = obj.SafeDownCast (vtkObject o)`
- `vtkTree = obj.GetInput ()` - Get the input to this writer.
- `vtkTree = obj.GetInput (int port)` - Get the input to this writer.

37.109 vtkUGFacetReader

37.109.1 Usage

vtkUGFacetReader is a source object that reads Unigraphics facet files. Unigraphics is a solid modeling system; facet files are the polygonal plot files it uses to create 3D plots.

To create an instance of class vtkUGFacetReader, simply invoke its constructor as follows

```
obj = vtkUGFacetReader
```

37.109.2 Methods

The class vtkUGFacetReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUGFacetReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUGFacetReader = obj.NewInstance ()`
- `vtkUGFacetReader = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Overload standard modified time function. If locator is modified, then this object is modified as well.
- `obj.SetFileName (string )` - Specify Unigraphics file name.
- `string = obj.GetFileName ()` - Specify Unigraphics file name.
- `int = obj.GetNumberOfParts ()` - Special methods for interrogating the data file.
- `short = obj.GetPartColorIndex (int partId)` - Retrieve color index for the parts in the file.
- `obj.SetPartNumber (int )` - Specify the desired part to extract. The part number must range between [0,NumberOfParts-1]. If the value is =(-1), then all parts will be extracted. If the value is ¡(-1), then no parts will be extracted but the part colors will be updated.
- `int = obj.GetPartNumber ()` - Specify the desired part to extract. The part number must range between [0,NumberOfParts-1]. If the value is =(-1), then all parts will be extracted. If the value is ¡(-1), then no parts will be extracted but the part colors will be updated.
- `obj.SetMerging (int )` - Turn on/off merging of points/triangles.
- `int = obj.GetMerging ()` - Turn on/off merging of points/triangles.
- `obj.MergingOn ()` - Turn on/off merging of points/triangles.
- `obj.MergingOff ()` - Turn on/off merging of points/triangles.
- `obj.SetLocator (vtkIncrementalPointLocator locator)` - Specify a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `vtkIncrementalPointLocator = obj.GetLocator ()` - Specify a spatial locator for merging points. By default an instance of vtkMergePoints is used.
- `obj.CreateDefaultLocator ()` - Create default locator. Used to create one when none is specified.

37.110 vtkUnstructuredGridReader

37.110.1 Usage

vtkUnstructuredGridReader is a source object that reads ASCII or binary unstructured grid data files in vtk format. (see text for format details). The output of this reader is a single vtkUnstructuredGrid data object. The superclass of this class, vtkDataReader, provides many methods for controlling the reading of the data file, see vtkDataReader for more information.

To create an instance of class vtkUnstructuredGridReader, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridReader
```

37.110.2 Methods

The class vtkUnstructuredGridReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridReader = obj.NewInstance ()`
- `vtkUnstructuredGridReader = obj.SafeDownCast (vtkObject o)`
- `vtkUnstructuredGrid = obj.GetOutput ()` - Get the output of this reader.
- `vtkUnstructuredGrid = obj.GetOutput (int idx)` - Get the output of this reader.
- `obj.SetOutput (vtkUnstructuredGrid output)` - Get the output of this reader.

37.111 vtkUnstructuredGridWriter

37.111.1 Usage

vtkUnstructuredGridWriter is a source object that writes ASCII or binary unstructured grid data files in vtk format. See text for format details.

To create an instance of class vtkUnstructuredGridWriter, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridWriter
```

37.111.2 Methods

The class vtkUnstructuredGridWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridWriter = obj.NewInstance ()`
- `vtkUnstructuredGridWriter = obj.SafeDownCast (vtkObject o)`
- `vtkUnstructuredGrid = obj.GetInput ()` - Get the input to this writer.
- `vtkUnstructuredGrid = obj.GetInput (int port)` - Get the input to this writer.

37.112 vtkVolume16Reader

37.112.1 Usage

vtkVolume16Reader is a source object that reads 16 bit image files.

Volume16Reader creates structured point datasets. The dimension of the dataset depends upon the number of files read. Reading a single file results in a 2D image, while reading more than one file results in a 3D volume.

File names are created using FilePattern and FilePrefix as follows: sprintf (filename, FilePattern, FilePrefix, number); where number is in the range ImageRange[0] to ImageRange[1]. If ImageRange[1] ¡= ImageRange[0], then slice number ImageRange[0] is read. Thus to read an image set ImageRange[0] = ImageRange[1] = slice number. The default behavior is to read a single file (i.e., image slice 1).

The DataMask instance variable is used to read data files with imbedded connectivity or segmentation information. For example, some data has the high order bit set to indicate connected surface. The DataMask allows you to select this data. Other important ivars include HeaderSize, which allows you to skip over initial info, and SwapBytes, which turns on/off byte swapping.

The Transform instance variable specifies a permutation transformation to map slice space into world space. vtkImageReader has replaced the functionality of this class and should be used instead.

To create an instance of class vtkVolume16Reader, simply invoke its constructor as follows

```
obj = vtkVolume16Reader
```

37.112.2 Methods

The class vtkVolume16Reader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolume16Reader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolume16Reader = obj.NewInstance ()`
- `vtkVolume16Reader = obj.SafeDownCast (vtkObject o)`
- `obj.SetDataDimensions (int , int )` - Specify the dimensions for the data.
- `obj.SetDataDimensions (int  a[2])` - Specify the dimensions for the data.
- `int = obj. GetDataDimensions ()` - Specify the dimensions for the data.
- `obj.SetDataMask (short )` - Specify a mask used to eliminate data in the data file (e.g., connectivity bits).
- `short = obj.GetDataMask ()` - Specify a mask used to eliminate data in the data file (e.g., connectivity bits).
- `obj.SetHeaderSize (int )` - Specify the number of bytes to seek over at start of image.
- `int = obj.GetHeaderSize ()` - Specify the number of bytes to seek over at start of image.
- `obj.SetDataByteOrderToBigEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.

- `obj.SetDataByteOrderToLittleEndian ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.
- `int = obj.GetDataByteOrder ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.
- `obj.SetDataByteOrder (int )` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.
- `string = obj.GetDataByteOrderAsString ()` - These methods should be used instead of the SwapBytes methods. They indicate the byte ordering of the file you are trying to read in. These methods will then either swap or not swap the bytes depending on the byte ordering of the machine it is being run on. For example, reading in a BigEndian file on a BigEndian machine will result in no swapping. Trying to read the same file on a LittleEndian machine will result in swapping. As a quick note most UNIX machines are BigEndian while PC's and VAX tend to be LittleEndian. So if the file you are reading in was generated on a VAX or PC, SetDataByteOrderToLittleEndian otherwise SetDataByteOrderToBigEndian.
- `obj.SetSwapBytes (int )` - Turn on/off byte swapping.
- `int = obj.GetSwapBytes ()` - Turn on/off byte swapping.
- `obj.SwapBytesOn ()` - Turn on/off byte swapping.
- `obj.SwapBytesOff ()` - Turn on/off byte swapping.
- `obj.SetTransform (vtkTransform )` - Set/Get transformation matrix to transform the data from slice space into world space. This matrix must be a permutation matrix. To qualify, the sums of the rows must be + or - 1.
- `vtkTransform = obj.GetTransform ()` - Set/Get transformation matrix to transform the data from slice space into world space. This matrix must be a permutation matrix. To qualify, the sums of the rows must be + or - 1.
- `vtkImageData = obj.GetImage (int ImageNumber)` - Other objects make use of these methods

37.113 vtkVolumeReader

37.113.1 Usage

vtkVolumeReader is a source object that reads image files.

VolumeReader creates structured point datasets. The dimension of the dataset depends upon the number of files read. Reading a single file results in a 2D image, while reading more than one file results in a 3D volume.

File names are created using FilePattern and FilePrefix as follows: sprintf (filename, FilePattern, FilePrefix, number); where number is in the range ImageRange[0] to ImageRange[1]. If ImageRange[1] ¡= ImageRange[0], then slice number ImageRange[0] is read. Thus to read an image set ImageRange[0] = ImageRange[1] = slice number. The default behavior is to read a single file (i.e., image slice 1).

The DataMask instance variable is used to read data files with imbedded connectivity or segmentation information. For example, some data has the high order bit set to indicate connected surface. The DataMask allows you to select this data. Other important ivars include HeaderSize, which allows you to skip over initial info, and SwapBytes, which turns on/off byte swapping. Consider using vtkImageReader as a replacement.

To create an instance of class vtkVolumeReader, simply invoke its constructor as follows

```
obj = vtkVolumeReader
```

37.113.2 Methods

The class vtkVolumeReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeReader = obj.NewInstance ()`
- `vtkVolumeReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFilePrefix (string )` - Specify file prefix for the image file(s).
- `string = obj.GetFilePrefix ()` - Specify file prefix for the image file(s).
- `obj.SetFilePattern (string )` - The sprintf format used to build filename from FilePrefix and number.
- `string = obj.GetFilePattern ()` - The sprintf format used to build filename from FilePrefix and number.
- `obj.SetImageRange (int , int )` - Set the range of files to read.
- `obj.SetImageRange (int  a[2])` - Set the range of files to read.
- `int = obj. GetImageRange ()` - Set the range of files to read.
- `obj.SetDataSpacing (double , double , double )` - Specify the spacing for the data.
- `obj.SetDataSpacing (double  a[3])` - Specify the spacing for the data.
- `double = obj. GetDataSpacing ()` - Specify the spacing for the data.
- `obj.SetDataOrigin (double , double , double )` - Specify the origin for the data.
- `obj.SetDataOrigin (double  a[3])` - Specify the origin for the data.
- `double = obj. GetDataOrigin ()` - Specify the origin for the data.
- `vtkImageData = obj.GetImage (int ImageNumber)` - Other objects make use of this method.

37.114 vtkWriter

37.114.1 Usage

vtkWriter is an abstract class for mapper objects that write their data to disk (or into a communications port). All writers respond to Write() method. This method insures that there is input and input is up to date.

To create an instance of class vtkWriter, simply invoke its constructor as follows

```
obj = vtkWriter
```

37.114.2 Methods

The class vtkWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWriter = obj.NewInstance ()`
- `vtkWriter = obj.SafeDownCast (vtkObject o)`
- `int = obj.Write ()` - Write data to output. Method executes subclasses WriteData() method, as well as StartMethod() and EndMethod() methods. Returns 1 on success and 0 on failure.
- `obj.EncodeString (string resname, string name, bool doublePercent)` - Encode the string so that the reader will not have problems. The resulting string is up to three times the size of the input string. doublePercent indicates whether to output a double 'escaped characters so the string may be used as a printf format string.
- `obj.SetInput (vtkDataObject input)` - Set/get the input to this writer.
- `obj.SetInput (int index, vtkDataObject input)` - Set/get the input to this writer.

37.115 vtkXMLCompositeDataReader

37.115.1 Usage

vtkXMLCompositeDataReader reads the VTK XML multi-group data file format. XML multi-group data files are meta-files that point to a list of serial VTK XML files. When reading in parallel, it will distribute sub-blocks among processor. If the number of sub-blocks is less than the number of processors, some processors will not have any sub-blocks for that group. If the number of sub-blocks is larger than the number of processors, each processor will possibly have more than 1 sub-block.

To create an instance of class vtkXMLCompositeDataReader, simply invoke its constructor as follows

```
obj = vtkXMLCompositeDataReader
```

37.115.2 Methods

The class vtkXMLCompositeDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLCompositeDataReader class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkXMLCompositeDataReader = obj.NewInstance ()`
- `vtkXMLCompositeDataReader = obj.SafeDownCast (vtkObject o)`
- `vtkCompositeDataSet = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkCompositeDataSet = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.

37.116 vtkXMLCompositeDataWriter

37.116.1 Usage

vtkXMLCompositeDataWriter writes (serially) the VTK XML multi-group, multi-block hierarchical and hierarchical box files. XML multi-group data files are meta-files that point to a list of serial VTK XML files.

To create an instance of class vtkXMLCompositeDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLCompositeDataWriter
```

37.116.2 Methods

The class vtkXMLCompositeDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLCompositeDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLCompositeDataWriter = obj.NewInstance ()`
- `vtkXMLCompositeDataWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.
- `int = obj.GetGhostLevel ()` - Get/Set the number of ghost levels to be written.
- `obj.SetGhostLevel (int )` - Get/Set the number of ghost levels to be written.
- `int = obj.GetWriteMetaFile ()` - Get/Set whether this instance will write the meta-file.
- `obj.SetWriteMetaFile (int flag)` - Get/Set whether this instance will write the meta-file.

37.117 vtkXMLDataParser

37.117.1 Usage

vtkXMLDataParser provides a subclass of vtkXMLParser that constructs a representation of an XML data format's file using vtkXMLDataElement to represent each XML element. This representation is then used by vtkXMLReader and its subclasses to traverse the structure of the file and extract data.

To create an instance of class vtkXMLDataParser, simply invoke its constructor as follows

```
obj = vtkXMLDataParser
```

37.117.2 Methods

The class vtkXMLDataParser has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLDataParser class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLDataParser = obj.NewInstance ()`
- `vtkXMLDataParser = obj.SafeDownCast (vtkObject o)`
- `vtkXMLDataElement = obj.GetRootElement ()` - Get the root element from the XML document.
- `obj.SetCompressor (vtkDataCompressor )` - Get/Set the compressor used to decompress binary and appended data after reading from the file.
- `vtkDataCompressor = obj.GetCompressor ()` - Get/Set the compressor used to decompress binary and appended data after reading from the file.
- `long = obj.GetWordTypeSize (int wordType)` - Get the size of a word of the given type.
- `int = obj.Parse ()` - Parse the XML input and check that the file is safe to read. Returns 1 for okay, 0 for error.
- `int = obj.GetAbort ()` - Get/Set flag to abort reading of data. This may be set by a progress event observer.
- `obj.SetAbort (int )` - Get/Set flag to abort reading of data. This may be set by a progress event observer.
- `float = obj.GetProgress ()` - Get/Set progress of reading data. This may be checked by a progress event observer.
- `obj.SetProgress (float )` - Get/Set progress of reading data. This may be checked by a progress event observer.
- `obj.SetAttributesEncoding (int )` - Get/Set the character encoding that will be used to set the attributes's encoding type of each vtkXMLDataElement created by this parser (i.e., the data element attributes will use that encoding internally). If set to VTK_ENCODING_NONE (default), the attribute encoding type will not be changed and will default to the vtkXMLDataElement default encoding type (see vtkXMLDataElement::AttributeEncoding).
- `int = obj.GetAttributesEncodingMinValue ()` - Get/Set the character encoding that will be used to set the attributes's encoding type of each vtkXMLDataElement created by this parser (i.e., the data element attributes will use that encoding internally). If set to VTK_ENCODING_NONE (default), the attribute encoding type will not be changed and will default to the vtkXMLDataElement default encoding type (see vtkXMLDataElement::AttributeEncoding).
- `int = obj.GetAttributesEncodingMaxValue ()` - Get/Set the character encoding that will be used to set the attributes's encoding type of each vtkXMLDataElement created by this parser (i.e., the data element attributes will use that encoding internally). If set to VTK_ENCODING_NONE (default), the attribute encoding type will not be changed and will default to the vtkXMLDataElement default encoding type (see vtkXMLDataElement::AttributeEncoding).

- `int = obj.GetAttributesEncoding ()` - Get/Set the character encoding that will be used to set the attributes's encoding type of each vtkXMLDataElement created by this parser (i.e., the data element attributes will use that encoding internally). If set to VTK_ENCODING_NONE (default), the attribute encoding type will not be changed and will default to the vtkXMLDataElement default encoding type (see vtkXMLDataElement::AttributeEncoding).

- `obj.CharacterDataHandler (string data, int length)` - If you need the text inside XMLElements, turn IgnoreCharacterData off. This method will then be called when the file is parsed, and the text will be stored in each XMLDataElement. VTK XML Readers store the information elsewhere, so the default is to ignore it.

37.118 vtkXMLDataReader

37.118.1 Usage

vtkXMLDataReader provides functionality common to all VTK XML file readers. Concrete subclasses call upon this functionality when needed.

To create an instance of class vtkXMLDataReader, simply invoke its constructor as follows

```
obj = vtkXMLDataReader
```

37.118.2 Methods

The class vtkXMLDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLDataReader class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkXMLDataReader = obj.NewInstance ()`

- `vtkXMLDataReader = obj.SafeDownCast (vtkObject o)`

- `vtkIdType = obj.GetNumberOfPoints ()` - Get the number of points in the output.

- `vtkIdType = obj.GetNumberOfCells ()` - Get the number of cells in the output.

- `obj.CopyOutputInformation (vtkInformation outInfo, int port)`

37.119 vtkXMLDataSetWriter

37.119.1 Usage

vtkXMLDataSetWriter is a wrapper around the VTK XML file format writers. Given an input vtkDataSet, the correct writer is automatically selected based on the type of input.

To create an instance of class vtkXMLDataSetWriter, simply invoke its constructor as follows

```
obj = vtkXMLDataSetWriter
```

37.119.2 Methods

The class vtkXMLDataSetWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLDataSetWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLDataSetWriter = obj.NewInstance ()`
- `vtkXMLDataSetWriter = obj.SafeDownCast (vtkObject o)`

37.120 vtkXMLFileReadTester

37.120.1 Usage

vtkXMLFileReadTester reads the smallest part of a file necessary to determine whether it is a VTK XML file. If so, it extracts the file type and version number.

To create an instance of class vtkXMLFileReadTester, simply invoke its constructor as follows

```
obj = vtkXMLFileReadTester
```

37.120.2 Methods

The class vtkXMLFileReadTester has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLFileReadTester class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLFileReadTester = obj.NewInstance ()`
- `vtkXMLFileReadTester = obj.SafeDownCast (vtkObject o)`
- `int = obj.TestReadFile ()` - Try to read the file given by FileName. Returns 1 if the file is a VTK XML file, and 0 otherwise.
- `obj.SetFileName (string )` - Get/Set the name of the file tested by TestReadFile().
- `string = obj.GetFileName ()` - Get/Set the name of the file tested by TestReadFile().
- `string = obj.GetFileDataType ()` - Get the data type of the XML file tested. If the file could not be read, returns NULL.
- `string = obj.GetFileVersion ()` - Get the file version of the XML file tested. If the file could not be read, returns NULL.

37.121 vtkXMLHierarchicalBoxDataReader

37.121.1 Usage

vtkXMLHierarchicalBoxDataReader reads the VTK XML hierarchical data file format. XML hierarchical data files are meta-files that point to a list of serial VTK XML files. When reading in parallel, it will distribute sub-blocks among processor. If the number of sub-blocks is less than the number of processors, some processors will not have any sub-blocks for that level. If the number of sub-blocks is larger than the number of processors, each processor will possibly have more than 1 sub-block.

To create an instance of class vtkXMLHierarchicalBoxDataReader, simply invoke its constructor as follows

```
obj = vtkXMLHierarchicalBoxDataReader
```

37.121.2 Methods

The class vtkXMLHierarchicalBoxDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLHierarchicalBoxDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLHierarchicalBoxDataReader = obj.NewInstance ()`
- `vtkXMLHierarchicalBoxDataReader = obj.SafeDownCast (vtkObject o)`

37.122 vtkXMLHierarchicalBoxDataWriter

37.122.1 Usage

vtkXMLHierarchicalBoxDataWriter is a vtkXMLCompositeDataWriter subclass to handle vtkHierarchicalBoxDataSet.

To create an instance of class vtkXMLHierarchicalBoxDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLHierarchicalBoxDataWriter
```

37.122.2 Methods

The class vtkXMLHierarchicalBoxDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLHierarchicalBoxDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLHierarchicalBoxDataWriter = obj.NewInstance ()`
- `vtkXMLHierarchicalBoxDataWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()`

37.123 vtkXMLHierarchicalDataReader

37.123.1 Usage

vtkXMLHierarchicalDataReader reads the VTK XML hierarchical data file format. XML hierarchical data files are meta-files that point to a list of serial VTK XML files. When reading in parallel, it will distribute sub-blocks among processor. If the number of sub-blocks is less than the number of processors, some processors will not have any sub-blocks for that level. If the number of sub-blocks is larger than the number of processors, each processor will possibly have more than 1 sub-block.

To create an instance of class vtkXMLHierarchicalDataReader, simply invoke its constructor as follows

```
obj = vtkXMLHierarchicalDataReader
```

37.123.2 Methods

The class vtkXMLHierarchicalDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLHierarchicalDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLHierarchicalDataReader = obj.NewInstance ()`
- `vtkXMLHierarchicalDataReader = obj.SafeDownCast (vtkObject o)`

37.124 vtkXMLHyperOctreeReader

37.124.1 Usage

vtkXMLHyperOctreeReader reads the VTK XML HyperOctree file format. One rectilinear grid file can be read to produce one output. Streaming is supported. The standard extension for this reader's file format is "vto". This reader is also used to read a single piece of the parallel file format.

To create an instance of class vtkXMLHyperOctreeReader, simply invoke its constructor as follows

```
obj = vtkXMLHyperOctreeReader
```

37.124.2 Methods

The class vtkXMLHyperOctreeReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLHyperOctreeReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLHyperOctreeReader = obj.NewInstance ()`
- `vtkXMLHyperOctreeReader = obj.SafeDownCast (vtkObject o)`
- `vtkHyperOctree = obj.GetOutput ()` - Get the reader's output.
- `vtkHyperOctree = obj.GetOutput (int idx)` - Get the reader's output.

37.125 vtkXMLHyperOctreeWriter

37.125.1 Usage

vtkXMLHyperOctreeWriter writes the VTK XML HyperOctree file format. One HyperOctree input can be written into one file in any number of streamed pieces. The standard extension for this writer's file format is "vto". This writer is also used to write a single piece of the parallel file format.

To create an instance of class vtkXMLHyperOctreeWriter, simply invoke its constructor as follows

```
obj = vtkXMLHyperOctreeWriter
```

37.125.2 Methods

The class vtkXMLHyperOctreeWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLHyperOctreeWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLHyperOctreeWriter = obj.NewInstance ()`
- `vtkXMLHyperOctreeWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.126 vtkXMLImageDataReader

37.126.1 Usage

vtkXMLImageDataReader reads the VTK XML ImageData file format. One image data file can be read to produce one output. Streaming is supported. The standard extension for this reader's file format is "vti". This reader is also used to read a single piece of the parallel file format.

To create an instance of class vtkXMLImageDataReader, simply invoke its constructor as follows

```
obj = vtkXMLImageDataReader
```

37.126.2 Methods

The class vtkXMLImageDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLImageDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLImageDataReader = obj.NewInstance ()`
- `vtkXMLImageDataReader = obj.SafeDownCast (vtkObject o)`
- `vtkImageData = obj.GetOutput ()` - Get the reader's output.
- `vtkImageData = obj.GetOutput (int idx)` - Get the reader's output.
- `obj.CopyOutputInformation (vtkInformation outInfo, int port)` - For the specified port, copy the information this reader sets up in SetupOutputInformation to outInfo

37.127 vtkXMLImageDataWriter

37.127.1 Usage

vtkXMLImageDataWriter writes the VTK XML ImageData file format. One image data input can be written into one file in any number of streamed pieces. The standard extension for this writer's file format is "vti". This writer is also used to write a single piece of the parallel file format.

To create an instance of class vtkXMLImageDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLImageDataWriter
```

37.127.2 Methods

The class vtkXMLImageDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLImageDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLImageDataWriter = obj.NewInstance ()`
- `vtkXMLImageDataWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.128 vtkXMLMaterial

37.128.1 Usage

vtkXMLMaterial encapsulates VTK Material description. It keeps a pointer to vtkXMLDataElement that defines the material and provides access to Shaders/Properties defined in it. .SECTION Thanks Shader support in VTK includes key contributions by Gary Templet at Sandia National Labs.

To create an instance of class vtkXMLMaterial, simply invoke its constructor as follows

```
obj = vtkXMLMaterial
```

37.128.2 Methods

The class vtkXMLMaterial has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLMaterial class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLMaterial = obj.NewInstance ()`
- `vtkXMLMaterial = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfProperties ()` - Get number of elements of type Property.
- `int = obj.GetNumberOfTextures ()` - Get number of elements of type Texture.
- `int = obj.GetNumberOfVertexShaders ()` - Get number of Vertex shaders.

- `int = obj.GetNumberOfFragmentShaders ()` - Get number of fragment shaders.
- `vtkXMLDataElement = obj.GetProperty (int id)` - Get the ith vtkXMLDataElement of type ¡Property /¿.
- `vtkXMLDataElement = obj.GetTexture (int id)` - Get the ith vtkXMLDataElement of type ¡Texture /¿.
- `vtkXMLShader = obj.GetVertexShader (int id)` - Get the ith vtkXMLDataElement of type ¡VertexShader /¿.
- `vtkXMLShader = obj.GetFragmentShader (int id)` - Get the ith vtkXMLDataElement of type ¡FragmentShader /¿.
- `vtkXMLDataElement = obj.GetRootElement ()` - Get/Set the XML root element that describes this material.
- `obj.SetRootElement (vtkXMLDataElement )` - Get/Set the XML root element that describes this material.
- `int = obj.GetShaderLanguage ()` - Get the Language used by the shaders in this Material. The Language of a vtkXMLMaterial is based on the Language of it's shaders.
- `int = obj.GetShaderStyle ()` - Get the style the shaders.

37.129 vtkXMLMaterialParser

37.129.1 Usage

vtkXMLMaterialParser parses a VTK Material file and provides that file's description of a number of vertex and fragment shaders along with data values specified for data members of vtkProperty. This material is to be applied to an actor through it's vtkProperty and augments VTK's concept of a vtkProperty to include explicitly include vertex and fragment shaders and parameter settings for those shaders. This effectively makes reflectance models and other shaders a material property. If no shaders are specified VTK should default to standard rendering.

.SECTION Design vtkXMLMaterialParser provides access to 3 distinct types of first-level vtkXMLDataElements that describe a VTK material. These elements are as follows:

vtkProperty - describe values for vtkProperty data members

vtkVertexShader - a vertex shader and enough information to install it into the hardware rendering pipeline including values for specific shader parameters and structures.

vtkFragmentShader - a fragment shader and enough information to install it into the hardware rendering pipeline including values for specific shader parameters and structures.

The design of the material file closely follows that of vtk's xml descriptions of it's data sets. This allows use of the very handy vtkXMLDataElement which provides easy access to an xml element's attribute values. Inlined data is currently not handled.

Ideally this class would be a Facade to a DOM parser, but VTK only provides access to expat, a SAX parser. Other vtk classes that parse xml files are tuned to read vtkDataSets and don't provide the functionality to handle generic xml data. As such they are of little use here.

This class may be extended for better data handling or may become a Facade to a DOM parser should on become part of the VTK code base. .SECTION Thanks Shader support in VTK includes key contributions by Gary Templet at Sandia National Labs.

To create an instance of class vtkXMLMaterialParser, simply invoke its constructor as follows

```
obj = vtkXMLMaterialParser
```

37.129.2 Methods

The class vtkXMLMaterialParser has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLMaterialParser class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLMaterialParser = obj.NewInstance ()`
- `vtkXMLMaterialParser = obj.SafeDownCast (vtkObject o)`
- `vtkXMLMaterial = obj.GetMaterial ()` - Set/Get the vtkXMLMaterial representation of the parsed material.
- `obj.SetMaterial (vtkXMLMaterial )` - Set/Get the vtkXMLMaterial representation of the parsed material.
- `int = obj.Parse ()` - Overridden to initialize the internal structures before the parsing begins.
- `int = obj.Parse (string inputString)` - Overridden to initialize the internal structures before the parsing begins.
- `int = obj.Parse (string inputString, int length)` - Overridden to initialize the internal structures before the parsing begins.
- `int = obj.InitializeParser ()` - Overridden to clean up internal structures before the chunk-parsing begins.

37.130 vtkXMLMaterialReader

37.130.1 Usage

vtkXMLMaterialReader provides access to three types of vtkXMLDataElement found in XML Material Files. This class sorts them by type and integer id from 0-N for N elements of a specific type starting with the first instance found.

.SECTION Design This class is basically a Facade for vtkXMLMaterialParser. Currently functionality is to only provide access to vtkXMLDataElements but further extensions may return higher level data structures.

Having both an vtkXMLMaterialParser and a vtkXMLMaterialReader is consistent with VTK's design for handling xml file and provides for future flexibility, that is better data handlers and interfacing with a DOM xml parser.

vtkProperty - defines values for some or all data members of vtkProperty

vtkVertexShader - defines vertex shaders

vtkFragmentShader - defines fragment shaders .SECTION Thanks Shader support in VTK includes key contributions by Gary Templet at Sandia National Labs.

To create an instance of class vtkXMLMaterialReader, simply invoke its constructor as follows

```
obj = vtkXMLMaterialReader
```

37.130.2 Methods

The class vtkXMLMaterialReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLMaterialReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLMaterialReader = obj.NewInstance ()`
- `vtkXMLMaterialReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Set and get file name.
- `string = obj.GetFileName ()` - Set and get file name.
- `obj.ReadMaterial ()` - Read the material file refered to in FileName. If the Reader hasn't changed since the last ReadMaterial(), it does not read the file again.
- `vtkXMLMaterial = obj.GetMaterial ()` - Get the Material representation read by the reader.

37.131 vtkXMLMultiBlockDataReader

37.131.1 Usage

vtkXMLMultiBlockDataReader reads the VTK XML multi-block data file format. XML multi-block data files are meta-files that point to a list of serial VTK XML files. When reading in parallel, it will distribute sub-blocks among processor. If the number of sub-blocks is less than the number of processors, some processors will not have any sub-blocks for that block. If the number of sub-blocks is larger than the number of processors, each processor will possibly have more than 1 sub-block.

To create an instance of class vtkXMLMultiBlockDataReader, simply invoke its constructor as follows

```
obj = vtkXMLMultiBlockDataReader
```

37.131.2 Methods

The class vtkXMLMultiBlockDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLMultiBlockDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLMultiBlockDataReader = obj.NewInstance ()`
- `vtkXMLMultiBlockDataReader = obj.SafeDownCast (vtkObject o)`

37.132 vtkXMLMultiBlockDataWriter

37.132.1 Usage

vtkXMLMultiBlockDataWriter is a vtkXMLCompositeDataWriter subclass to handle vtkMultiBlockDataSet.

To create an instance of class vtkXMLMultiBlockDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLMultiBlockDataWriter
```

37.132.2 Methods

The class vtkXMLMultiBlockDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLMultiBlockDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLMultiBlockDataWriter = obj.NewInstance ()`
- `vtkXMLMultiBlockDataWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()`

37.133 vtkXMLMultiGroupDataReader

37.133.1 Usage

vtkXMLMultiGroupDataReader is a legacy reader that reads multi group files into multiblock datasets.

To create an instance of class vtkXMLMultiGroupDataReader, simply invoke its constructor as follows

```
obj = vtkXMLMultiGroupDataReader
```

37.133.2 Methods

The class vtkXMLMultiGroupDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLMultiGroupDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLMultiGroupDataReader = obj.NewInstance ()`
- `vtkXMLMultiGroupDataReader = obj.SafeDownCast (vtkObject o)`

37.134 vtkXMLParser

37.134.1 Usage

vtkXMLParser reads a stream and parses XML element tags and corresponding attributes. Each element begin tag and its attributes are sent to the StartElement method. Each element end tag is sent to the EndElement method. Subclasses should replace these methods to actually use the tags. .SECTION ToDo: Add commands for parsing in Tcl.

To create an instance of class vtkXMLParser, simply invoke its constructor as follows

```
obj = vtkXMLParser
```

37.134.2 Methods

The class vtkXMLParser has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLParser class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLParser = obj.NewInstance ()`
- `vtkXMLParser = obj.SafeDownCast (vtkObject o)`
- `long = obj.TellG ()` - Used by subclasses and their supporting classes. These methods wrap around the tellg and seekg methods of the input stream to work-around stream bugs on various platforms.
- `obj.SeekG (long position)` - Used by subclasses and their supporting classes. These methods wrap around the tellg and seekg methods of the input stream to work-around stream bugs on various platforms.
- `int = obj.Parse ()` - Parse the XML input.
- `int = obj.Parse (string inputString)` - Parse the XML message. If length is specified, parse only the first "length" characters
- `int = obj.Parse (string inputString, int length)` - Parse the XML message. If length is specified, parse only the first "length" characters
- `int = obj.InitializeParser ()` - When parsing fragments of XML or streaming XML, use the following three methods. InitializeParser method initialize parser but does not perform any actual parsing. ParseChunk parses framgent of XML. This has to match to what was already parsed. CleanupParser finishes parsing. If there were errors, CleanupParser will report them.
- `int = obj.ParseChunk (string inputString, int length)` - When parsing fragments of XML or streaming XML, use the following three methods. InitializeParser method initialize parser but does not perform any actual parsing. ParseChunk parses framgent of XML. This has to match to what was already parsed. CleanupParser finishes parsing. If there were errors, CleanupParser will report them.
- `int = obj.CleanupParser ()` - When parsing fragments of XML or streaming XML, use the following three methods. InitializeParser method initialize parser but does not perform any actual parsing. ParseChunk parses framgent of XML. This has to match to what was already parsed. CleanupParser finishes parsing. If there were errors, CleanupParser will report them.
- `obj.SetFileName (string )` - Set and get file name.
- `string = obj.GetFileName ()` - Set and get file name.
- `obj.SetIgnoreCharacterData (int )` - If this is off (the default), CharacterDataHandler will be called to process text within XML Elements. If this is on, the text will be ignored.
- `int = obj.GetIgnoreCharacterData ()` - If this is off (the default), CharacterDataHandler will be called to process text within XML Elements. If this is on, the text will be ignored.
- `obj.SetEncoding (string )` - Set and get the encoding the parser should expect (NULL defaults to Expat's own default encoder, i.e UTF-8). This should be set before parsing (i.e. a call to Parse()) or even initializing the parser (i.e. a call to InitializeParser())
- `string = obj.GetEncoding ()` - Set and get the encoding the parser should expect (NULL defaults to Expat's own default encoder, i.e UTF-8). This should be set before parsing (i.e. a call to Parse()) or even initializing the parser (i.e. a call to InitializeParser())

37.135 vtkXMLPDataReader

37.135.1 Usage

vtkXMLPDataReader provides functionality common to all PVTK XML file readers. Concrete subclasses call upon this functionality when needed.

To create an instance of class vtkXMLPDataReader, simply invoke its constructor as follows

```
obj = vtkXMLPDataReader
```

37.135.2 Methods

The class vtkXMLPDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPDataReader = obj.NewInstance ()`
- `vtkXMLPDataReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfPieces ()` - Get the number of pieces from the summary file being read.
- `obj.CopyOutputInformation (vtkInformation outInfo, int port)`

37.136 vtkXMLPDataSetWriter

37.136.1 Usage

vtkXMLPDataSetWriter is a wrapper around the PVTK XML file format writers. Given an input vtkDataSet, the correct writer is automatically selected based on the type of input.

To create an instance of class vtkXMLPDataSetWriter, simply invoke its constructor as follows

```
obj = vtkXMLPDataSetWriter
```

37.136.2 Methods

The class vtkXMLPDataSetWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPDataSetWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPDataSetWriter = obj.NewInstance ()`
- `vtkXMLPDataSetWriter = obj.SafeDownCast (vtkObject o)`

37.137 vtkXMLPDataWriter

37.137.1 Usage

vtkXMLPDataWriter is the superclass for all XML parallel data set writers. It provides functionality needed for writing parallel formats, such as the selection of which writer writes the summary file and what range of pieces are assigned to each serial writer.

To create an instance of class vtkXMLPDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLPDataWriter
```

37.137.2 Methods

The class vtkXMLPDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPDataWriter = obj.NewInstance ()`
- `vtkXMLPDataWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfPieces (int )` - Get/Set the number of pieces that are being written in parallel.
- `int = obj.GetNumberOfPieces ()` - Get/Set the number of pieces that are being written in parallel.
- `obj.SetStartPiece (int )` - Get/Set the range of pieces assigned to this writer.
- `int = obj.GetStartPiece ()` - Get/Set the range of pieces assigned to this writer.
- `obj.SetEndPiece (int )` - Get/Set the range of pieces assigned to this writer.
- `int = obj.GetEndPiece ()` - Get/Set the range of pieces assigned to this writer.
- `obj.SetGhostLevel (int )` - Get/Set the ghost level used for this writer's piece.
- `int = obj.GetGhostLevel ()` - Get/Set the ghost level used for this writer's piece.
- `obj.SetWriteSummaryFile (int flag)` - Get/Set whether this instance of the writer should write the summary file that refers to all of the pieces' individual files. Default is yes only for piece 0 writer.
- `int = obj.GetWriteSummaryFile ()` - Get/Set whether this instance of the writer should write the summary file that refers to all of the pieces' individual files. Default is yes only for piece 0 writer.
- `obj.WriteSummaryFileOn ()` - Get/Set whether this instance of the writer should write the summary file that refers to all of the pieces' individual files. Default is yes only for piece 0 writer.
- `obj.WriteSummaryFileOff ()` - Get/Set whether this instance of the writer should write the summary file that refers to all of the pieces' individual files. Default is yes only for piece 0 writer.

37.138 vtkXMLPImageDataReader

37.138.1 Usage

vtkXMLPImageDataReader reads the PVTK XML ImageData file format. This reads the parallel format's summary file and then uses vtkXMLImageDataReader to read data from the individual ImageData piece files. Streaming is supported. The standard extension for this reader's file format is "pvti".

To create an instance of class vtkXMLPImageDataReader, simply invoke its constructor as follows

```
obj = vtkXMLPImageDataReader
```

37.138.2 Methods

The class vtkXMLPImageDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPImageDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPImageDataReader = obj.NewInstance ()`
- `vtkXMLPImageDataReader = obj.SafeDownCast (vtkObject o)`
- `vtkImageData = obj.GetOutput ()` - Get the reader's output.
- `vtkImageData = obj.GetOutput (int idx)` - Get the reader's output.
- `obj.CopyOutputInformation (vtkInformation outInfo, int port)`

37.139 vtkXMLPImageDataWriter

37.139.1 Usage

vtkXMLPImageDataWriter writes the PVTK XML ImageData file format. One image data input can be written into a parallel file format with any number of pieces spread across files. The standard extension for this writer's file format is "pvti". This writer uses vtkXMLImageDataWriter to write the individual piece files.

To create an instance of class vtkXMLPImageDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLPImageDataWriter
```

37.139.2 Methods

The class vtkXMLPImageDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPImageDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPImageDataWriter = obj.NewInstance ()`
- `vtkXMLPImageDataWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.140 vtkXMLPolyDataReader

37.140.1 Usage

vtkXMLPolyDataReader reads the VTK XML PolyData file format. One polygonal data file can be read to produce one output. Streaming is supported. The standard extension for this reader's file format is "vtp". This reader is also used to read a single piece of the parallel file format.

To create an instance of class vtkXMLPolyDataReader, simply invoke its constructor as follows

```
obj = vtkXMLPolyDataReader
```

37.140.2 Methods

The class vtkXMLPolyDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPolyDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPolyDataReader = obj.NewInstance ()`
- `vtkXMLPolyDataReader = obj.SafeDownCast (vtkObject o)`
- `vtkPolyData = obj.GetOutput ()` - Get the reader's output.
- `vtkPolyData = obj.GetOutput (int idx)` - Get the reader's output.
- `vtkIdType = obj.GetNumberOfVerts ()` - Get the number of verts/lines/strips/polys in the output.
- `vtkIdType = obj.GetNumberOfLines ()` - Get the number of verts/lines/strips/polys in the output.
- `vtkIdType = obj.GetNumberOfStrips ()` - Get the number of verts/lines/strips/polys in the output.
- `vtkIdType = obj.GetNumberOfPolys ()` - Get the number of verts/lines/strips/polys in the output.

37.141 vtkXMLPolyDataWriter

37.141.1 Usage

vtkXMLPolyDataWriter writes the VTK XML PolyData file format. One polygonal data input can be written into one file in any number of streamed pieces (if supported by the rest of the pipeline). The standard extension for this writer's file format is "vtp". This writer is also used to write a single piece of the parallel file format.

To create an instance of class vtkXMLPolyDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLPolyDataWriter
```

37.141.2 Methods

The class vtkXMLPolyDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPolyDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPolyDataWriter = obj.NewInstance ()`
- `vtkXMLPolyDataWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.142 vtkXMLPPolyDataReader

37.142.1 Usage

vtkXMLPPolyDataReader reads the PVTK XML PolyData file format. This reads the parallel format's summary file and then uses vtkXMLPolyDataReader to read data from the individual PolyData piece files. Streaming is supported. The standard extension for this reader's file format is "pvtp".

To create an instance of class vtkXMLPPolyDataReader, simply invoke its constructor as follows

```
obj = vtkXMLPPolyDataReader
```

37.142.2 Methods

The class vtkXMLPPolyDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPPolyDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPPolyDataReader = obj.NewInstance ()`
- `vtkXMLPPolyDataReader = obj.SafeDownCast (vtkObject o)`
- `vtkPolyData = obj.GetOutput ()` - Get the reader's output.
- `vtkPolyData = obj.GetOutput (int idx)` - Get the reader's output.

37.143 vtkXMLPPolyDataWriter

37.143.1 Usage

vtkXMLPPolyDataWriter writes the PVTK XML PolyData file format. One poly data input can be written into a parallel file format with any number of pieces spread across files. The standard extension for this writer's file format is "pvtp". This writer uses vtkXMLPolyDataWriter to write the individual piece files.

To create an instance of class vtkXMLPPolyDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLPPolyDataWriter
```

37.143.2 Methods

The class vtkXMLPPolyDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPPolyDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPPolyDataWriter = obj.NewInstance ()`
- `vtkXMLPPolyDataWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.144 vtkXMLPRectilinearGridReader

37.144.1 Usage

vtkXMLPRectilinearGridReader reads the PVTK XML RectilinearGrid file format. This reads the parallel format's summary file and then uses vtkXMLRectilinearGridReader to read data from the individual RectilinearGrid piece files. Streaming is supported. The standard extension for this reader's file format is "pvtr".

To create an instance of class vtkXMLPRectilinearGridReader, simply invoke its constructor as follows

```
obj = vtkXMLPRectilinearGridReader
```

37.144.2 Methods

The class vtkXMLPRectilinearGridReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPRectilinearGridReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPRectilinearGridReader = obj.NewInstance ()`
- `vtkXMLPRectilinearGridReader = obj.SafeDownCast (vtkObject o)`
- `vtkRectilinearGrid = obj.GetOutput ()` - Get the reader's output.
- `vtkRectilinearGrid = obj.GetOutput (int idx)` - Get the reader's output.

37.145 vtkXMLPRectilinearGridWriter

37.145.1 Usage

vtkXMLPRectilinearGridWriter writes the PVTK XML RectilinearGrid file format. One rectilinear grid input can be written into a parallel file format with any number of pieces spread across files. The standard extension for this writer's file format is "pvtr". This writer uses vtkXMLRectilinearGridWriter to write the individual piece files.

To create an instance of class vtkXMLPRectilinearGridWriter, simply invoke its constructor as follows

```
obj = vtkXMLPRectilinearGridWriter
```

37.145.2 Methods

The class vtkXMLPRectilinearGridWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPRectilinearGridWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPRectilinearGridWriter = obj.NewInstance ()`
- `vtkXMLPRectilinearGridWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.146 vtkXMLPStructuredDataReader

37.146.1 Usage

vtkXMLPStructuredDataReader provides functionality common to all parallel structured data format readers.

To create an instance of class vtkXMLPStructuredDataReader, simply invoke its constructor as follows

```
obj = vtkXMLPStructuredDataReader
```

37.146.2 Methods

The class vtkXMLPStructuredDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPStructuredDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPStructuredDataReader = obj.NewInstance ()`
- `vtkXMLPStructuredDataReader = obj.SafeDownCast (vtkObject o)`
- `vtkExtentTranslator = obj.GetExtentTranslator ()` - Get an extent translator that will create pieces matching the input file's piece breakdown. This can be used further down the pipeline to prevent reading from outside this process's piece. The translator is only valid after an UpdateInformation has been called.
- `obj.CopyOutputInformation (vtkInformation outInfo, int port)`

37.147 vtkXMLPStructuredDataWriter

37.147.1 Usage

vtkXMLPStructuredDataWriter provides PVTK XML writing functionality that is common among all the parallel structured data formats.

To create an instance of class vtkXMLPStructuredDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLPStructuredDataWriter
```

37.147.2 Methods

The class vtkXMLPStructuredDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPStructuredDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPStructuredDataWriter = obj.NewInstance ()`
- `vtkXMLPStructuredDataWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetExtentTranslator (vtkExtentTranslator )` - Get/Set the extent translator used for creating pieces.
- `vtkExtentTranslator = obj.GetExtentTranslator ()` - Get/Set the extent translator used for creating pieces.

37.148 vtkXMLPStructuredGridReader

37.148.1 Usage

vtkXMLPStructuredGridReader reads the PVTK XML StructuredGrid file format. This reads the parallel format's summary file and then uses vtkXMLStructuredGridReader to read data from the individual StructuredGrid piece files. Streaming is supported. The standard extension for this reader's file format is "pvts".

To create an instance of class vtkXMLPStructuredGridReader, simply invoke its constructor as follows

```
obj = vtkXMLPStructuredGridReader
```

37.148.2 Methods

The class vtkXMLPStructuredGridReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPStructuredGridReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPStructuredGridReader = obj.NewInstance ()`
- `vtkXMLPStructuredGridReader = obj.SafeDownCast (vtkObject o)`
- `vtkStructuredGrid = obj.GetOutput ()` - Get the reader's output.
- `vtkStructuredGrid = obj.GetOutput (int idx)` - Needed for ParaView

37.149 vtkXMLPStructuredGridWriter

37.149.1 Usage

vtkXMLPStructuredGridWriter writes the PVTK XML StructuredGrid file format. One structured grid input can be written into a parallel file format with any number of pieces spread across files. The standard extension for this writer's file format is "pvts". This writer uses vtkXMLStructuredGridWriter to write the individual piece files.

To create an instance of class vtkXMLPStructuredGridWriter, simply invoke its constructor as follows

```
obj = vtkXMLPStructuredGridWriter
```

37.149.2 Methods

The class vtkXMLPStructuredGridWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPStructuredGridWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPStructuredGridWriter = obj.NewInstance ()`
- `vtkXMLPStructuredGridWriter = obj.SafeDownCast (vtkObject o)`

37.150 vtkXMLPUnstructuredDataReader

37.150.1 Usage

vtkXMLPUnstructuredDataReader provides functionality common to all parallel unstructured data format readers.

To create an instance of class vtkXMLPUnstructuredDataReader, simply invoke its constructor as follows

```
obj = vtkXMLPUnstructuredDataReader
```

37.150.2 Methods

The class vtkXMLPUnstructuredDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPUnstructuredDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPUnstructuredDataReader = obj.NewInstance ()`
- `vtkXMLPUnstructuredDataReader = obj.SafeDownCast (vtkObject o)`
- `obj.CopyOutputInformation (vtkInformation outInfo, int port)`

37.151 vtkXMLPUnstructuredDataWriter

37.151.1 Usage

vtkXMLPUnstructuredDataWriter provides PVTK XML writing functionality that is common among all the parallel unstructured data formats.

To create an instance of class vtkXMLPUnstructuredDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLPUnstructuredDataWriter
```

37.151.2 Methods

The class vtkXMLPUnstructuredDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPUnstructuredDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPUnstructuredDataWriter = obj.NewInstance ()`
- `vtkXMLPUnstructuredDataWriter = obj.SafeDownCast (vtkObject o)`

37.152 vtkXMLPUnstructuredGridReader

37.152.1 Usage

vtkXMLPUnstructuredGridReader reads the PVTK XML UnstructuredGrid file format. This reads the parallel format's summary file and then uses vtkXMLUnstructuredGridReader to read data from the individual UnstructuredGrid piece files. Streaming is supported. The standard extension for this reader's file format is "pvtu".

To create an instance of class vtkXMLPUnstructuredGridReader, simply invoke its constructor as follows

```
obj = vtkXMLPUnstructuredGridReader
```

37.152.2 Methods

The class vtkXMLPUnstructuredGridReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPUnstructuredGridReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPUnstructuredGridReader = obj.NewInstance ()`
- `vtkXMLPUnstructuredGridReader = obj.SafeDownCast (vtkObject o)`
- `vtkUnstructuredGrid = obj.GetOutput ()` - Get the reader's output.
- `vtkUnstructuredGrid = obj.GetOutput (int idx)` - Get the reader's output.

37.153 vtkXMLPUnstructuredGridWriter

37.153.1 Usage

vtkXMLPUnstructuredGridWriter writes the PVTK XML UnstructuredGrid file format. One unstructured grid input can be written into a parallel file format with any number of pieces spread across files. The standard extension for this writer's file format is "pvtu". This writer uses vtkXMLUnstructuredGridWriter to write the individual piece files.

To create an instance of class vtkXMLPUnstructuredGridWriter, simply invoke its constructor as follows

```
obj = vtkXMLPUnstructuredGridWriter
```

37.153.2 Methods

The class vtkXMLPUnstructuredGridWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPUnstructuredGridWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPUnstructuredGridWriter = obj.NewInstance ()`
- `vtkXMLPUnstructuredGridWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.154 vtkXMLReader

37.154.1 Usage

vtkXMLReader uses vtkXMLDataParser to parse a VTK XML input file. Concrete subclasses then traverse the parsed file structure and extract data.

To create an instance of class vtkXMLReader, simply invoke its constructor as follows

```
obj = vtkXMLReader
```

37.154.2 Methods

The class vtkXMLReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLReader = obj.NewInstance ()`
- `vtkXMLReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Get/Set the name of the input file.
- `string = obj.GetFileName ()` - Get/Set the name of the input file.
- `int = obj.CanReadFile (string name)` - Test whether the file with the given name can be read by this reader.
- `vtkDataSet = obj.GetOutputAsDataSet ()` - Get the output as a vtkDataSet pointer.
- `vtkDataSet = obj.GetOutputAsDataSet (int index)` - Get the output as a vtkDataSet pointer.
- `vtkDataArraySelection = obj.GetPointDataArraySelection ()` - Get the data array selection tables used to configure which data arrays are loaded by the reader.
- `vtkDataArraySelection = obj.GetCellDataArraySelection ()` - Get the data array selection tables used to configure which data arrays are loaded by the reader.
- `int = obj.GetNumberOfPointArrays ()` - Get the number of point or cell arrays available in the input.
- `int = obj.GetNumberOfCellArrays ()` - Get the number of point or cell arrays available in the input.
- `string = obj.GetPointArrayName (int index)` - Get the name of the point or cell array with the given index in the input.
- `string = obj.GetCellArrayName (int index)` - Get the name of the point or cell array with the given index in the input.
- `int = obj.GetPointArrayStatus (string name)` - Get/Set whether the point or cell array with the given name is to be read.
- `int = obj.GetCellArrayStatus (string name)` - Get/Set whether the point or cell array with the given name is to be read.
- `obj.SetPointArrayStatus (string name, int status)` - Get/Set whether the point or cell array with the given name is to be read.

- `obj.SetCellArrayStatus (string name, int status)` - Get/Set whether the point or cell array with the given name is to be read.
- `obj.CopyOutputInformation (vtkInformation , int )` - Which TimeStep to read.
- `obj.SetTimeStep (int )` - Which TimeStep to read.
- `int = obj.GetTimeStep ()` - Which TimeStep to read.
- `int = obj.GetNumberOfTimeSteps ()`
- `int = obj. GetTimeStepRange ()` - Which TimeStepRange to read
- `obj.SetTimeStepRange (int , int )` - Which TimeStepRange to read
- `obj.SetTimeStepRange (int  a[2])` - Which TimeStepRange to read

37.155 vtkXMLRectilinearGridReader

37.155.1 Usage

vtkXMLRectilinearGridReader reads the VTK XML RectilinearGrid file format. One rectilinear grid file can be read to produce one output. Streaming is supported. The standard extension for this reader's file format is "vtr". This reader is also used to read a single piece of the parallel file format.

To create an instance of class vtkXMLRectilinearGridReader, simply invoke its constructor as follows

```
obj = vtkXMLRectilinearGridReader
```

37.155.2 Methods

The class vtkXMLRectilinearGridReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLRectilinearGridReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLRectilinearGridReader = obj.NewInstance ()`
- `vtkXMLRectilinearGridReader = obj.SafeDownCast (vtkObject o)`
- `vtkRectilinearGrid = obj.GetOutput ()` - Get the reader's output.
- `vtkRectilinearGrid = obj.GetOutput (int idx)` - Get the reader's output.

37.156 vtkXMLRectilinearGridWriter

37.156.1 Usage

vtkXMLRectilinearGridWriter writes the VTK XML RectilinearGrid file format. One rectilinear grid input can be written into one file in any number of streamed pieces. The standard extension for this writer's file format is "vtr". This writer is also used to write a single piece of the parallel file format.

To create an instance of class vtkXMLRectilinearGridWriter, simply invoke its constructor as follows

```
obj = vtkXMLRectilinearGridWriter
```

37.156.2 Methods

The class vtkXMLRectilinearGridWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLRectilinearGridWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLRectilinearGridWriter = obj.NewInstance ()`
- `vtkXMLRectilinearGridWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.157 vtkXMLShader

37.157.1 Usage

vtkXMLShader encapsulates the XML description for a Shader. It provides convenient access to various attributes/properties of a shader. .SECTION Thanks Shader support in VTK includes key contributions by Gary Templet at Sandia National Labs.

To create an instance of class vtkXMLShader, simply invoke its constructor as follows

```
obj = vtkXMLShader
```

37.157.2 Methods

The class vtkXMLShader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLShader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLShader = obj.NewInstance ()`
- `vtkXMLShader = obj.SafeDownCast (vtkObject o)`
- `vtkXMLDataElement = obj.GetRootElement ()` - Get/Set the XML root element that describes this shader.
- `obj.SetRootElement (vtkXMLDataElement )` - Get/Set the XML root element that describes this shader.
- `int = obj.GetLanguage ()` - Returns the shader's language as defined in the XML description.
- `int = obj.GetScope ()` - Returns the type of the shader as defined in the XML description.
- `int = obj.GetLocation ()` - Returns the location of the shader as defined in the XML description.
- `int = obj.GetStyle ()` - Returns the style of the shader as optionaly defined in the XML description. If not present, default style is 1. "style=2" means it is a shader without a main(). In style 2, the "main" function for the vertex shader part is void propFuncVS(void), the main function for the fragment shader part is void propFuncFS(). This is useful when combining a shader at the actor level and a shader defines at the renderer level, like the depth peeling pass.

- `string = obj.GetName ()` - Get the name of the Shader.
- `string = obj.GetEntry ()` - Get the entry point to the shader code as defined in the XML.
- `string = obj.GetCode ()` - Get the shader code.

37.158 vtkXMLStructuredDataReader

37.158.1 Usage

vtkXMLStructuredDataReader provides functionality common to all structured data format readers.

To create an instance of class vtkXMLStructuredDataReader, simply invoke its constructor as follows

```
obj = vtkXMLStructuredDataReader
```

37.158.2 Methods

The class vtkXMLStructuredDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLStructuredDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLStructuredDataReader = obj.NewInstance ()`
- `vtkXMLStructuredDataReader = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetNumberOfPoints ()` - Get the number of points in the output.
- `vtkIdType = obj.GetNumberOfCells ()` - Get the number of cells in the output.
- `obj.SetWholeSlices (int )` - Get/Set whether the reader gets a whole slice from disk when only a rectangle inside it is needed. This mode reads more data than necessary, but prevents many short reads from interacting poorly with the compression and encoding schemes.
- `int = obj.GetWholeSlices ()` - Get/Set whether the reader gets a whole slice from disk when only a rectangle inside it is needed. This mode reads more data than necessary, but prevents many short reads from interacting poorly with the compression and encoding schemes.
- `obj.WholeSlicesOn ()` - Get/Set whether the reader gets a whole slice from disk when only a rectangle inside it is needed. This mode reads more data than necessary, but prevents many short reads from interacting poorly with the compression and encoding schemes.
- `obj.WholeSlicesOff ()` - Get/Set whether the reader gets a whole slice from disk when only a rectangle inside it is needed. This mode reads more data than necessary, but prevents many short reads from interacting poorly with the compression and encoding schemes.
- `obj.CopyOutputInformation (vtkInformation outInfo, int port)` - For the specified port, copy the information this reader sets up in SetupOutputInformation to outInfo

37.159 vtkXMLStructuredDataWriter

37.159.1 Usage

vtkXMLStructuredDataWriter provides VTK XML writing functionality that is common among all the structured data formats.

To create an instance of class vtkXMLStructuredDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLStructuredDataWriter
```

37.159.2 Methods

The class vtkXMLStructuredDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLStructuredDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLStructuredDataWriter = obj.NewInstance ()`
- `vtkXMLStructuredDataWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfPieces (int )` - Get/Set the number of pieces used to stream the image through the pipeline while writing to the file.
- `int = obj.GetNumberOfPieces ()` - Get/Set the number of pieces used to stream the image through the pipeline while writing to the file.
- `obj.SetWriteExtent (int , int , int , int , int , int )` - Get/Set the extent of the input that should be treated as the WholeExtent in the output file. The default is the WholeExtent of the input.
- `obj.SetWriteExtent (int  a[6])` - Get/Set the extent of the input that should be treated as the WholeExtent in the output file. The default is the WholeExtent of the input.
- `int = obj. GetWriteExtent ()` - Get/Set the extent of the input that should be treated as the WholeExtent in the output file. The default is the WholeExtent of the input.
- `obj.SetExtentTranslator (vtkExtentTranslator )` - Get/Set the extent translator used for streaming.
- `vtkExtentTranslator = obj.GetExtentTranslator ()` - Get/Set the extent translator used for streaming.

37.160 vtkXMLStructuredGridReader

37.160.1 Usage

vtkXMLStructuredGridReader reads the VTK XML StructuredGrid file format. One structured grid file can be read to produce one output. Streaming is supported. The standard extension for this reader's file format is "vts". This reader is also used to read a single piece of the parallel file format.

To create an instance of class vtkXMLStructuredGridReader, simply invoke its constructor as follows

```
obj = vtkXMLStructuredGridReader
```

37.160.2 Methods

The class vtkXMLStructuredGridReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLStructuredGridReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLStructuredGridReader = obj.NewInstance ()`
- `vtkXMLStructuredGridReader = obj.SafeDownCast (vtkObject o)`
- `vtkStructuredGrid = obj.GetOutput ()` - Get the reader's output.
- `vtkStructuredGrid = obj.GetOutput (int idx)` - Get the reader's output.

37.161 vtkXMLStructuredGridWriter

37.161.1 Usage

vtkXMLStructuredGridWriter writes the VTK XML StructuredGrid file format. One structured grid input can be written into one file in any number of streamed pieces. The standard extension for this writer's file format is "vts". This writer is also used to write a single piece of the parallel file format.

To create an instance of class vtkXMLStructuredGridWriter, simply invoke its constructor as follows

```
obj = vtkXMLStructuredGridWriter
```

37.161.2 Methods

The class vtkXMLStructuredGridWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLStructuredGridWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLStructuredGridWriter = obj.NewInstance ()`
- `vtkXMLStructuredGridWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.162 vtkXMLUnstructuredDataReader

37.162.1 Usage

vtkXMLUnstructuredDataReader provides functionality common to all unstructured data format readers.

To create an instance of class vtkXMLUnstructuredDataReader, simply invoke its constructor as follows

```
obj = vtkXMLUnstructuredDataReader
```

37.162.2 Methods

The class vtkXMLUnstructuredDataReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLUnstructuredDataReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLUnstructuredDataReader = obj.NewInstance ()`
- `vtkXMLUnstructuredDataReader = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetNumberOfPoints ()` - Get the number of points in the output.
- `vtkIdType = obj.GetNumberOfCells ()` - Get the number of cells in the output.
- `obj.SetupUpdateExtent (int piece, int numberOfPieces, int ghostLevel)` - Setup the reader as if the given update extent were requested by its output. This can be used after an UpdateInformation to validate GetNumberOfPoints() and GetNumberOfCells() without actually reading data.
- `obj.CopyOutputInformation (vtkInformation outInfo, int port)`

37.163 vtkXMLUnstructuredDataWriter

37.163.1 Usage

vtkXMLUnstructuredDataWriter provides VTK XML writing functionality that is common among all the unstructured data formats.

To create an instance of class vtkXMLUnstructuredDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLUnstructuredDataWriter
```

37.163.2 Methods

The class vtkXMLUnstructuredDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLUnstructuredDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLUnstructuredDataWriter = obj.NewInstance ()`
- `vtkXMLUnstructuredDataWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfPieces (int )` - Get/Set the number of pieces used to stream the image through the pipeline while writing to the file.
- `int = obj.GetNumberOfPieces ()` - Get/Set the number of pieces used to stream the image through the pipeline while writing to the file.
- `obj.SetWritePiece (int )` - Get/Set the piece to write to the file. If this is negative or equal to the NumberOfPieces, all pieces will be written.
- `int = obj.GetWritePiece ()` - Get/Set the piece to write to the file. If this is negative or equal to the NumberOfPieces, all pieces will be written.

- `obj.SetGhostLevel (int )` - Get/Set the ghost level used to pad each piece.
- `int = obj.GetGhostLevel ()` - Get/Set the ghost level used to pad each piece.

37.164 vtkXMLUnstructuredGridReader

37.164.1 Usage

vtkXMLUnstructuredGridReader reads the VTK XML UnstructuredGrid file format. One unstructured grid file can be read to produce one output. Streaming is supported. The standard extension for this reader's file format is "vtu". This reader is also used to read a single piece of the parallel file format.

To create an instance of class vtkXMLUnstructuredGridReader, simply invoke its constructor as follows

```
obj = vtkXMLUnstructuredGridReader
```

37.164.2 Methods

The class vtkXMLUnstructuredGridReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLUnstructuredGridReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLUnstructuredGridReader = obj.NewInstance ()`
- `vtkXMLUnstructuredGridReader = obj.SafeDownCast (vtkObject o)`
- `vtkUnstructuredGrid = obj.GetOutput ()` - Get the reader's output.
- `vtkUnstructuredGrid = obj.GetOutput (int idx)` - Get the reader's output.

37.165 vtkXMLUnstructuredGridWriter

37.165.1 Usage

vtkXMLUnstructuredGridWriter writes the VTK XML UnstructuredGrid file format. One unstructured grid input can be written into one file in any number of streamed pieces (if supported by the rest of the pipeline). The standard extension for this writer's file format is "vtu". This writer is also used to write a single piece of the parallel file format.

To create an instance of class vtkXMLUnstructuredGridWriter, simply invoke its constructor as follows

```
obj = vtkXMLUnstructuredGridWriter
```

37.165.2 Methods

The class vtkXMLUnstructuredGridWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLUnstructuredGridWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLUnstructuredGridWriter = obj.NewInstance ()`

- `vtkXMLUnstructuredGridWriter = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

37.166 vtkXMLUtilities

37.166.1 Usage

vtkXMLUtilities provides XML-related convenience functions.

To create an instance of class vtkXMLUtilities, simply invoke its constructor as follows

```
obj = vtkXMLUtilities
```

37.166.2 Methods

The class vtkXMLUtilities has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLUtilities class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLUtilities = obj.NewInstance ()`
- `vtkXMLUtilities = obj.SafeDownCast (vtkObject o)`

37.167 vtkXMLWriter

37.167.1 Usage

vtkXMLWriter provides methods implementing most of the functionality needed to write VTK XML file formats. Concrete subclasses provide actual writer implementations calling upon this functionality.

To create an instance of class vtkXMLWriter, simply invoke its constructor as follows

```
obj = vtkXMLWriter
```

37.167.2 Methods

The class vtkXMLWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLWriter = obj.NewInstance ()`
- `vtkXMLWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetByteOrder (int )` - Get/Set the byte order of data written to the file. The default is the machine's hardware byte order.
- `int = obj.GetByteOrder ()` - Get/Set the byte order of data written to the file. The default is the machine's hardware byte order.

- `obj.SetByteOrderToBigEndian ()` - Get/Set the byte order of data written to the file. The default is the machine's hardware byte order.
- `obj.SetByteOrderToLittleEndian ()` - Get/Set the byte order of data written to the file. The default is the machine's hardware byte order.
- `obj.SetIdType (int )` - Get/Set the size of the vtkIdType values stored in the file. The default is the real size of vtkIdType.
- `int = obj.GetIdType ()` - Get/Set the size of the vtkIdType values stored in the file. The default is the real size of vtkIdType.
- `obj.SetIdTypeToInt32 ()` - Get/Set the size of the vtkIdType values stored in the file. The default is the real size of vtkIdType.
- `obj.SetIdTypeToInt64 ()` - Get/Set the size of the vtkIdType values stored in the file. The default is the real size of vtkIdType.
- `obj.SetFileName (string )` - Get/Set the name of the output file.
- `string = obj.GetFileName ()` - Get/Set the name of the output file.
- `obj.SetCompressor (vtkDataCompressor )` - Get/Set the compressor used to compress binary and appended data before writing to the file. Default is a vtkZLibDataCompressor.
- `vtkDataCompressor = obj.GetCompressor ()` - Get/Set the compressor used to compress binary and appended data before writing to the file. Default is a vtkZLibDataCompressor.
- `obj.SetCompressorType (int compressorType)` - Convenience functions to set the compressor to certain known types.
- `obj.SetCompressorTypeToNone ()` - Convenience functions to set the compressor to certain known types.
- `obj.SetCompressorTypeToZLib ()` - Get/Set the block size used in compression. When reading, this controls the granularity of how much extra information must be read when only part of the data are requested. The value should be a multiple of the largest scalar data type.
- `obj.SetBlockSize (int blockSize)` - Get/Set the block size used in compression. When reading, this controls the granularity of how much extra information must be read when only part of the data are requested. The value should be a multiple of the largest scalar data type.
- `int = obj.GetBlockSize ()` - Get/Set the block size used in compression. When reading, this controls the granularity of how much extra information must be read when only part of the data are requested. The value should be a multiple of the largest scalar data type.
- `obj.SetDataMode (int )` - Get/Set the data mode used for the file's data. The options are vtkXMLWriter::Ascii, vtkXMLWriter::Binary, and vtkXMLWriter::Appended.
- `int = obj.GetDataMode ()` - Get/Set the data mode used for the file's data. The options are vtkXMLWriter::Ascii, vtkXMLWriter::Binary, and vtkXMLWriter::Appended.
- `obj.SetDataModeToAscii ()` - Get/Set the data mode used for the file's data. The options are vtkXMLWriter::Ascii, vtkXMLWriter::Binary, and vtkXMLWriter::Appended.
- `obj.SetDataModeToBinary ()` - Get/Set the data mode used for the file's data. The options are vtkXMLWriter::Ascii, vtkXMLWriter::Binary, and vtkXMLWriter::Appended.
- `obj.SetDataModeToAppended ()` - Get/Set the data mode used for the file's data. The options are vtkXMLWriter::Ascii, vtkXMLWriter::Binary, and vtkXMLWriter::Appended.

- `obj.SetEncodeAppendedData (int )` - Get/Set whether the appended data section is base64 encoded. If encoded, reading and writing will be slower, but the file will be fully valid XML and text-only. If not encoded, the XML specification will be violated, but reading and writing will be fast. The default is to do the encoding.

- `int = obj.GetEncodeAppendedData ()` - Get/Set whether the appended data section is base64 encoded. If encoded, reading and writing will be slower, but the file will be fully valid XML and text-only. If not encoded, the XML specification will be violated, but reading and writing will be fast. The default is to do the encoding.

- `obj.EncodeAppendedDataOn ()` - Get/Set whether the appended data section is base64 encoded. If encoded, reading and writing will be slower, but the file will be fully valid XML and text-only. If not encoded, the XML specification will be violated, but reading and writing will be fast. The default is to do the encoding.

- `obj.EncodeAppendedDataOff ()` - Get/Set whether the appended data section is base64 encoded. If encoded, reading and writing will be slower, but the file will be fully valid XML and text-only. If not encoded, the XML specification will be violated, but reading and writing will be fast. The default is to do the encoding.

- `obj.SetInput (vtkDataObject )` - Set/Get an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline

- `obj.SetInput (int , vtkDataObject )` - Set/Get an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline

- `vtkDataObject = obj.GetInput (int port)` - Set/Get an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline

- `vtkDataObject = obj.GetInput ()` - Set/Get an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline

- `string = obj.GetDefaultFileExtension ()` - Get the default file extension for files written by this writer.

- `int = obj.Write ()` - Invoke the writer. Returns 1 for success, 0 for failure.

- `obj.SetTimeStep (int )` - Which TimeStep to write.

- `int = obj.GetTimeStep ()` - Which TimeStep to write.

- `int = obj. GetTimeStepRange ()` - Which TimeStepRange to write.

- `obj.SetTimeStepRange (int , int )` - Which TimeStepRange to write.

- `obj.SetTimeStepRange (int  a[2])` - Which TimeStepRange to write.

- `int = obj.GetNumberOfTimeSteps ()` - Set the number of time steps

- `obj.SetNumberOfTimeSteps (int )` - Set the number of time steps

- `obj.Start ()` - API to interface an outside the VTK pipeline control

- `obj.Stop ()` - API to interface an outside the VTK pipeline control

- `obj.WriteNextTime (double time)` - API to interface an outside the VTK pipeline control

37.168 vtkXYZMolReader

37.168.1 Usage

vtkXYZMolReader is a source object that reads Molecule files The FileName must be specified
.SECTION Thanks Dr. Jean M. Favre who developed and contributed this class
To create an instance of class vtkXYZMolReader, simply invoke its constructor as follows

```
obj = vtkXYZMolReader
```

37.168.2 Methods

The class vtkXYZMolReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXYZMolReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXYZMolReader = obj.NewInstance ()`
- `vtkXYZMolReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string name)` - Test whether the file with the given name can be read by this reader.
- `obj.SetTimeStep (int )` - Set the current time step. It should be greater than 0 and smaller than MaxTimeStep.
- `int = obj.GetTimeStep ()` - Set the current time step. It should be greater than 0 and smaller than MaxTimeStep.
- `int = obj.GetMaxTimeStep ()` - Get the maximum time step.

37.169 vtkZLibDataCompressor

37.169.1 Usage

vtkZLibDataCompressor provides a concrete vtkDataCompressor class using zlib for compressing and uncompressing data.
To create an instance of class vtkZLibDataCompressor, simply invoke its constructor as follows

```
obj = vtkZLibDataCompressor
```

37.169.2 Methods

The class vtkZLibDataCompressor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkZLibDataCompressor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkZLibDataCompressor = obj.NewInstance ()`
- `vtkZLibDataCompressor = obj.SafeDownCast (vtkObject o)`

- `long = obj.GetMaximumCompressionSpace (long size)` - Get the maximum space that may be needed to store data of the given uncompressed size after compression. This is the minimum size of the output buffer that can be passed to the four-argument Compress method.
- `obj.SetCompressionLevel (int )` - Get/Set the compression level.
- `int = obj.GetCompressionLevelMinValue ()` - Get/Set the compression level.
- `int = obj.GetCompressionLevelMaxValue ()` - Get/Set the compression level.
- `int = obj.GetCompressionLevel ()` - Get/Set the compression level.

Chapter 38

Visualization Toolkit Parallel Classes

38.1 vtkBranchExtentTranslator

38.1.1 Usage

vtkBranchExtentTranslator is like extent translator, but it uses an alternative source as a whole extent. The whole extent passed is assumed to be a subextent of the original source. we simply take the intersection of the split extent and the whole extdent passed in. We are attempting to make branching pipelines request consistent extents with the same piece requests.

To create an instance of class vtkBranchExtentTranslator, simply invoke its constructor as follows

```
obj = vtkBranchExtentTranslator
```

38.1.2 Methods

The class vtkBranchExtentTranslator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBranchExtentTranslator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBranchExtentTranslator = obj.NewInstance ()`
- `vtkBranchExtentTranslator = obj.SafeDownCast (vtkObject o)`
- `obj.SetOriginalSource (vtkImageData )` - This is the original upstream image source.
- `vtkImageData = obj.GetOriginalSource ()` - This is the original upstream image source.
- `int = obj.PieceToExtent ()` - Generates the extent from the pieces.
- `obj.SetAssignedPiece (int )` - This unstructured extent/piece is store here for the users convenience. It is not used internally. The intent was to let an "assignment" be made when the translator/first source is created. The translator/assignment can be used for any new filter that uses the original source as output. Branches will then have the same assignment.
- `int = obj.GetAssignedPiece ()` - This unstructured extent/piece is store here for the users convenience. It is not used internally. The intent was to let an "assignment" be made when the translator/first source is created. The translator/assignment can be used for any new filter that uses the original source as output. Branches will then have the same assignment.

- `obj.SetAssignedNumberOfPieces (int )` - This unstructured extent/piece is store here for the users convenience. It is not used internally. The intent was to let an "assignment" be made when the translator/first source is created. The translator/assignment can be used for any new filter that uses the original source as output. Branches will then have the same assignment.
- `int = obj.GetAssignedNumberOfPieces ()` - This unstructured extent/piece is store here for the users convenience. It is not used internally. The intent was to let an "assignment" be made when the translator/first source is created. The translator/assignment can be used for any new filter that uses the original source as output. Branches will then have the same assignment.

38.2 vtkCachingInterpolatedVelocityField

38.2.1 Usage

vtkCachingInterpolatedVelocityField acts as a continuous velocity field by performing cell interpolation on the underlying vtkDataSet. This is a concrete sub-class of vtkFunctionSet with NumberOfIndependentVariables = 4 (x,y,z,t) and NumberOfFunctions = 3 (u,v,w). Normally, every time an evaluation is performed, the cell which contains the point (x,y,z) has to be found by calling FindCell. This is a computationally expensive operation. In certain cases, the cell search can be avoided or shortened by providing a guess for the cell id. For example, in streamline integration, the next evaluation is usually in the same or a neighbour cell. For this reason, vtkCachingInterpolatedVelocityField stores the last cell id. If caching is turned on, it uses this id as the starting point.

To create an instance of class vtkCachingInterpolatedVelocityField, simply invoke its constructor as follows

```
obj = vtkCachingInterpolatedVelocityField
```

38.2.2 Methods

The class vtkCachingInterpolatedVelocityField has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCachingInterpolatedVelocityField class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCachingInterpolatedVelocityField = obj.NewInstance ()`
- `vtkCachingInterpolatedVelocityField = obj.SafeDownCast (vtkObject o)`
- `int = obj.FunctionValues (double x, double f)` - Evaluate the velocity field, f=u,v,w, at x, y, z. returns 1 if valid, 0 if test failed
- `int = obj.InsideTest (double x)` - Evaluate the velocity field, f=u,v,w, at x, y, z. returns 1 if valid, 0 if test failed
- `obj.SetDataSet (int I, vtkDataSet dataset, bool staticdataset, vtkAbstractCellLocator locator)` - Add a dataset used by the interpolation function evaluation.
- `string = obj.GetVectorsSelection ()` - If you want to work with an arbitrary vector array, then set its name here. By default this in NULL and the filter will use the active vector array.
- `obj.SelectVectors (string fieldName)` - Return the cell id cached from last evaluation.
- `obj.SetLastCellInfo (vtkIdType c, int datasetindex)` - Return the cell id cached from last evaluation.

- `obj.ClearLastCellInfo ()` - Set the last cell id to -1 so that the next search does not start from the previous cell
- `int = obj.GetLastWeights (double w)` - Returns the interpolation weights/pcoords cached from last evaluation if the cached cell is valid (returns 1). Otherwise, it does not change w and returns 0.
- `int = obj.GetLastLocalCoordinates (double pcoords[3])` - Returns the interpolation weights/pcoords cached from last evaluation if the cached cell is valid (returns 1). Otherwise, it does not change w and returns 0.
- `int = obj.GetCellCacheHit ()` - Caching statistics.
- `int = obj.GetDataSetCacheHit ()` - Caching statistics.
- `int = obj.GetCacheMiss ()` - Caching statistics.

38.3 vtkCollectGraph

38.3.1 Usage

This filter has code to collect a graph from across processes onto vertex 0. Collection can be turned on or off using the "PassThrough" flag.

To create an instance of class vtkCollectGraph, simply invoke its constructor as follows

```
obj = vtkCollectGraph
```

38.3.2 Methods

The class vtkCollectGraph has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCollectGraph class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCollectGraph = obj.NewInstance ()`
- `vtkCollectGraph = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `obj.SetSocketController (vtkSocketController )` - When this filter is being used in client-server mode, this is the controller used to communicate between client and server. Client should not set the other controller.
- `vtkSocketController = obj.GetSocketController ()` - When this filter is being used in client-server mode, this is the controller used to communicate between client and server. Client should not set the other controller.
- `obj.SetPassThrough (int )` - To collect or just copy input to output. Off (collect) by default.
- `int = obj.GetPassThrough ()` - To collect or just copy input to output. Off (collect) by default.
- `obj.PassThroughOn ()` - To collect or just copy input to output. Off (collect) by default.

- `obj.PassThroughOff ()` - To collect or just copy input to output. Off (collect) by default.
- `obj.SetOutputType (int )` - Directedness flag, used to signal whether the output graph is directed or undirected. DIRECTED_OUTPUT expects that this filter is generating a directed graph. UNDIRECTED_OUTPUT expects that this filter is generating an undirected graph. DIRECTED_OUTPUT and UNDIRECTED_OUTPUT flags should only be set on the client filter. Server filters should be set to USE_INPUT_TYPE since they have valid input and the directedness is determined from the input type.
- `int = obj.GetOutputType ()` - Directedness flag, used to signal whether the output graph is directed or undirected. DIRECTED_OUTPUT expects that this filter is generating a directed graph. UNDIRECTED_OUTPUT expects that this filter is generating an undirected graph. DIRECTED_OUTPUT and UNDIRECTED_OUTPUT flags should only be set on the client filter. Server filters should be set to USE_INPUT_TYPE since they have valid input and the directedness is determined from the input type.

38.4 vtkCollectPolyData

38.4.1 Usage

This filter has code to collect polydat from across processes onto node 0. Collection can be turned on or off using the "PassThrough" flag.

To create an instance of class vtkCollectPolyData, simply invoke its constructor as follows

```
obj = vtkCollectPolyData
```

38.4.2 Methods

The class vtkCollectPolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCollectPolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCollectPolyData = obj.NewInstance ()`
- `vtkCollectPolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `obj.SetSocketController (vtkSocketController )` - When this filter is being used in client-server mode, this is the controller used to communicate between client and server. Client should not set the other controller.
- `vtkSocketController = obj.GetSocketController ()` - When this filter is being used in client-server mode, this is the controller used to communicate between client and server. Client should not set the other controller.
- `obj.SetPassThrough (int )` - To collect or just copy input to output. Off (collect) by default.
- `int = obj.GetPassThrough ()` - To collect or just copy input to output. Off (collect) by default.

- `obj.PassThroughOn ()` - To collect or just copy input to output. Off (collect) by default.
- `obj.PassThroughOff ()` - To collect or just copy input to output. Off (collect) by default.

38.5 vtkCollectTable

38.5.1 Usage

This filter has code to collect a table from across processes onto node 0. Collection can be turned on or off using the "PassThrough" flag.

To create an instance of class vtkCollectTable, simply invoke its constructor as follows

```
obj = vtkCollectTable
```

38.5.2 Methods

The class vtkCollectTable has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCollectTable class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCollectTable = obj.NewInstance ()`
- `vtkCollectTable = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `obj.SetSocketController (vtkSocketController )` - When this filter is being used in client-server mode, this is the controller used to communicate between client and server. Client should not set the other controller.
- `vtkSocketController = obj.GetSocketController ()` - When this filter is being used in client-server mode, this is the controller used to communicate between client and server. Client should not set the other controller.
- `obj.SetPassThrough (int )` - To collect or just copy input to output. Off (collect) by default.
- `int = obj.GetPassThrough ()` - To collect or just copy input to output. Off (collect) by default.
- `obj.PassThroughOn ()` - To collect or just copy input to output. Off (collect) by default.
- `obj.PassThroughOff ()` - To collect or just copy input to output. Off (collect) by default.

38.6 vtkCommunicator

38.6.1 Usage

This is an abstact class which contains functionality for sending and receiving inter-process messages. It contains methods for marshaling an object into a string (currently used by the MPI communicator but not the shared memory communicator).

To create an instance of class vtkCommunicator, simply invoke its constructor as follows

```
obj = vtkCommunicator
```

38.6.2 Methods

The class vtkCommunicator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCommunicator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCommunicator = obj.NewInstance ()`
- `vtkCommunicator = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfProcesses (int num)` - Set the number of processes you will be using. This defaults to the maximum number available. If you set this to a value higher than the default, you will get an error.
- `int = obj.GetNumberOfProcesses ()` - Set the number of processes you will be using. This defaults to the maximum number available. If you set this to a value higher than the default, you will get an error.
- `int = obj.GetLocalProcessId ()` - Tells you which process [0, NumProcess) you are in.
- `int = obj.Send (vtkDataObject data, int remoteHandle, int tag)` - This method sends a data object to a destination. Tag eliminates ambiguity and is used to match sends to receives.
- `int = obj.Send (vtkDataArray data, int remoteHandle, int tag)` - This method sends a data array to a destination. Tag eliminates ambiguity and is used to match sends to receives.
- `int = obj.Send (int data, vtkIdType length, int remoteHandle, int tag)` - Convenience methods for sending data arrays.
- `int = obj.Send (int data, vtkIdType length, int remoteHandle, int tag)` - Convenience methods for sending data arrays.
- `int = obj.Send (long data, vtkIdType length, int remoteHandle, int tag)` - Convenience methods for sending data arrays.
- `int = obj.Send (string data, vtkIdType length, int remoteHandle, int tag)` - Convenience methods for sending data arrays.
- `int = obj.Send (string data, vtkIdType length, int remoteHandle, int tag)` - Convenience methods for sending data arrays.
- `int = obj.Send (float data, vtkIdType length, int remoteHandle, int tag)` - Convenience methods for sending data arrays.
- `int = obj.Send (double data, vtkIdType length, int remoteHandle, int tag)` - Convenience methods for sending data arrays.
- `int = obj.Receive (vtkDataObject data, int remoteHandle, int tag)` - This method receives a data object from a corresponding send. It blocks until the receive is finished.
- `vtkDataObject = obj.ReceiveDataObject (int remoteHandle, int tag)` - The caller does not have to know the data type before this call is made. It returns the newly created object.
- `int = obj.Receive (vtkDataArray data, int remoteHandle, int tag)` - This method receives a data array from a corresponding send. It blocks until the receive is finished.

- `int = obj.Receive (int data, vtkIdType maxlength, int remoteHandle, int tag)` - Convenience methods for receiving data arrays.
- `int = obj.Receive (int data, vtkIdType maxlength, int remoteHandle, int tag)` - Convenience methods for receiving data arrays.
- `int = obj.Receive (long data, vtkIdType maxlength, int remoteHandle, int tag)` - Convenience methods for receiving data arrays.
- `int = obj.Receive (string data, vtkIdType maxlength, int remoteHandle, int tag)` - Convenience methods for receiving data arrays.
- `int = obj.Receive (string data, vtkIdType maxlength, int remoteHandle, int tag)` - Convenience methods for receiving data arrays.
- `int = obj.Receive (float data, vtkIdType maxlength, int remoteHandle, int tag)` - Convenience methods for receiving data arrays.
- `int = obj.Receive (double data, vtkIdType maxlength, int remoteHandle, int tag)` - Convenience methods for receiving data arrays.
- `vtkIdType = obj.GetCount ()` - Returns the number of words received by the most recent Receive(). Note that this is not the number of bytes received, but the number of items of the data-type received by the most recent Receive() eg. if Receive(int*,..) was used, then this returns the number of ints received; if Receive(double*,..) was used, then this returns the number of doubles received etc. The return value is valid only after a successful Receive().
- `obj.Barrier ()` - Will block the processes until all other processes reach the Barrier function.
- `int = obj.Broadcast (int data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (long data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (string data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (string data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (float data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (double data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (vtkDataObject data, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (vtkDataArray data, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.

- `int = obj.Gather (int sendBuffer, int recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ɖestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (long sendBuffer, long recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ɖestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (string sendBuffer, string recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ɖestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (string sendBuffer, string recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ɖestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (float sendBuffer, float recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ɖestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (double sendBuffer, double recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ɖestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (vtkDataArray sendBuffer, vtkDataArray recvBuffer, int destProcessId)` - Gather collects arrays in the process with id ɖestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Scatter (int sendBuffer, int recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id şrcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first ļength values, process 1 receives the second ļength values, and so on. Scatter is the inverse operation of Gather.
- `int = obj.Scatter (long sendBuffer, long recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id şrcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first ļength values, process 1 receives the second ļength values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (string sendBuffer, string recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (string sendBuffer, string recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (float sendBuffer, float recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (double sendBuffer, double recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (vtkDataArray sendBuffer, vtkDataArray recvBuffer, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.AllGather (int sendBuffer, int recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.

- `int = obj.AllGather (long sendBuffer, long recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.

- `int = obj.AllGather (string sendBuffer, string recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.

- `int = obj.AllGather (string sendBuffer, string recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.

- `int = obj.AllGather (float sendBuffer, float recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.

- `int = obj.AllGather (double sendBuffer, double recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.

- `int = obj.AllGather (vtkDataArray sendBuffer, vtkDataArray recvBuffer)` - Same as gather except that the result ends up on all processes.

- `int = obj.Reduce (int sendBuffer, int recvBuffer, vtkIdType length, int operation, int destProcessI` - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.

- `int = obj.Reduce (long sendBuffer, long recvBuffer, vtkIdType length, int operation, int destProces` - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.

- `int = obj.Reduce (string sendBuffer, string recvBuffer, vtkIdType length, int operation, int destPr` - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.

- `int = obj.Reduce (string sendBuffer, string recvBuffer, vtkIdType length, int operation, int destPr`
  - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.Reduce (float sendBuffer, float recvBuffer, vtkIdType length, int operation, int destProc`
  - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.Reduce (double sendBuffer, double recvBuffer, vtkIdType length, int operation, int destPr`
  - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.Reduce (vtkDataArray sendBuffer, vtkDataArray recvBuffer, int operation, int destProcessI`
  - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.AllReduce (int sendBuffer, int recvBuffer, vtkIdType length, int operation)`
  - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (long sendBuffer, long recvBuffer, vtkIdType length, int operation)`
  - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (string sendBuffer, string recvBuffer, vtkIdType length, int operation)`
  - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (string sendBuffer, string recvBuffer, vtkIdType length, int operation)`
  - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (float sendBuffer, float recvBuffer, vtkIdType length, int operation)`
  - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (double sendBuffer, double recvBuffer, vtkIdType length, int operation)`
  - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (vtkDataArray sendBuffer, vtkDataArray recvBuffer, int operation)`
  - Same as Reduce except that the result is placed in all of the processes.

38.7 vtkCompositer

38.7.1 Usage

vtkCompositer operates in multiple processes. Each compositer has a render window. They use vtkMultiProcessControllers to communicate the color and depth buffer to process 0's render window. It will not handle transparency well.

To create an instance of class vtkCompositer, simply invoke its constructor as follows

```
obj = vtkCompositer
```

38.7.2 Methods

The class vtkCompositer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkCompositer = obj.NewInstance ()`
- `vtkCompositer = obj.SafeDownCast (vtkObject o)`
- `obj.CompositeBuffer (vtkDataArray pBuf, vtkFloatArray zBuf, vtkDataArray pTmp, vtkFloatArray zTmp)` - This method gets called on every process. The final image gets put into pBuf and zBuf.
- `obj.SetController (vtkMultiProcessController )` - Access to the controller.
- `vtkMultiProcessController = obj.GetController ()` - Access to the controller.
- `obj.SetNumberOfProcesses (int )` - A hack to get a sub world until I can get communicators working.
- `int = obj.GetNumberOfProcesses ()` - A hack to get a sub world until I can get communicators working.

38.8 vtkCompositeRenderManager

38.8.1 Usage

vtkCompositeRenderManager is a subclass of vtkParallelRenderManager that uses compositing to do parallel rendering. This class has replaced vtkCompositeManager.

To create an instance of class vtkCompositeRenderManager, simply invoke its constructor as follows

```
obj = vtkCompositeRenderManager
```

38.8.2 Methods

The class vtkCompositeRenderManager has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositeRenderManager class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositeRenderManager = obj.NewInstance ()`
- `vtkCompositeRenderManager = obj.SafeDownCast (vtkObject o)`
- `obj.SetCompositer (vtkCompositer c)` - Set/Get the composite algorithm.
- `vtkCompositer = obj.GetCompositer ()` - Set/Get the composite algorithm.
- `double = obj.GetImageProcessingTime ()` - Get rendering metrics.

38.9 vtkCompressCompositer

38.9.1 Usage

vtkCompressCompositer operates in multiple processes. Each compositer has a render window. They use vtkMultiProcessController to communicate the color and depth buffer to process 0's render window. It will not handle transparency. Compositing is run length encoding of background pixels.

SECTION See Also vtkCompositeManager.

To create an instance of class vtkCompressCompositer, simply invoke its constructor as follows

```
obj = vtkCompressCompositer
```

38.9.2 Methods

The class vtkCompressCompositer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompressCompositer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompressCompositer = obj.NewInstance ()`
- `vtkCompressCompositer = obj.SafeDownCast (vtkObject o)`
- `obj.CompositeBuffer (vtkDataArray pBuf, vtkFloatArray zBuf, vtkDataArray pTmp, vtkFloatArray zTmp)`

38.10 vtkCutMaterial

38.10.1 Usage

vtkCutMaterial computes a cut plane based on an up vector, center of the bounding box and the location of the maximum variable value. These computed values are available so that they can be used to set the camera for the best view of the plane.

To create an instance of class vtkCutMaterial, simply invoke its constructor as follows

```
obj = vtkCutMaterial
```

38.10.2 Methods

The class vtkCutMaterial has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCutMaterial class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCutMaterial = obj.NewInstance ()`
- `vtkCutMaterial = obj.SafeDownCast (vtkObject o)`
- `obj.SetMaterialArrayName (string )` - Cell array that contains the material values.
- `string = obj.GetMaterialArrayName ()` - Cell array that contains the material values.
- `obj.SetMaterial (int )` - Material to probe.
- `int = obj.GetMaterial ()` - Material to probe.
- `obj.SetArrayName (string )` - For now, we just use the cell values. The array name to cut.
- `string = obj.GetArrayName ()` - For now, we just use the cell values. The array name to cut.
- `obj.SetUpVector (double , double , double )` - The last piece of information that specifies the plane.
- `obj.SetUpVector (double  a[3])` - The last piece of information that specifies the plane.
- `double = obj. GetUpVector ()` - The last piece of information that specifies the plane.

- `double = obj. GetMaximumPoint ()` - Accesses to the values computed during the execute method. They could be used to get a good camera view for the resulting plane.
- `double = obj. GetCenterPoint ()` - Accesses to the values computed during the execute method. They could be used to get a good camera view for the resulting plane.
- `double = obj. GetNormal ()` - Accesses to the values computed during the execute method. They could be used to get a good camera view for the resulting plane.

38.11 vtkDistributedDataFilter

38.11.1 Usage

This filter redistributes data among processors in a parallel application into spatially contiguous vtkUnstructuredGrids. The execution model anticipated is that all processes read in part of a large vtkDataSet. Each process sets the input of filter to be that DataSet. When executed, this filter builds in parallel a k-d tree, decomposing the space occupied by the distributed DataSet into spatial regions. It assigns each spatial region to a processor. The data is then redistributed and the output is a single vtkUnstructuredGrid containing the cells in the process' assigned regions.

This filter is sometimes called "D3" for "distributed data decomposition".

Enhancement: You can set the k-d tree decomposition, rather than have D3 compute it. This allows you to divide a dataset using the decomposition computed for another dataset. Obtain a description of the k-d tree cuts this way:

vtkBSPCuts *cuts = D3Object1-¿GetCuts()

And set it this way:

D3Object2-¿SetCuts(cuts)

It is desirable to have a field array of global node IDs for two reasons:

1. When merging together sub grids that were distributed across processors, global node IDs can be used to remove duplicate points and significantly reduce the size of the resulting output grid. If no such array is available, D3 will use a tolerance to merge points, which is much slower.

2. If ghost cells have been requested, D3 requires a global node ID array in order to request and transfer ghost cells in parallel among the processors. If there is no global node ID array, D3 will in parallel create a global node ID array, and the time to do this can be significant.

If you know the name of a global node ID array in the input dataset, set that name with this method. If you leave it unset, D3 will search the input data set for certain common names of global node ID arrays. If none is found, and ghost cells have been requested, D3 will create a temporary global node ID array before aquiring ghost cells. It is also desirable to have global element IDs. However, if they don't exist D3 can create them relatively quickly. Set the name of the global element ID array if you have it. If it is not set, D3 will search for it using common names. If still not found, D3 will create a temporary array of global element IDs.

To create an instance of class vtkDistributedDataFilter, simply invoke its constructor as follows

```
obj = vtkDistributedDataFilter
```

38.11.2 Methods

The class vtkDistributedDataFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDistributedDataFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDistributedDataFilter = obj.NewInstance ()`

- `vtkDistributedDataFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController c)` - Set/Get the communicator object
- `vtkMultiProcessController = obj.GetController ()` - Set/Get the communicator object
- `vtkPKdTree = obj.GetKdtree ()`
- `obj.RetainKdtreeOn ()`
- `obj.RetainKdtreeOff ()`
- `int = obj.GetRetainKdtree ()`
- `obj.SetRetainKdtree (int )`
- `obj.IncludeAllIntersectingCellsOn ()`
- `obj.IncludeAllIntersectingCellsOff ()`
- `int = obj.GetIncludeAllIntersectingCells ()`
- `obj.SetIncludeAllIntersectingCells (int )`
- `obj.ClipCellsOn ()`
- `obj.ClipCellsOff ()`
- `int = obj.GetClipCells ()`
- `obj.SetClipCells (int )`
- `obj.SetBoundaryMode (int mode)` - Handling of ClipCells and IncludeAllIntersectingCells.
- `obj.SetBoundaryModeToAssignToOneRegion ()` - Handling of ClipCells and IncludeAllIntersecting-Cells.
- `obj.SetBoundaryModeToAssignToAllIntersectingRegions ()` - Handling of ClipCells and IncludeAllIntersectingCells.
- `obj.SetBoundaryModeToSplitBoundaryCells ()` - Handling of ClipCells and IncludeAllIntersecting-Cells.
- `int = obj.GetBoundaryMode ()` - Handling of ClipCells and IncludeAllIntersectingCells.
- `obj.UseMinimalMemoryOn ()`
- `obj.UseMinimalMemoryOff ()`
- `int = obj.GetUseMinimalMemory ()`
- `obj.SetUseMinimalMemory (int )`
- `obj.TimingOn ()`
- `obj.TimingOff ()`
- `obj.SetTiming (int )`
- `int = obj.GetTiming ()`

- `vtkBSPCuts = obj.GetCuts ()` - You can set the k-d tree decomposition, rather than have D3 compute it. This allows you to divide a dataset using the decomposition computed for another dataset. Obtain a description of the k-d tree cuts this way:

  vtkBSPCuts *cuts = D3Object1-¿GetCuts()

  And set it this way:

  D3Object2-¿SetCuts(cuts)

- `obj.SetCuts (vtkBSPCuts cuts)` - You can set the k-d tree decomposition, rather than have D3 compute it. This allows you to divide a dataset using the decomposition computed for another dataset. Obtain a description of the k-d tree cuts this way:

  vtkBSPCuts *cuts = D3Object1-¿GetCuts()

  And set it this way:

  D3Object2-¿SetCuts(cuts)

38.12 vtkDistributedStreamTracer

38.12.1 Usage

This filter integrates streamlines on a distributed dataset. It is essentially a serial algorithm: only one process is active at one time and it is not more efficient than a single process integration. It is useful when the data is too large to be on one process and has to be kept distributed.

To create an instance of class vtkDistributedStreamTracer, simply invoke its constructor as follows

```
obj = vtkDistributedStreamTracer
```

38.12.2 Methods

The class vtkDistributedStreamTracer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDistributedStreamTracer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDistributedStreamTracer = obj.NewInstance ()`
- `vtkDistributedStreamTracer = obj.SafeDownCast (vtkObject o)`

38.13 vtkDummyCommunicator

38.13.1 Usage

This is a dummy communicator, which can be used by applications that always require a controller but are also compiled on systems without threads or MPI. Because there is always only one process, no real communication takes place.

To create an instance of class vtkDummyCommunicator, simply invoke its constructor as follows

```
obj = vtkDummyCommunicator
```

38.13.2 Methods

The class vtkDummyCommunicator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDummyCommunicator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDummyCommunicator = obj.NewInstance ()`
- `vtkDummyCommunicator = obj.SafeDownCast (vtkObject o)`

38.14 vtkDummyController

38.14.1 Usage

This is a dummy controller which can be used by applications which always require a controller but are also compile on systems without threads or mpi.

To create an instance of class vtkDummyController, simply invoke its constructor as follows

```
obj = vtkDummyController
```

38.14.2 Methods

The class vtkDummyController has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDummyController class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDummyController = obj.NewInstance ()`
- `vtkDummyController = obj.SafeDownCast (vtkObject o)`
- `obj.Finalize ()` - This method is for setting up the processes.
- `obj.Finalize (int )` - This method always returns 0.
- `int = obj.GetLocalProcessId ()` - Directly calls the single method.
- `obj.SingleMethodExecute ()` - Directly calls the single method.
- `obj.MultipleMethodExecute ()` - Directly calls multiple method 0.
- `obj.CreateOutputWindow ()` - If you don't need any special functionality from the controller, you can swap out the dummy communicator for another one.
- `vtkCommunicator = obj.GetCommunicator ()` - If you don't need any special functionality from the controller, you can swap out the dummy communicator for another one.
- `vtkCommunicator = obj.GetRMICommunicator ()` - If you don't need any special functionality from the controller, you can swap out the dummy communicator for another one.
- `obj.SetCommunicator (vtkCommunicator )` - If you don't need any special functionality from the controller, you can swap out the dummy communicator for another one.
- `obj.SetRMICommunicator (vtkCommunicator )` - If you don't need any special functionality from the controller, you can swap out the dummy communicator for another one.

38.15 vtkDuplicatePolyData

38.15.1 Usage

This filter collects poly data and duplicates it on every node. Converts data parallel so every node has a complete copy of the data. The filter is used at the end of a pipeline for driving a tiled display.

To create an instance of class vtkDuplicatePolyData, simply invoke its constructor as follows

```
obj = vtkDuplicatePolyData
```

38.15.2 Methods

The class vtkDuplicatePolyData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDuplicatePolyData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDuplicatePolyData = obj.NewInstance ()`
- `vtkDuplicatePolyData = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `obj.InitializeSchedule (int numProcs)`
- `obj.SetSynchronous (int )` - This flag causes sends and receives to be matched. When this flag is off, two sends occur then two receives. I want to see if it makes a difference in performance. The flag is on by default.
- `int = obj.GetSynchronous ()` - This flag causes sends and receives to be matched. When this flag is off, two sends occur then two receives. I want to see if it makes a difference in performance. The flag is on by default.
- `obj.SynchronousOn ()` - This flag causes sends and receives to be matched. When this flag is off, two sends occur then two receives. I want to see if it makes a difference in performance. The flag is on by default.
- `obj.SynchronousOff ()` - This flag causes sends and receives to be matched. When this flag is off, two sends occur then two receives. I want to see if it makes a difference in performance. The flag is on by default.
- `vtkSocketController = obj.GetSocketController ()` - This duplicate filter works in client server mode when this controller is set. We have a client flag to diferentiate the client and server because the socket controller is odd: Proth processes think their id is 0.
- `obj.SetSocketController (vtkSocketController controller)` - This duplicate filter works in client server mode when this controller is set. We have a client flag to diferentiate the client and server because the socket controller is odd: Proth processes think their id is 0.
- `obj.SetClientFlag (int )` - This duplicate filter works in client server mode when this controller is set. We have a client flag to diferentiate the client and server because the socket controller is odd: Proth processes think their id is 0.

- `int = obj.GetClientFlag ()` - This duplicate filter works in client server mode when this controller is set. We have a client flag to diferentiate the client and server because the socket controller is odd: Proth processes think their id is 0.
- `long = obj.GetMemorySize ()` - This returns to size of the output (on this process). This method is not really used. It is needed to have the same API as vtkCollectPolyData.

38.16 vtkEnSightWriter

38.16.1 Usage

vtkEnSightWriter is a source object that writes binary unstructured grid data files in EnSight format. See EnSight Manual for format details

To create an instance of class vtkEnSightWriter, simply invoke its constructor as follows

```
obj = vtkEnSightWriter
```

38.16.2 Methods

The class vtkEnSightWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEnSightWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEnSightWriter = obj.NewInstance ()`
- `vtkEnSightWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetProcessNumber (int )`
- `int = obj.GetProcessNumber ()`
- `obj.SetPath (string )` - Specify path of EnSight data files to write.
- `string = obj.GetPath ()` - Specify path of EnSight data files to write.
- `obj.SetBaseName (string )` - Specify base name of EnSight data files to write.
- `string = obj.GetBaseName ()` - Specify base name of EnSight data files to write.
- `obj.SetFileName (string )` - Specify the path and base name of the output files.
- `string = obj.GetFileName ()` - Specify the path and base name of the output files.
- `obj.SetTimeStep (int )`
- `int = obj.GetTimeStep ()`
- `obj.SetGhostLevel (int )`
- `int = obj.GetGhostLevel ()`
- `obj.SetTransientGeometry (bool )`
- `bool = obj.GetTransientGeometry ()`
- `obj.SetNumberOfBlocks (int )`

- `int = obj.GetNumberOfBlocks ()`
- `obj.SetBlockIDs (int val)`
- `obj.SetInput (vtkUnstructuredGrid input)` - Specify the input data or filter.
- `vtkUnstructuredGrid = obj.GetInput ()` - Specify the input data or filter.
- `obj.WriteCaseFile (int TotalTimeSteps)`
- `obj.WriteSOSCaseFile (int NumProcs)`
- `obj.SetModelMetadata (vtkModelMetadata model)`
- `vtkModelMetadata = obj.GetModelMetadata ()`

38.17 vtkExodusIIWriter

38.17.1 Usage

This is a vtkWriter that writes it's vtkUnstructuredGrid input out to an Exodus II file. Go to http://endo.sandia.gov/SEACAS for more information about the Exodus II format.

Exodus files contain much information that is not captured in a vtkUnstructuredGrid, such as time steps, information lines, node sets, and side sets. This information can be stored in a vtkModelMetadata object.

The vtkExodusReader and vtkPExodusReader can create a vtkModelMetadata object and embed it in a vtkUnstructuredGrid in a series of field arrays. This writer searches for these field arrays and will use the metadata contained in them when creating the new Exodus II file.

You can also explicitly give the vtkExodusIIWriter a vtkModelMetadata object to use when writing the file.

In the absence of the information provided by vtkModelMetadata, if this writer is not part of a parallel application, we will use reasonable defaults for all the values in the output Exodus file. If you don't provide a block ID element array, we'll create a block for each cell type that appears in the unstructured grid.

However if this writer is part of a parallel application (hence writing out a distributed Exodus file), then we need at the very least a list of all the block IDs that appear in the file. And we need the element array of block IDs for the input unstructured grid.

In the absense of a vtkModelMetadata object, you can also provide time step information which we will include in the output Exodus file.

.SECTION Caveats If the input floating point field arrays and point locations are all floats or all doubles, this class will operate more efficiently. Mixing floats and doubles will slow you down, because Exodus II requires that we write only floats or only doubles.

We use the terms "point" and "node" interchangeably. Also, we use the terms "element" and "cell" interchangeably.

To create an instance of class vtkExodusIIWriter, simply invoke its constructor as follows

```
obj = vtkExodusIIWriter
```

38.17.2 Methods

The class vtkExodusIIWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExodusIIWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExodusIIWriter = obj.NewInstance ()`

- `vtkExodusIIWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetModelMetadata (vtkModelMetadata )`
- `vtkModelMetadata = obj.GetModelMetadata ()`
- `obj.SetFileName (string )`
- `string = obj.GetFileName ()`
- `obj.SetStoreDoubles (int )`
- `int = obj.GetStoreDoubles ()`
- `obj.SetGhostLevel (int )`
- `int = obj.GetGhostLevel ()`
- `obj.SetWriteOutBlockIdArray (int )`
- `int = obj.GetWriteOutBlockIdArray ()`
- `obj.WriteOutBlockIdArrayOn ()`
- `obj.WriteOutBlockIdArrayOff ()`
- `obj.SetWriteOutGlobalNodeIdArray (int )`
- `int = obj.GetWriteOutGlobalNodeIdArray ()`
- `obj.WriteOutGlobalNodeIdArrayOn ()`
- `obj.WriteOutGlobalNodeIdArrayOff ()`
- `obj.SetWriteOutGlobalElementIdArray (int )`
- `int = obj.GetWriteOutGlobalElementIdArray ()`
- `obj.WriteOutGlobalElementIdArrayOn ()`
- `obj.WriteOutGlobalElementIdArrayOff ()`
- `obj.SetWriteAllTimeSteps (int )`
- `int = obj.GetWriteAllTimeSteps ()`
- `obj.WriteAllTimeStepsOn ()`
- `obj.WriteAllTimeStepsOff ()`
- `obj.SetBlockIdArrayName (string )`
- `string = obj.GetBlockIdArrayName ()`

38.18 vtkExtractCTHPart

38.18.1 Usage

vtkExtractCTHPart is a filter that is specialized for creating visualization of a CTH simulation. First it converts the cell data to point data. It contours the selected volume fraction at a value of 0.5. The user has the option of clipping the part with a plane. Clipped surfaces of the part are generated.

To create an instance of class vtkExtractCTHPart, simply invoke its constructor as follows

```
obj = vtkExtractCTHPart
```

38.18.2 Methods

The class vtkExtractCTHPart has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractCTHPart class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractCTHPart = obj.NewInstance ()`
- `vtkExtractCTHPart = obj.SafeDownCast (vtkObject o)`
- `obj.RemoveDoubleVolumeArrayNames ()` - Names of cell volume fraction arrays to extract.
- `obj.RemoveFloatVolumeArrayNames ()` - Names of cell volume fraction arrays to extract.
- `obj.RemoveUnsignedCharVolumeArrayNames ()` - Names of cell volume fraction arrays to extract.
- `int = obj.GetNumberOfVolumeArrayNames ()` - Names of cell volume fraction arrays to extract.
- `string = obj.GetVolumeArrayName (int idx)` - Names of cell volume fraction arrays to extract.
- `obj.RemoveAllVolumeArrayNames ()` - Names of cell volume fraction arrays to extract. for backwards compatibility
- `obj.AddDoubleVolumeArrayName (string arrayName)`
- `obj.AddFloatVolumeArrayName (string arrayName)`
- `obj.AddUnsignedCharVolumeArrayName (string arrayName)`
- `obj.AddVolumeArrayName (string arrayName)`
- `obj.SetClipPlane (vtkPlane clipPlane)` - Set, get or maninpulate the implicit clipping plane.
- `vtkPlane = obj.GetClipPlane ()` - Set, get or maninpulate the implicit clipping plane.
- `long = obj.GetMTime ()` - Look at clip plane to compute MTime.
- `obj.SetController (vtkMultiProcessController controller)` - Set the controller used to coordinate parallel processing.
- `vtkMultiProcessController = obj.GetController ()` - Return the controller used to coordinate parallel processing. By default, it is the global controller.
- `obj.SetVolumeFractionSurfaceValue (double )` - Set and get the volume fraction surface value. This value should be between 0 and 1
- `double = obj.GetVolumeFractionSurfaceValueMinValue ()` - Set and get the volume fraction surface value. This value should be between 0 and 1
- `double = obj.GetVolumeFractionSurfaceValueMaxValue ()` - Set and get the volume fraction surface value. This value should be between 0 and 1
- `double = obj.GetVolumeFractionSurfaceValue ()` - Set and get the volume fraction surface value. This value should be between 0 and 1

38.19 vtkExtractPiece

38.19.1 Usage

vtkExtractPiece returns the appropriate piece of each sub-dataset in the vtkCompositeDataSet. This filter can handle sub-datasets of type vtkImageData, vtkPolyData, vtkRectilinearGrid, vtkStructuredGrid, and vtkUnstructuredGrid; it does not handle sub-grids of type vtkCompositeDataSet.

To create an instance of class vtkExtractPiece, simply invoke its constructor as follows

```
obj = vtkExtractPiece
```

38.19.2 Methods

The class vtkExtractPiece has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractPiece class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractPiece = obj.NewInstance ()`
- `vtkExtractPiece = obj.SafeDownCast (vtkObject o)`

38.20 vtkExtractUserDefinedPiece

38.20.1 Usage

Provided a function that determines which cells are zero-level cells ("the piece"), this class outputs the piece with the requested number of ghost levels. The only difference between this class and the class it is derived from is that the zero-level cells are specified by a function you provide, instead of determined by dividing up the cells based on cell Id.

To create an instance of class vtkExtractUserDefinedPiece, simply invoke its constructor as follows

```
obj = vtkExtractUserDefinedPiece
```

38.20.2 Methods

The class vtkExtractUserDefinedPiece has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExtractUserDefinedPiece class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkExtractUserDefinedPiece = obj.NewInstance ()`
- `vtkExtractUserDefinedPiece = obj.SafeDownCast (vtkObject o)`

38.21 vtkImageRenderManager

38.21.1 Usage

vtkImageRenderManager is a subclass of vtkParallelRenderManager that uses RGBA compositing (blending) to do parallel rendering. This is the exact opposite of vtkCompositeRenderManager. It actually does nothing special. It relies on the rendering pipeline to be initialized with a vtkCompositeRGBAPass. Compositing makes sense only for renderers in layer 0.

To create an instance of class vtkImageRenderManager, simply invoke its constructor as follows

```
obj = vtkImageRenderManager
```

38.21.2 Methods

The class vtkImageRenderManager has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageRenderManager class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageRenderManager = obj.NewInstance ()`
- `vtkImageRenderManager = obj.SafeDownCast (vtkObject o)`

38.22 vtkMemoryLimitImageDataStreamer

38.22.1 Usage

To satisfy a request, this filter calls update on its input many times with smaller update extents. All processing up stream streams smaller pieces.

To create an instance of class vtkMemoryLimitImageDataStreamer, simply invoke its constructor as follows

```
obj = vtkMemoryLimitImageDataStreamer
```

38.22.2 Methods

The class vtkMemoryLimitImageDataStreamer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMemoryLimitImageDataStreamer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMemoryLimitImageDataStreamer = obj.NewInstance ()`
- `vtkMemoryLimitImageDataStreamer = obj.SafeDownCast (vtkObject o)`
- `obj.SetMemoryLimit (long )` - Set / Get the memory limit in kilobytes.
- `long = obj.GetMemoryLimit ()` - Set / Get the memory limit in kilobytes.

38.23 vtkMPIImageReader

38.23.1 Usage

vtkMPIImageReader provides the mechanism to read a brick of bytes (or shorts, or ints, or floats, or doubles, ...) from a file or series of files. You can use it to read raw image data from files. You may also be able to subclass this to read simple file formats.

What distinguishes this class from vtkImageReader and vtkImageReader2 is that it performs synchronized parallel I/O using the MPIIO layer. This can make a huge difference in file read times, especially when reading in parallel from a parallel file system.

Dispite the name of this class, vtkMPIImageReader will work even if MPI is not available. If MPI is not available or MPIIO is not available or the given Controller is not a vtkMPIController (or NULL), then this class will silently work exactly like its superclass. The point is that you can safely use this class in applications that may or may not be compiled with MPI (or may or may not actually be run with MPI).

To create an instance of class vtkMPIImageReader, simply invoke its constructor as follows

```
obj = vtkMPIImageReader
```

38.23.2 Methods

The class vtkMPIImageReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMPIImageReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMPIImageReader = obj.NewInstance ()`
- `vtkMPIImageReader = obj.SafeDownCast (vtkObject o)`
- `vtkMultiProcessController = obj.GetController ()` - Get/set the multi process controller to use for coordinated reads. By default, set to the global controller.
- `obj.SetController (vtkMultiProcessController )` - Get/set the multi process controller to use for coordinated reads. By default, set to the global controller.

38.24 vtkMultiProcessController

38.24.1 Usage

vtkMultiProcessController is used to control multiple processes in a distributed computing environment. It has methods for executing single/multiple method(s) on multiple processors, triggering registered callbacks (Remote Methods) (AddRMI(), TriggerRMI()) and communication. Please note that the communication is done using the communicator which is accessible to the user. Therefore it is possible to get the communicator with GetCommunicator() and use it to send and receive data. This is the encouraged communication method. The internal (RMI) communications are done using a second internal communicator (called RMI-Communicator).

To create an instance of class vtkMultiProcessController, simply invoke its constructor as follows

```
obj = vtkMultiProcessController
```

38.24.2 Methods

The class vtkMultiProcessController has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMultiProcessController class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMultiProcessController = obj.NewInstance ()`
- `vtkMultiProcessController = obj.SafeDownCast (vtkObject o)`
- `obj.Finalize ()` - This method is for cleaning up. If a subclass needs to clean up process communication (i.e. MPI) it would over ride this method.
- `obj.Finalize (int finalizedExternally)` - This method is for cleaning up. If a subclass needs to clean up process communication (i.e. MPI) it would over ride this method. Provided for finalization outside vtk.
- `obj.SetNumberOfProcesses (int num)` - Set the number of processes you will be using. This defaults to the maximum number available. If you set this to a value higher than the default, you will get an error.
- `int = obj.GetNumberOfProcesses ()` - Set the number of processes you will be using. This defaults to the maximum number available. If you set this to a value higher than the default, you will get an error.
- `obj.SingleMethodExecute ()` - Execute the SingleMethod (as define by SetSingleMethod) using this-¿NumberOfProcesses processes. This will only return when all the processes finish executing their methods.
- `obj.MultipleMethodExecute ()` - Execute the MultipleMethods (as define by calling SetMultipleMethod for each of the required this-¿NumberOfProcesses methods) using this-¿NumberOfProcesses processes.
- `int = obj.GetLocalProcessId ()` - Tells you which process [0, NumProcess) you are in.
- `obj.CreateOutputWindow ()` - This method can be used to tell the controller to create a special output window in which all messages are preceded by the process id.
- `vtkMultiProcessController = obj.CreateSubController (vtkProcessGroup group)` - Creates a new controller with the processes specified by the given group. The new controller will already be initialized for you. You are responsible for deleting the controller once you are done. It is invalid to pass this method a group with a different communicator than is used by this controller. This operation is collective accross all processes defined in the group. It is undefined what will happen if the group is not the same on all processes. This method must be called by all processes in the controller regardless of whether they are in the group. NULL is returned on all process not in the group.
- `vtkMultiProcessController = obj.PartitionController (int localColor, int localKey)` - Partitions this controller based on a coloring. That is, each process passes in a color. All processes with the same color are grouped into the same partition. The processes are ordered by their self-assigned key. Lower keys have lower process ids. Ties are broken by the current process ids. (For example, if all the keys are 0, then the resulting processes will be ordered in the same way.) This method returns a new controller to each process that represents the local partition. This is basically the same operation as MPI_Comm_split.
- `obj.TriggerBreakRMIs ()` - A conveniance method. Called on process 0 to break "ProcessRMIs" loop on all other processes.

- `obj.TriggerRMI (int remoteProcessId, string arg, int tag)` - Convenience method when there is no argument.

- `obj.TriggerRMI (int remoteProcessId, int tag)` - This is a convenicence method to trigger an RMI call on all the "children" of the current node. The children of the current node can be determined by drawing a binary tree starting at node 0 and then assigned nodes ids incrementally in a breadth-first fashion from left to right. This is designed to be used when trigger an RMI call on all satellites from the root node.

- `obj.TriggerRMIOnAllChildren (string arg, int tag)` - This is a convenicence method to trigger an RMI call on all the "children" of the current node. The children of the current node can be determined by drawing a binary tree starting at node 0 and then assigned nodes ids incrementally in a breadth-first fashion from left to right. This is designed to be used when trigger an RMI call on all satellites from the root node.

- `obj.TriggerRMIOnAllChildren (int tag)` - Calling this method gives control to the controller to start processing RMIs. Possible return values are: RMI_NO_ERROR, RMI_TAG_ERROR : rmi tag could not be received, RMI_ARG_ERROR : rmi arg could not be received. If reportErrors is false, no vtkErrorMacro is called. ProcessRMIs() calls ProcessRMIs(int) with reportErrors = 0. If dont_loop is 1, this call just process one RMI message and exits.

- `int = obj.ProcessRMIs (int reportErrors, int dont\_loop)` - Calling this method gives control to the controller to start processing RMIs. Possible return values are: RMI_NO_ERROR, RMI_TAG_ERROR : rmi tag could not be received, RMI_ARG_ERROR : rmi arg could not be received. If reportErrors is false, no vtkErrorMacro is called. ProcessRMIs() calls ProcessRMIs(int) with reportErrors = 0. If dont_loop is 1, this call just process one RMI message and exits.

- `int = obj.ProcessRMIs ()` - Calling this method gives control to the controller to start processing RMIs. Possible return values are: RMI_NO_ERROR, RMI_TAG_ERROR : rmi tag could not be received, RMI_ARG_ERROR : rmi arg could not be received. If reportErrors is false, no vtkErrorMacro is called. ProcessRMIs() calls ProcessRMIs(int) with reportErrors = 0. If dont_loop is 1, this call just process one RMI message and exits.

- `obj.SetBreakFlag (int )` - Setting this flag to 1 will cause the ProcessRMIs loop to return. This also causes vtkUpStreamPorts to return from their WaitForUpdate loops.

- `int = obj.GetBreakFlag ()` - Setting this flag to 1 will cause the ProcessRMIs loop to return. This also causes vtkUpStreamPorts to return from their WaitForUpdate loops.

- `vtkCommunicator = obj.GetCommunicator ()` - Returns the communicator associated with this controller. A default communicator is created in constructor.

- `obj.Barrier ()` - This method can be used to synchronize processes.

- `int = obj.Send (int data, vtkIdType length, int remoteProcessId, int tag)` - This method sends data to another process. Tag eliminates ambiguity when multiple sends or receives exist in the same process. It is recommended to use custom tag number over 100. vtkMultiProcessController has reserved tags between 1 and 4. vtkCommunicator has reserved tags between 10 and 16.

- `int = obj.Send (int data, vtkIdType length, int remoteProcessId, int tag)` - This method sends data to another process. Tag eliminates ambiguity when multiple sends or receives exist in the same process. It is recommended to use custom tag number over 100. vtkMultiProcessController has reserved tags between 1 and 4. vtkCommunicator has reserved tags between 10 and 16.

- `int = obj.Send (long data, vtkIdType length, int remoteProcessId, int tag)` - This method sends data to another process. Tag eliminates ambiguity when multiple sends or receives exist in the same process. It is recommended to use custom tag number over 100. vtkMultiProcessController has reserved tags between 1 and 4. vtkCommunicator has reserved tags between 10 and 16.

- `int = obj.Send (string data, vtkIdType length, int remoteProcessId, int tag)` - This method sends data to another process. Tag eliminates ambiguity when multiple sends or receives exist in the same process. It is recommended to use custom tag number over 100. vtkMultiProcessController has reserved tags between 1 and 4. vtkCommunicator has reserved tags between 10 and 16.

- `int = obj.Send (string data, vtkIdType length, int remoteProcessId, int tag)` - This method sends data to another process. Tag eliminates ambiguity when multiple sends or receives exist in the same process. It is recommended to use custom tag number over 100. vtkMultiProcessController has reserved tags between 1 and 4. vtkCommunicator has reserved tags between 10 and 16.

- `int = obj.Send (float data, vtkIdType length, int remoteProcessId, int tag)` - This method sends data to another process. Tag eliminates ambiguity when multiple sends or receives exist in the same process. It is recommended to use custom tag number over 100. vtkMultiProcessController has reserved tags between 1 and 4. vtkCommunicator has reserved tags between 10 and 16.

- `int = obj.Send (double data, vtkIdType length, int remoteProcessId, int tag)` - This method sends data to another process. Tag eliminates ambiguity when multiple sends or receives exist in the same process. It is recommended to use custom tag number over 100. vtkMultiProcessController has reserved tags between 1 and 4. vtkCommunicator has reserved tags between 10 and 16.

- `int = obj.Send (vtkDataObject data, int remoteId, int tag)` - This method sends data to another process. Tag eliminates ambiguity when multiple sends or receives exist in the same process. It is recommended to use custom tag number over 100. vtkMultiProcessController has reserved tags between 1 and 4. vtkCommunicator has reserved tags between 10 and 16.

- `int = obj.Send (vtkDataArray data, int remoteId, int tag)` - This method sends data to another process. Tag eliminates ambiguity when multiple sends or receives exist in the same process. It is recommended to use custom tag number over 100. vtkMultiProcessController has reserved tags between 1 and 4. vtkCommunicator has reserved tags between 10 and 16.

- `int = obj.Receive (int data, vtkIdType maxlength, int remoteProcessId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `int = obj.Receive (int data, vtkIdType maxlength, int remoteProcessId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `int = obj.Receive (long data, vtkIdType maxlength, int remoteProcessId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `int = obj.Receive (string data, vtkIdType maxlength, int remoteProcessId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `int = obj.Receive (string data, vtkIdType maxlength, int remoteProcessId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `int = obj.Receive (float data, vtkIdType maxlength, int remoteProcessId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `int = obj.Receive (double data, vtkIdType maxlength, int remoteProcessId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `int = obj.Receive (vtkDataObject data, int remoteId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `int = obj.Receive (vtkDataArray data, int remoteId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `vtkDataObject = obj.ReceiveDataObject (int remoteId, int tag)` - This method receives data from a corresponding send. It blocks until the receive is finished. It calls methods in "data" to communicate the sending data. In the overrloads that take in a maxlength argument, this length is the maximum length of the message to receive. If the maxlength is less than the length of the message sent by the sender, an error will be flagged. Once a message is received, use the GetCount() method to determine the actual size of the data received.

- `vtkIdType = obj.GetCount ()` - Returns the number of words received by the most recent Receive(). Note that this is not the number of bytes received, but the number of items of the data-type received by the most recent Receive() eg. if Receive(int*,..) was used, then this returns the number of ints received; if Receive(double*,..) was used, then this returns the number of doubles received etc. The return value is valid only after a successful Receive().

- `int = obj.Broadcast (int data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id srcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.

- `int = obj.Broadcast (long data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id srcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.

- `int = obj.Broadcast (string data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (string data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (float data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (double data, vtkIdType length, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (vtkDataObject data, int srcProcessId)` - Broadcast sends the array in the process with id şrcProcessId to all of the other processes. All processes must call these method with the same arguments in order for it to complete.
- `int = obj.Broadcast (vtkDataArray data, int srcProcessId)` - Gather collects arrays in the process with id ḑestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (int sendBuffer, int recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ḑestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (long sendBuffer, long recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ḑestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (string sendBuffer, string recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ḑestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (string sendBuffer, string recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ḑestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The ļength argument (which must be the same on all processes) is the length of the sendBuffers. The ŗecvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.
- `int = obj.Gather (float sendBuffer, float recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id ḑestProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the

messages and stores them in rank order. The length argument (which must be the same on all processes) is the length of the sendBuffers. The recvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.

- `int = obj.Gather (double sendBuffer, double recvBuffer, vtkIdType length, int destProcessId)` - Gather collects arrays in the process with id destProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The length argument (which must be the same on all processes) is the length of the sendBuffers. The recvBuffer (on te destination process) must be of length length*numProcesses. Gather is the inverse operation of Scatter.

- `int = obj.Gather (vtkDataArray sendBuffer, vtkDataArray recvBuffer, int destProcessId)` - GatherV is the vector variant of Gather. It extends the functionality of Gather by allowing a varying count of data from each process. GatherV collects arrays in the process with id destProcessId. Each process (including the destination) sends the contents of its send buffer to the destination process. The destination process receives the messages and stores them in rank order. The sendLength argument defines how much the local process sends to destProcessId and recvLengths is an array containing the amount destProcessId receives from each process, in rank order.

- `int = obj.Scatter (int sendBuffer, int recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (long sendBuffer, long recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (string sendBuffer, string recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (string sendBuffer, string recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (float sendBuffer, float recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (double sendBuffer, double recvBuffer, vtkIdType length, int srcProcessId)` - Scatter takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer. Process 0 receives the first length values, process 1 receives the second length values, and so on. Scatter is the inverse operation of Gather.

- `int = obj.Scatter (vtkDataArray sendBuffer, vtkDataArray recvBuffer, int srcProcessId)` - ScatterV is the vector variant of Scatter. It extends the functionality of Scatter by allowing a varying count of data to each process. ScatterV takes an array in the process with id srcProcessId and distributes it. Each process (including the source) receives a portion of the send buffer defined by the sendLengths and offsets arrays.

- `int = obj.AllGather (int sendBuffer, int recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.

- `int = obj.AllGather (long sendBuffer, long recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.
- `int = obj.AllGather (string sendBuffer, string recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.
- `int = obj.AllGather (string sendBuffer, string recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.
- `int = obj.AllGather (float sendBuffer, float recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.
- `int = obj.AllGather (double sendBuffer, double recvBuffer, vtkIdType length)` - Same as gather except that the result ends up on all processes.
- `int = obj.AllGather (vtkDataArray sendBuffer, vtkDataArray recvBuffer)` - Same as GatherV except that the result is placed in all processes.
- `int = obj.Reduce (int sendBuffer, int recvBuffer, vtkIdType length, int operation, int destProcessI` - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.Reduce (long sendBuffer, long recvBuffer, vtkIdType length, int operation, int destProces` - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.Reduce (string sendBuffer, string recvBuffer, vtkIdType length, int operation, int destPr` - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.Reduce (string sendBuffer, string recvBuffer, vtkIdType length, int operation, int destPr` - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.Reduce (float sendBuffer, float recvBuffer, vtkIdType length, int operation, int destProc` - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.Reduce (double sendBuffer, double recvBuffer, vtkIdType length, int operation, int destPr` - Reduce an array to the given destination process. This version of Reduce takes an identifier defined in the vtkCommunicator::StandardOperations enum to define the operation.
- `int = obj.Reduce (vtkDataArray sendBuffer, vtkDataArray recvBuffer, int operation, int destProcessI` - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (int sendBuffer, int recvBuffer, vtkIdType length, int operation)` - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (long sendBuffer, long recvBuffer, vtkIdType length, int operation)` - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (string sendBuffer, string recvBuffer, vtkIdType length, int operation)` - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (string sendBuffer, string recvBuffer, vtkIdType length, int operation)` - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (float sendBuffer, float recvBuffer, vtkIdType length, int operation)` - Same as Reduce except that the result is placed in all of the processes.

- `int = obj.AllReduce (double sendBuffer, double recvBuffer, vtkIdType length, int operation)` - Same as Reduce except that the result is placed in all of the processes.
- `int = obj.AllReduce (vtkDataArray sendBuffer, vtkDataArray recvBuffer, int operation)`

38.25 vtkParallelFactory

38.25.1 Usage

To create an instance of class vtkParallelFactory, simply invoke its constructor as follows

```
obj = vtkParallelFactory
```

38.25.2 Methods

The class vtkParallelFactory has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParallelFactory class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParallelFactory = obj.NewInstance ()`
- `vtkParallelFactory = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetVTKSourceVersion ()`
- `string = obj.GetDescription ()`

38.26 vtkParallelRenderManager

38.26.1 Usage

vtkParallelRenderManager operates in multiple processes. It provides proper renderers and render windows for performing the parallel rendering correctly. It can also attach itself to render windows and propagate rendering events and camera views.

.SECTION Note: Many parallel rendering schemes do not correctly handle transparency. Unless otherwise documented, assume a sub class does not.

.SECTION ToDo: Synchronization/barrier primitives.

Query ranges of scalar values of objects in addition to the boundry in three-space

To create an instance of class vtkParallelRenderManager, simply invoke its constructor as follows

```
obj = vtkParallelRenderManager
```

38.26.2 Methods

The class vtkParallelRenderManager has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParallelRenderManager class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkParallelRenderManager = obj.NewInstance ()`
- `vtkParallelRenderManager = obj.SafeDownCast (vtkObject o)`
- `vtkRenderWindow = obj.MakeRenderWindow ()` - Builds a vtkRenderWindow compatible with this render manager. The user program is responsible for registering the render window with the SetRenderWindow method and calling Delete. It is not advisable to use a parallel render manager with a render window that was not built with this method.
- `vtkRenderer = obj.MakeRenderer ()` - Builds a vtkRenderer compatible with this render manager. (Should we also register it?) The user program is responsible for calling Delete. It is not advisable to use a parallel render manager with a renderer that was not built with this method.
- `vtkRenderWindow = obj.GetRenderWindow ()` - Set/Get the RenderWindow to use for compositing. We add a start and end observer to the window.
- `obj.SetRenderWindow (vtkRenderWindow renWin)` - Set/Get the RenderWindow to use for compositing. We add a start and end observer to the window.
- `vtkMultiProcessController = obj.GetController ()` - Set/Get the vtkMultiProcessController which will handle communications for the parallel rendering.
- `obj.SetController (vtkMultiProcessController controller)` - Set/Get the vtkMultiProcessController which will handle communications for the parallel rendering.
- `obj.InitializePieces ()` - This method sets the piece and number of pieces for each actor with a polydata mapper.
- `obj.InitializeOffScreen ()` - Make all rendering windows not viewable set as off screen rendering. To make all renderwindows on screen rendering again, call OffScreenRenderingOff on all the render windows. This class assumes the window on root node is the only one viewable. Subclasses should change this as necessary.
- `obj.StartInteractor ()` - Initializes the RMIs and then, if on root node, starts the interactor on the attached render window. Otherwise, starts processing RMIs. When the interactor returns, it breaks the RMI listening on all other processors.
- `obj.StartServices ()` - If on node other than root, starts serving RMI requests for parallel renders.
- `obj.StopServices ()` - If on root node, stops the RMI processing on all service nodes.
- `obj.StartRender ()` - Callbacks that initialize and finish rendering and other tasks.
- `obj.EndRender ()` - Callbacks that initialize and finish rendering and other tasks.
- `obj.SatelliteStartRender ()` - Callbacks that initialize and finish rendering and other tasks.
- `obj.SatelliteEndRender ()` - Callbacks that initialize and finish rendering and other tasks.
- `obj.RenderRMI ()` - Callbacks that initialize and finish rendering and other tasks.
- `obj.ResetCamera (vtkRenderer ren)` - Callbacks that initialize and finish rendering and other tasks.
- `obj.ResetCameraClippingRange (vtkRenderer ren)` - Callbacks that initialize and finish rendering and other tasks.
- `obj.ComputeVisiblePropBoundsRMI (int renderId)` - Callbacks that initialize and finish rendering and other tasks.
- `obj.InitializeRMIs ()`

- `obj.ResetAllCameras ()` - Resets the camera of each renderer contained in the RenderWindow. Should only be called in the "root" process, and all remote processes must be processing RMIs for this method to complete.
- `obj.ComputeVisiblePropBounds (vtkRenderer ren, double bounds[6])` - Calculates the bounds by gathering information from all processes.
- `obj.SetParallelRendering (int )` - Turns on/off parallel rendering. When on (the default) the object responds to render events of the attached window, propagates the render event to other processors, and otherwise enables the parallel rendering process.
- `int = obj.GetParallelRendering ()` - Turns on/off parallel rendering. When on (the default) the object responds to render events of the attached window, propagates the render event to other processors, and otherwise enables the parallel rendering process.
- `obj.ParallelRenderingOn ()` - Turns on/off parallel rendering. When on (the default) the object responds to render events of the attached window, propagates the render event to other processors, and otherwise enables the parallel rendering process.
- `obj.ParallelRenderingOff ()` - Turns on/off parallel rendering. When on (the default) the object responds to render events of the attached window, propagates the render event to other processors, and otherwise enables the parallel rendering process.
- `obj.SetRenderEventPropagation (int )` - Turns on/off render event propagation. When on (the default) and ParallelRendering is on, process 0 will send an RMI call to all remote processes to perform a synchronized render. When off, render must be manually called on each process.
- `int = obj.GetRenderEventPropagation ()` - Turns on/off render event propagation. When on (the default) and ParallelRendering is on, process 0 will send an RMI call to all remote processes to perform a synchronized render. When off, render must be manually called on each process.
- `obj.RenderEventPropagationOn ()` - Turns on/off render event propagation. When on (the default) and ParallelRendering is on, process 0 will send an RMI call to all remote processes to perform a synchronized render. When off, render must be manually called on each process.
- `obj.RenderEventPropagationOff ()` - Turns on/off render event propagation. When on (the default) and ParallelRendering is on, process 0 will send an RMI call to all remote processes to perform a synchronized render. When off, render must be manually called on each process.
- `obj.SetUseCompositing (int )` - This is used for tiled display rendering. When data has been duplicated on all processes, then we do not need to compositing. Cameras and renders are still propagated though.
- `int = obj.GetUseCompositing ()` - This is used for tiled display rendering. When data has been duplicated on all processes, then we do not need to compositing. Cameras and renders are still propagated though.
- `obj.UseCompositingOn ()` - This is used for tiled display rendering. When data has been duplicated on all processes, then we do not need to compositing. Cameras and renders are still propagated though.
- `obj.UseCompositingOff ()` - This is used for tiled display rendering. When data has been duplicated on all processes, then we do not need to compositing. Cameras and renders are still propagated though.
- `obj.SetImageReductionFactor (double factor)` - Set/Get the reduction factor (for sort-last based parallel renderers). The size of rendered image is divided by the reduction factor and then is blown up to the size of the current vtkRenderWindow. Setting higher reduction factors enables shorter image transfer times (which is often the bottleneck) but will greatly reduce image quality. A reduction factor of 2 or greater should only be used for intermediate images in interactive applications. A reduction factor of 1 (or less) will result in no change in image quality. A parallel render manager may ignore

the image reduction factor if it will result in little or no performance enhancements (eg. it does not do image space manipulations).

- `double = obj.GetImageReductionFactor ()` - Set/Get the reduction factor (for sort-last based parallel renderers). The size of rendered image is divided by the reduction factor and then is blown up to the size of the current vtkRenderWindow. Setting higher reduction factors enables shorter image transfer times (which is often the bottleneck) but will greatly reduce image quality. A reduction factor of 2 or greater should only be used for intermediate images in interactive applications. A reduction factor of 1 (or less) will result in no change in image quality. A parallel render manager may ignore the image reduction factor if it will result in little or no performance enhancements (eg. it does not do image space manipulations).
- `obj.SetMaxImageReductionFactor (double )`
- `double = obj.GetMaxImageReductionFactor ()`
- `obj.SetImageReductionFactorForUpdateRate (double DesiredUpdateRate)` - Sets the ReductionFactor based on the given desired update rate and the rendering metrics taken from the last time UpdateServerInfo was called. Note that if AutoReductionFactor is on, this function is called with the desired update rate of the render window automatically.
- `obj.SetAutoImageReductionFactor (int )` - If on, the ReductionFactor is automatically adjusted to best meet the the DesiredUpdateRate in the current RenderWindow based on metrics from the last render.
- `int = obj.GetAutoImageReductionFactor ()` - If on, the ReductionFactor is automatically adjusted to best meet the the DesiredUpdateRate in the current RenderWindow based on metrics from the last render.
- `obj.AutoImageReductionFactorOn ()` - If on, the ReductionFactor is automatically adjusted to best meet the the DesiredUpdateRate in the current RenderWindow based on metrics from the last render.
- `obj.AutoImageReductionFactorOff ()` - If on, the ReductionFactor is automatically adjusted to best meet the the DesiredUpdateRate in the current RenderWindow based on metrics from the last render.
- `double = obj.GetRenderTime ()` - Get rendering metrics.
- `double = obj.GetImageProcessingTime ()` - Get rendering metrics.
- `int = obj.GetSyncRenderWindowRenderers ()` - By default, the state of all renderers in the root's render window is propagated to the rest of the processes. In order for this to work, all render windows must have the same renderers in the same order. If this is not the case, you can turn off the SyncRenderWindowRenderers. When this flag is off, the list of renderers held by this parallel render manager (initially empty) is synced. You can modify the list of renderers with the AddRenderer, RemoveRenderer, and RemoveAllRenderers methods.
- `obj.SetSyncRenderWindowRenderers (int )` - By default, the state of all renderers in the root's render window is propagated to the rest of the processes. In order for this to work, all render windows must have the same renderers in the same order. If this is not the case, you can turn off the SyncRenderWindowRenderers. When this flag is off, the list of renderers held by this parallel render manager (initially empty) is synced. You can modify the list of renderers with the AddRenderer, RemoveRenderer, and RemoveAllRenderers methods.
- `obj.SyncRenderWindowRenderersOn ()` - By default, the state of all renderers in the root's render window is propagated to the rest of the processes. In order for this to work, all render windows must have the same renderers in the same order. If this is not the case, you can turn off the SyncRenderWindowRenderers. When this flag is off, the list of renderers held by this parallel render manager (initially empty) is synced. You can modify the list of renderers with the AddRenderer, RemoveRenderer, and RemoveAllRenderers methods.

- `obj.SyncRenderWindowRenderersOff ()` - By default, the state of all renderers in the root's render window is propagated to the rest of the processes. In order for this to work, all render windows must have the same renderers in the same order. If this is not the case, you can turn off the SyncRenderWindowRenderers. When this flag is off, the list of renderers held by this parallel render manager (initially empty) is synced. You can modify the list of renderers with the AddRenderer, RemoveRenderer, and RemoveAllRenderers methods.

- `obj.AddRenderer (vtkRenderer )` - By default, the state of all renderers in the root's render window is propagated to the rest of the processes. In order for this to work, all render windows must have the same renderers in the same order. If this is not the case, you can turn off the SyncRenderWindowRenderers. When this flag is off, the list of renderers held by this parallel render manager (initially empty) is synced. You can modify the list of renderers with the AddRenderer, RemoveRenderer, and RemoveAllRenderers methods.

- `obj.RemoveRenderer (vtkRenderer )` - By default, the state of all renderers in the root's render window is propagated to the rest of the processes. In order for this to work, all render windows must have the same renderers in the same order. If this is not the case, you can turn off the SyncRenderWindowRenderers. When this flag is off, the list of renderers held by this parallel render manager (initially empty) is synced. You can modify the list of renderers with the AddRenderer, RemoveRenderer, and RemoveAllRenderers methods.

- `obj.RemoveAllRenderers ()` - By default, the state of all renderers in the root's render window is propagated to the rest of the processes. In order for this to work, all render windows must have the same renderers in the same order. If this is not the case, you can turn off the SyncRenderWindowRenderers. When this flag is off, the list of renderers held by this parallel render manager (initially empty) is synced. You can modify the list of renderers with the AddRenderer, RemoveRenderer, and RemoveAllRenderers methods.

- `obj.SetWriteBackImages (int )` - If on (the default), the result of any image space manipulations are written back to the render window frame buffer. If off, the image stored in the frame buffer may not be correct. Either way, the correct frame buffer images may be read with vtkParallelRenderManager::GetPixelData. Turning WriteBackImages off may result in a speedup if the render window is not visible to the user and images are read back for further processing or transit.

- `int = obj.GetWriteBackImages ()` - If on (the default), the result of any image space manipulations are written back to the render window frame buffer. If off, the image stored in the frame buffer may not be correct. Either way, the correct frame buffer images may be read with vtkParallelRenderManager::GetPixelData. Turning WriteBackImages off may result in a speedup if the render window is not visible to the user and images are read back for further processing or transit.

- `obj.WriteBackImagesOn ()` - If on (the default), the result of any image space manipulations are written back to the render window frame buffer. If off, the image stored in the frame buffer may not be correct. Either way, the correct frame buffer images may be read with vtkParallelRenderManager::GetPixelData. Turning WriteBackImages off may result in a speedup if the render window is not visible to the user and images are read back for further processing or transit.

- `obj.WriteBackImagesOff ()` - If on (the default), the result of any image space manipulations are written back to the render window frame buffer. If off, the image stored in the frame buffer may not be correct. Either way, the correct frame buffer images may be read with vtkParallelRenderManager::GetPixelData. Turning WriteBackImages off may result in a speedup if the render window is not visible to the user and images are read back for further processing or transit.

- `obj.SetMagnifyImages (int )` - If on (the default), when the ImageReductionFactor is greater than 1 and WriteBackImages is on, the image will be magnified to fill the entire render window.

- `int = obj.GetMagnifyImages ()` - If on (the default), when the ImageReductionFactor is greater than 1 and WriteBackImages is on, the image will be magnified to fill the entire render window.

- `obj.MagnifyImagesOn ()` - If on (the default), when the ImageReductionFactor is greater than 1 and WriteBackImages is on, the image will be magnified to fill the entire render window.
- `obj.MagnifyImagesOff ()` - If on (the default), when the ImageReductionFactor is greater than 1 and WriteBackImages is on, the image will be magnified to fill the entire render window.
- `obj.SetMagnifyImageMethod (int method)` - Sets the method used to magnify images. Nearest simply replicates each pixel enough times to fill the image. Linear performs linear interpolation between the pixels.
- `int = obj.GetMagnifyImageMethod ()` - Sets the method used to magnify images. Nearest simply replicates each pixel enough times to fill the image. Linear performs linear interpolation between the pixels.
- `obj.SetMagnifyImageMethodToNearest ()` - Sets the method used to magnify images. Nearest simply replicates each pixel enough times to fill the image. Linear performs linear interpolation between the pixels.
- `obj.SetMagnifyImageMethodToLinear ()` - Convenience functions for magnifying images.
- `obj.MagnifyImage (vtkUnsignedCharArray fullImage, int fullImageSize[2], vtkUnsignedCharArray reduce` - Convenience functions for magnifying images.
- `obj.GetPixelData (vtkUnsignedCharArray data)` - The most appropriate way to retrieve full size image data after a render. Will work regardless of whether WriteBackImages or MagnifyImage is on or off. The data returned may be a shallow copy of an internal array. Therefore, the data may be invalid after the next render or if the ParallelRenderManager is destroyed.
- `obj.GetPixelData (int x1, int y1, int x2, int y2, vtkUnsignedCharArray data)` - The most appropriate way to retrieve full size image data after a render. Will work regardless of whether WriteBackImages or MagnifyImage is on or off. The data returned may be a shallow copy of an internal array. Therefore, the data may be invalid after the next render or if the ParallelRenderManager is destroyed.
- `obj.GetReducedPixelData (vtkUnsignedCharArray data)` - The most appropriate way to retrieve reduced size image data after a render. Will work regardless of whether WriteBackImages or MagnifyImage is on or off. The data returned may be a shallow copy of an internal array. Therefore, the data may be invalid after the next render or if the ParallelRenderManager is destroyed.
- `obj.GetReducedPixelData (int x1, int y1, int x2, int y2, vtkUnsignedCharArray data)` - The most appropriate way to retrieve reduced size image data after a render. Will work regardless of whether WriteBackImages or MagnifyImage is on or off. The data returned may be a shallow copy of an internal array. Therefore, the data may be invalid after the next render or if the ParallelRenderManager is destroyed.
- `int = obj. GetFullImageSize ()` - Returns the full image size calculated at the last render.
- `int = obj. GetReducedImageSize ()` - Returns the reduced image size calculated at the last render.
- `obj.TileWindows (int xsize, int ysize, int nColumns)` - Given the x and y size of the render windows, reposition them in a tile of n columns.
- `obj.SetUseRGBA (int )` - Get/Set if all Images must use RGBA instead of RGB. By default, this flag is on.
- `int = obj.GetUseRGBA ()` - Get/Set if all Images must use RGBA instead of RGB. By default, this flag is on.
- `obj.SetForceRenderWindowSize (int )` - If ForceRenderWindowSize is set to true, the render manager will use the RenderWindowSize ivar instead of getting the size from the render window.

- `int = obj.GetForceRenderWindowSize ()` - If ForceRenderWindowSize is set to true, the render manager will use the RenderWindowSize ivar instead of getting the size from the render window.
- `obj.SetForcedRenderWindowSize (int , int )` - If ForceRenderWindowSize is set to true, the render manager will use the Size ivar instead of getting the size from the render window.
- `obj.SetForcedRenderWindowSize (int a[2])` - If ForceRenderWindowSize is set to true, the render manager will use the Size ivar instead of getting the size from the render window.
- `int = obj. GetForcedRenderWindowSize ()` - If ForceRenderWindowSize is set to true, the render manager will use the Size ivar instead of getting the size from the render window.
- `obj.StartService ()` - @deprecated Replaced by vtkParallelRenderManager::StartServices() as of VTK 5.0.
- `obj.SetUseBackBuffer (int )`
- `int = obj.GetUseBackBuffer ()`
- `obj.UseBackBufferOn ()`
- `obj.UseBackBufferOff ()`
- `obj.SetSynchronizeTileProperties (int )` - When set the render manager will synchronize the TileViewport and TileScale properties. This may not be desirable in cases where there's some other mechanism to set the tile dimensions eg. Tile displays.
- `int = obj.GetSynchronizeTileProperties ()` - When set the render manager will synchronize the TileViewport and TileScale properties. This may not be desirable in cases where there's some other mechanism to set the tile dimensions eg. Tile displays.
- `obj.SynchronizeTilePropertiesOn ()` - When set the render manager will synchronize the TileViewport and TileScale properties. This may not be desirable in cases where there's some other mechanism to set the tile dimensions eg. Tile displays.
- `obj.SynchronizeTilePropertiesOff ()` - When set the render manager will synchronize the TileViewport and TileScale properties. This may not be desirable in cases where there's some other mechanism to set the tile dimensions eg. Tile displays.
- `obj.GenericStartRenderCallback ()` - INTERNAL METHODS (DON NOT USE). There are internal methods made public so that they can be called from callback functions.
- `obj.GenericEndRenderCallback ()` - INTERNAL METHODS (DON NOT USE). There are internal methods made public so that they can be called from callback functions.

38.27 vtkPassThroughFilter

38.27.1 Usage

This filter shallow copies it's input to it's output. It is normally used by PVSources with multiple outputs as the VTK filter in the dummy connection objects at each output.

To create an instance of class vtkPassThroughFilter, simply invoke its constructor as follows

```
obj = vtkPassThroughFilter
```

38.27.2 Methods

The class vtkPassThroughFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPassThroughFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPassThroughFilter = obj.NewInstance ()`
- `vtkPassThroughFilter = obj.SafeDownCast (vtkObject o)`

38.28 vtkPCellDataToPointData

38.28.1 Usage

Like it super class, this filter averages the cell data around a point to get new point data. This subclass requests a layer of ghost cells to make the results invariant to pieces. There is a "PieceInvariant" flag that lets the user change the behavior of the filter to that of its superclass.

To create an instance of class vtkPCellDataToPointData, simply invoke its constructor as follows

```
obj = vtkPCellDataToPointData
```

38.28.2 Methods

The class vtkPCellDataToPointData has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPCellDataToPointData class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPCellDataToPointData = obj.NewInstance ()`
- `vtkPCellDataToPointData = obj.SafeDownCast (vtkObject o)`
- `obj.SetPieceInvariant (int )` - To get piece invariance, this filter has to request an extra ghost level. By default piece invariance is on.
- `int = obj.GetPieceInvariant ()` - To get piece invariance, this filter has to request an extra ghost level. By default piece invariance is on.
- `obj.PieceInvariantOn ()` - To get piece invariance, this filter has to request an extra ghost level. By default piece invariance is on.
- `obj.PieceInvariantOff ()` - To get piece invariance, this filter has to request an extra ghost level. By default piece invariance is on.

38.29 vtkPChacoReader

38.29.1 Usage

vtkPChacoReader is a unstructured grid source object that reads Chaco files. The file is read by process 0 and converted into a vtkUnstructuredGrid. The vtkDistributedDataFilter is invoked to divide the grid among the processes.

To create an instance of class vtkPChacoReader, simply invoke its constructor as follows

```
obj = vtkPChacoReader
```

38.29.2 Methods

The class vtkPChacoReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPChacoReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPChacoReader = obj.NewInstance ()`
- `vtkPChacoReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController c)`
- `vtkMultiProcessController = obj.GetController ()`

38.30 vtkPCosmoHaloFinder

38.30.1 Usage

vtkPCosmoHaloFinder is a filter object that operates on the unstructured grid of all particles and assigns each particle a halo id.

To create an instance of class vtkPCosmoHaloFinder, simply invoke its constructor as follows

```
obj = vtkPCosmoHaloFinder
```

38.30.2 Methods

The class vtkPCosmoHaloFinder has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPCosmoHaloFinder class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPCosmoHaloFinder = obj.NewInstance ()`
- `vtkPCosmoHaloFinder = obj.SafeDownCast (vtkObject o)`
- `vtkMultiProcessController = obj.GetController ()` - Set the communicator object for interprocess communication
- `obj.SetController (vtkMultiProcessController )` - Set the communicator object for interprocess communication

- `obj.SetNP (int )` - Specify the number of seeded particles in one dimension (total = np$\hat{3}$)
- `int = obj.GetNP ()` - Specify the number of seeded particles in one dimension (total = np$\hat{3}$)
- `obj.SetRL (float )` - Specify the physical box dimensions size (rL) (default 91)
- `float = obj.GetRL ()` - Specify the physical box dimensions size (rL) (default 91)
- `obj.SetOverlap (float )` - Specify the ghost cell spacing (edge boundary of box) (default 5)
- `float = obj.GetOverlap ()` - Specify the ghost cell spacing (edge boundary of box) (default 5)
- `obj.SetPMin (int )` - Specify the minimum number of particles for a halo (pmin)
- `int = obj.GetPMin ()` - Specify the minimum number of particles for a halo (pmin)
- `obj.SetBB (float )` - Specify the linking length (bb)
- `float = obj.GetBB ()` - Specify the linking length (bb)
- `obj.SetParticleMass (float )` - Specify the particle mass
- `float = obj.GetParticleMass ()` - Specify the particle mass
- `obj.SetCopyHaloDataToParticles (int )` - Copy the halo information to the original particles (Default on)
- `int = obj.GetCopyHaloDataToParticles ()` - Copy the halo information to the original particles (Default on)

38.31 vtkPCosmoReader

38.31.1 Usage

vtkPCosmoReader creates a vtkUnstructuredGrid from a binary cosmology file. The file contains fields for: x_position, x_velocity (float) y_position, y_velocity (float) z-position, z_velocity (float) mass (float) identification tag (integer)

If the file contains particle information x,y,z is the location of the particle in simulation space with a velocity vector and a mass which will be the same for all particles.

To create an instance of class vtkPCosmoReader, simply invoke its constructor as follows

```
obj = vtkPCosmoReader
```

38.31.2 Methods

The class vtkPCosmoReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPCosmoReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPCosmoReader = obj.NewInstance ()`
- `vtkPCosmoReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify the name of the cosmology particle binary file to read
- `string = obj.GetFileName ()` - Specify the name of the cosmology particle binary file to read

- `obj.SetRL (float )` - Specify the physical box dimensions size (rL) (default 91)
- `float = obj.GetRL ()` - Specify the physical box dimensions size (rL) (default 91)
- `obj.SetOverlap (float )` - Specify the ghost cell spacing (edge boundary of box) (default 5)
- `float = obj.GetOverlap ()` - Specify the ghost cell spacing (edge boundary of box) (default 5)
- `obj.SetReadMode (int )` - Set the read mode (0 = one-to-one, 1 = default, round-robin)
- `int = obj.GetReadMode ()` - Set the read mode (0 = one-to-one, 1 = default, round-robin)
- `obj.SetCosmoFormat (int )` - Set the filetype to Gadget or Cosmo read mode (0 = Gadget, 1 = default, Cosmo)
- `int = obj.GetCosmoFormat ()` - Set the filetype to Gadget or Cosmo read mode (0 = Gadget, 1 = default, Cosmo)
- `vtkMultiProcessController = obj.GetController ()` - Set the communicator object for interprocess communication
- `obj.SetController (vtkMultiProcessController )` - Set the communicator object for interprocess communication

38.32 vtkPDataSetReader

38.32.1 Usage

vtkPDataSetReader will read a piece of a file, it takes as input a metadata file that lists all of the files in a data set.

To create an instance of class vtkPDataSetReader, simply invoke its constructor as follows

```
obj = vtkPDataSetReader
```

38.32.2 Methods

The class vtkPDataSetReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPDataSetReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPDataSetReader = obj.NewInstance ()`
- `vtkPDataSetReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - This file to open and read.
- `string = obj.GetFileName ()` - This file to open and read.
- `int = obj.GetDataType ()` - This is set when UpdateInformation is called. It shows the type of the output.
- `int = obj.CanReadFile (string filename)` - Called to determine if the file can be read by the reader.

38.33 vtkPDataSetWriter

38.33.1 Usage

vtkPDataSetWriter will write a piece of a file, and will also create a metadata file that lists all of the files in a data set.

To create an instance of class vtkPDataSetWriter, simply invoke its constructor as follows

```
obj = vtkPDataSetWriter
```

38.33.2 Methods

The class vtkPDataSetWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPDataSetWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPDataSetWriter = obj.NewInstance ()`
- `vtkPDataSetWriter = obj.SafeDownCast (vtkObject o)`
- `int = obj.Write ()` - Write the pvtk file and cooresponding vtk files.
- `obj.SetNumberOfPieces (int num)` - This is how many pieces the whole data set will be divided into.
- `int = obj.GetNumberOfPieces ()` - This is how many pieces the whole data set will be divided into.
- `obj.SetGhostLevel (int )` - Extra ghost cells will be written out to each piece file if this value is larger than 0.
- `int = obj.GetGhostLevel ()` - Extra ghost cells will be written out to each piece file if this value is larger than 0.
- `obj.SetStartPiece (int )` - This is the range of pieces that that this writer is responsible for writing. All pieces must be written by some process. The process that writes piece 0 also writes the pvtk file that lists all the piece file names.
- `int = obj.GetStartPiece ()` - This is the range of pieces that that this writer is responsible for writing. All pieces must be written by some process. The process that writes piece 0 also writes the pvtk file that lists all the piece file names.
- `obj.SetEndPiece (int )` - This is the range of pieces that that this writer is responsible for writing. All pieces must be written by some process. The process that writes piece 0 also writes the pvtk file that lists all the piece file names.
- `int = obj.GetEndPiece ()` - This is the range of pieces that that this writer is responsible for writing. All pieces must be written by some process. The process that writes piece 0 also writes the pvtk file that lists all the piece file names.
- `obj.SetFilePattern (string )` - This file pattern uses the file name and piece number to contruct a file name for the piece file.
- `string = obj.GetFilePattern ()` - This file pattern uses the file name and piece number to contruct a file name for the piece file.

- `obj.SetUseRelativeFileNames (int )` - This flag determines whether to use absolute paths for the piece files. By default the pieces are put in the main directory, and the piece file names in the meta data pvtk file are relative to this directory. This should make moving the whole lot to another directory, an easier task.

- `int = obj.GetUseRelativeFileNames ()` - This flag determines whether to use absolute paths for the piece files. By default the pieces are put in the main directory, and the piece file names in the meta data pvtk file are relative to this directory. This should make moving the whole lot to another directory, an easier task.

- `obj.UseRelativeFileNamesOn ()` - This flag determines whether to use absolute paths for the piece files. By default the pieces are put in the main directory, and the piece file names in the meta data pvtk file are relative to this directory. This should make moving the whole lot to another directory, an easier task.

- `obj.UseRelativeFileNamesOff ()` - This flag determines whether to use absolute paths for the piece files. By default the pieces are put in the main directory, and the piece file names in the meta data pvtk file are relative to this directory. This should make moving the whole lot to another directory, an easier task.

38.34 vtkPExtractArraysOverTime

38.34.1 Usage

vtkPExtractArraysOverTime is a parallelized version of vtkExtractArraysOverTime. vtkExtractArraysOverTime extract point or cell data given a selection. For every cell or point extracted, vtkExtractArraysOverTime create a vtkTable that is placed in an appropriately named block in an output multi-block dataset. For global-id based selections or location based selections, it's possible that over time the cell/point moves across processes. This filter ensures that such extractions spread across processes are combined correctly into a single vtkTable. This filter produces a valid output on the root node alone, all other nodes, simply have empty multi-block dataset with number of blocks matching the root (to ensure that all processes have the same structure).

To create an instance of class vtkPExtractArraysOverTime, simply invoke its constructor as follows

```
obj = vtkPExtractArraysOverTime
```

38.34.2 Methods

The class vtkPExtractArraysOverTime has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPExtractArraysOverTime class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkPExtractArraysOverTime = obj.NewInstance ()`

- `vtkPExtractArraysOverTime = obj.SafeDownCast (vtkObject o)`

- `obj.SetController (vtkMultiProcessController )` - Set and get the controller.

- `vtkMultiProcessController = obj.GetController ()` - Set and get the controller.

38.35 vtkPieceRequestFilter

38.35.1 Usage

Sends the piece and number of pieces to upstream filters; passes the input to the output unmodified.

To create an instance of class vtkPieceRequestFilter, simply invoke its constructor as follows

```
obj = vtkPieceRequestFilter
```

38.35.2 Methods

The class vtkPieceRequestFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPieceRequestFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPieceRequestFilter = obj.NewInstance ()`
- `vtkPieceRequestFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfPieces (int )` - The total number of pieces.
- `int = obj.GetNumberOfPiecesMinValue ()` - The total number of pieces.
- `int = obj.GetNumberOfPiecesMaxValue ()` - The total number of pieces.
- `int = obj.GetNumberOfPieces ()` - The total number of pieces.
- `obj.SetPiece (int )` - The piece to extract.
- `int = obj.GetPieceMinValue ()` - The piece to extract.
- `int = obj.GetPieceMaxValue ()` - The piece to extract.
- `int = obj.GetPiece ()` - The piece to extract.
- `vtkDataObject = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm.

38.36 vtkPieceScalars

38.36.1 Usage

vtkPieceScalars is meant to display which piece is being requested as scalar values. It is usefull for visualizing the partitioning for streaming or distributed pipelines.

To create an instance of class vtkPieceScalars, simply invoke its constructor as follows

```
obj = vtkPieceScalars
```

38.36.2 Methods

The class vtkPieceScalars has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPieceScalars class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPieceScalars = obj.NewInstance ()`
- `vtkPieceScalars = obj.SafeDownCast (vtkObject o)`
- `obj.SetScalarModeToCellData ()` - Option to centerate cell scalars of points scalars. Default is point scalars.
- `obj.SetScalarModeToPointData ()` - Option to centerate cell scalars of points scalars. Default is point scalars.
- `int = obj.GetScalarMode ()`
- `obj.SetRandomMode (int )`
- `int = obj.GetRandomMode ()`
- `obj.RandomModeOn ()`
- `obj.RandomModeOff ()`

38.37 vtkPImageWriter

38.37.1 Usage

vtkPImageWriter writes images to files with any data type. The data type of the file is the same scalar type as the input. The dimensionality determines whether the data will be written in one or multiple files. This class is used as the superclass of most image writing classes such as vtkBMPWriter etc. It supports streaming.

To create an instance of class vtkPImageWriter, simply invoke its constructor as follows

```
obj = vtkPImageWriter
```

38.37.2 Methods

The class vtkPImageWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPImageWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPImageWriter = obj.NewInstance ()`
- `vtkPImageWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetMemoryLimit (long )` - Set / Get the memory limit in kilobytes. The writer will stream to attempt to keep the pipeline size within this limit
- `long = obj.GetMemoryLimit ()` - Set / Get the memory limit in kilobytes. The writer will stream to attempt to keep the pipeline size within this limit

38.38 vtkPKdTree

38.38.1 Usage

Build, in parallel, a k-d tree decomposition of one or more vtkDataSets distributed across processors. We assume each process has read in one portion of a large distributed data set. When done, each process has access to the k-d tree structure, can obtain information about which process contains data for each spatial region, and can depth sort the spatial regions.

This class can also assign spatial regions to processors, based on one of several region assignment schemes. By default a contiguous, convex region is assigned to each process. Several queries return information about how many and what cells I have that lie in a region assigned to another process.

To create an instance of class vtkPKdTree, simply invoke its constructor as follows

```
obj = vtkPKdTree
```

38.38.2 Methods

The class vtkPKdTree has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPKdTree class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPKdTree = obj.NewInstance ()`
- `vtkPKdTree = obj.SafeDownCast (vtkObject o)`
- `obj.BuildLocator ()` - Build the spatial decomposition. Call this explicitly after changing any parameters affecting the build of the tree. It must be called by all processes in the parallel application, or it will hang.
- `vtkIdType = obj.GetTotalNumberOfCells ()` - Create tables of counts of cells per process per region. These tables can be accessed with queries like "HasData", "GetProcessCellCountForRegion", and so on. You must have called BuildLocator() beforehand. This method must be called by all processes or it will hang. Returns 1 on error, 0 when no error.
- `int = obj.CreateProcessCellCountData ()` - Create tables of counts of cells per process per region. These tables can be accessed with queries like "HasData", "GetProcessCellCountForRegion", and so on. You must have called BuildLocator() beforehand. This method must be called by all processes or it will hang. Returns 1 on error, 0 when no error.
- `int = obj.CreateGlobalDataArrayBounds ()` - A convenience function which compiles the global bounds of the data arrays across processes. These bounds can be accessed with "GetCellArrayGlobalRange" and "GetPointArrayGlobalRange". This method must be called by all processes or it will hang. Returns 1 on error, 0 when no error.
- `obj.SetController (vtkMultiProcessController c)` - Set/Get the communicator object
- `vtkMultiProcessController = obj.GetController ()` - Set/Get the communicator object
- `int = obj.GetRegionAssignment ()` - The PKdTree class can assign spatial regions to processors after building the k-d tree, using one of several partitioning criteria. These functions Set/Get whether this assignment is computed. The default is "Off", no assignment is computed. If "On", and no assignment scheme is specified, contiguous assignment will be computed. Specifying an assignment scheme (with AssignRegions*()) automatically turns on RegionAssignment.

- `int = obj.AssignRegions (int map, int numRegions)` - Assign spatial regions to processes via a user defined map. The user-supplied map is indexed by region ID, and provides a process ID for each region.
- `int = obj.AssignRegionsRoundRobin ()` - Let the PKdTree class assign a process to each region in a round robin fashion. If the k-d tree has not yet been built, the regions will be assigned after BuildLocator executes.
- `int = obj.AssignRegionsContiguous ()` - Let the PKdTree class assign a process to each region by assigning contiguous sets of spatial regions to each process. The set of regions assigned to each process will always have a union that is a convex space (a box). If the k-d tree has not yet been built, the regions will be assigned after BuildLocator executes.
- `int = obj.GetRegionAssignmentList (int procId, vtkIntArray list)` - Writes the list of region IDs assigned to the specified process. Regions IDs start at 0 and increase by 1 from there. Returns the number of regions in the list.
- `obj.GetAllProcessesBorderingOnPoint (float x, float y, float z, vtkIntArray list)` - The k-d tree spatial regions have been assigned to processes. Given a point on the boundary of one of the regions, this method creates a list of all processes whose region boundaries include that point. This may be required when looking for processes that have cells adjacent to the cells of a given process.
- `int = obj.GetProcessAssignedToRegion (int regionId)` - Returns the ID of the process assigned to the region.
- `int = obj.HasData (int processId, int regionId)` - Returns 1 if the process has data for the given region, 0 otherwise.
- `int = obj.GetProcessCellCountForRegion (int processId, int regionId)` - Returns the number of cells the specified process has in the specified region.
- `int = obj.GetTotalProcessesInRegion (int regionId)` - Returns the total number of processes that have data falling within this spatial region.
- `int = obj.GetProcessListForRegion (int regionId, vtkIntArray processes)` - Adds the list of processes having data for the given region to the supplied list, returns the number of processes added.
- `int = obj.GetProcessesCellCountForRegion (int regionId, int count, int len)` - Writes the number of cells each process has for the region to the supplied list of length len. Returns the number of cell counts written. The order of the cell counts corresponds to the order of process IDs in the process list returned by GetProcessListForRegion.
- `int = obj.GetTotalRegionsForProcess (int processId)` - Returns the total number of spatial regions that a given process has data for.
- `int = obj.GetRegionListForProcess (int processId, vtkIntArray regions)` - Adds the region IDs for which this process has data to the supplied vtkIntArray. Retruns the number of regions.
- `int = obj.GetRegionsCellCountForProcess (int ProcessId, int count, int len)` - Writes to the supplied integer array the number of cells this process has for each region. Returns the number of cell counts written. The order of the cell counts corresponds to the order of region IDs in the region list returned by GetRegionListForProcess.
- `vtkIdType = obj.GetCellListsForProcessRegions (int ProcessId, int set, vtkIdList inRegionCells, vtk` - After regions have been assigned to processes, I may want to know which cells I have that are in the regions assigned to a particular process.

  This method takes a process ID and two vtkIdLists. It writes to the first list the IDs of the cells contained in the process' regions. (That is, their cell centroid is contained in the region.) To the

second list it write the IDs of the cells which intersect the process' regions but whose cell centroid lies elsewhere.

The total number of cell IDs written to both lists is returned. Either list pointer passed in can be NULL, and it will be ignored. If there are multiple data sets, you must specify which data set you wish cell IDs for.

The caller should delete these two lists when done. This method uses the cell lists created in vtkKdTree::CreateCellLists(). If the cell lists for the process' regions do not exist, this method will first build the cell lists for all regions by calling CreateCellLists(). You must remember to DeleteCellLists() when done with all calls to this method, as cell lists can require a great deal of memory.

- `vtkIdType = obj.GetCellListsForProcessRegions (int ProcessId, vtkDataSet set, vtkIdList inRegionCel`
  - After regions have been assigned to processes, I may want to know which cells I have that are in the regions assigned to a particular process.

  This method takes a process ID and two vtkIdLists. It writes to the first list the IDs of the cells contained in the process' regions. (That is, their cell centroid is contained in the region.) To the second list it write the IDs of the cells which intersect the process' regions but whose cell centroid lies elsewhere.

  The total number of cell IDs written to both lists is returned. Either list pointer passed in can be NULL, and it will be ignored. If there are multiple data sets, you must specify which data set you wish cell IDs for.

  The caller should delete these two lists when done. This method uses the cell lists created in vtkKdTree::CreateCellLists(). If the cell lists for the process' regions do not exist, this method will first build the cell lists for all regions by calling CreateCellLists(). You must remember to DeleteCellLists() when done with all calls to this method, as cell lists can require a great deal of memory.

- `vtkIdType = obj.GetCellListsForProcessRegions (int ProcessId, vtkIdList inRegionCells, vtkIdList or`
  - After regions have been assigned to processes, I may want to know which cells I have that are in the regions assigned to a particular process.

  This method takes a process ID and two vtkIdLists. It writes to the first list the IDs of the cells contained in the process' regions. (That is, their cell centroid is contained in the region.) To the second list it write the IDs of the cells which intersect the process' regions but whose cell centroid lies elsewhere.

  The total number of cell IDs written to both lists is returned. Either list pointer passed in can be NULL, and it will be ignored. If there are multiple data sets, you must specify which data set you wish cell IDs for.

  The caller should delete these two lists when done. This method uses the cell lists created in vtkKdTree::CreateCellLists(). If the cell lists for the process' regions do not exist, this method will first build the cell lists for all regions by calling CreateCellLists(). You must remember to DeleteCellLists() when done with all calls to this method, as cell lists can require a great deal of memory.

- `int = obj.DepthOrderAllProcesses (double directionOfProjection, vtkIntArray orderedList)`
  - DO NOT CALL. Deprecated in VTK 5.2. Use ViewOrderAllProcessesInDirection or ViewOrderAllProcessesFromPosition.

- `int = obj.ViewOrderAllProcessesInDirection (double directionOfProjection[3], vtkIntArray orderedLis`
  - Return a list of all processes in order from front to back given a vector direction of projection. Use this to do visibility sorts in parallel projection mode. 'orderedList' will be resized to the number of processes. The return value is the number of processes.

- `int = obj.ViewOrderAllProcessesFromPosition (double cameraPosition[3], vtkIntArray orderedList)`
  - Return a list of all processes in order from front to back given a camera position. Use this to do visibility sorts in perspective projection mode. 'orderedList' will be resized to the number of processes. The return value is the number of processes.

- `int = obj.GetCellArrayGlobalRange (string name, float range[2])`
- `int = obj.GetPointArrayGlobalRange (string name, float range[2])`
- `int = obj.GetCellArrayGlobalRange (string name, double range[2])`
- `int = obj.GetPointArrayGlobalRange (string name, double range[2])`
- `int = obj.GetCellArrayGlobalRange (int arrayIndex, double range[2])`
- `int = obj.GetPointArrayGlobalRange (int arrayIndex, double range[2])`
- `int = obj.GetCellArrayGlobalRange (int arrayIndex, float range[2])`
- `int = obj.GetPointArrayGlobalRange (int arrayIndex, float range[2])`

38.39 vtkPLinearExtrusionFilter

38.39.1 Usage

vtkPLinearExtrusionFilter is a parallel version of vtkLinearExtrusionFilter.

To create an instance of class vtkPLinearExtrusionFilter, simply invoke its constructor as follows

```
obj = vtkPLinearExtrusionFilter
```

38.39.2 Methods

The class vtkPLinearExtrusionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPLinearExtrusionFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPLinearExtrusionFilter = obj.NewInstance ()`
- `vtkPLinearExtrusionFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetPieceInvariant (int )`
- `int = obj.GetPieceInvariant ()`
- `obj.PieceInvariantOn ()`
- `obj.PieceInvariantOff ()`

38.40 vtkPNrrdReader

38.40.1 Usage

vtkPNrrdReader is a subclass of vtkMPIImageReader that will read Nrrd format header information of the image before reading the data. This means that the reader will automatically set information like file dimensions.

.SECTION Bugs

There are several limitations on what type of nrrd files we can read. This reader only supports nrrd files in raw format. Other encodings like ascii and hex will result in errors. When reading in detached headers, this only supports reading one file that is detached.

To create an instance of class vtkPNrrdReader, simply invoke its constructor as follows

```
obj = vtkPNrrdReader
```

38.40.2 Methods

The class vtkPNrrdReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPNrrdReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPNrrdReader = obj.NewInstance ()`
- `vtkPNrrdReader = obj.SafeDownCast (vtkObject o)`
- `int = obj.CanReadFile (string filename)`

38.41 vtkPOpenFOAMReader

38.41.1 Usage

vtkPOpenFOAMReader creates a multiblock dataset. It reads parallel-decomposed mesh information and time dependent data. The polyMesh folders contain mesh information. The time folders contain transient data for the cells. Each folder can contain any number of data files.

To create an instance of class vtkPOpenFOAMReader, simply invoke its constructor as follows

```
obj = vtkPOpenFOAMReader
```

38.41.2 Methods

The class vtkPOpenFOAMReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPOpenFOAMReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPOpenFOAMReader = obj.NewInstance ()`
- `vtkPOpenFOAMReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetCaseType (int t)` - Set and get case type. 0 = decomposed case, 1 = reconstructed case.
- `obj.SetController (vtkMultiProcessController )` - Set and get the controller.
- `vtkMultiProcessController = obj.GetController ()` - Set and get the controller.

38.42 vtkPOPReader

38.42.1 Usage

vtkPOPReader Just converts from images to a structured grid for now.

To create an instance of class vtkPOPReader, simply invoke its constructor as follows

```
obj = vtkPOPReader
```

38.42.2 Methods

The class vtkPOPReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPOPReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPOPReader = obj.NewInstance ()`
- `vtkPOPReader = obj.SafeDownCast (vtkObject o)`
- `int = obj. GetDimensions ()` - This is the longitude and latitude dimensions of the structured grid.
- `string = obj.GetGridFileName ()` - This file contains the latitude and longitude of the grid. It must be double with no header.
- `string = obj.GetUFlowFileName ()` - These files contains the u and v components of the flow.
- `string = obj.GetVFlowFileName ()` - These files contains the u and v components of the flow.
- `obj.SetFileName (string )` - This file contains information about all the files.
- `string = obj.GetFileName ()` - This file contains information about all the files.
- `obj.SetRadius (double )` - Radius of the earth.
- `double = obj.GetRadius ()` - Radius of the earth.
- `obj.SetClipExtent (int , int , int , int , int , int )` - Because the data can be so large, here is an option to clip while reading.
- `obj.SetClipExtent (int  a[6])` - Because the data can be so large, here is an option to clip while reading.
- `int = obj. GetClipExtent ()` - Because the data can be so large, here is an option to clip while reading.
- `obj.SetNumberOfGhostLevels (int )` - Set the number of ghost levels to include in the data
- `int = obj.GetNumberOfGhostLevels ()` - Set the number of ghost levels to include in the data

38.43 vtkPOutlineCornerFilter

38.43.1 Usage

vtkPOutlineCornerFilter works like vtkOutlineCornerFilter, but it looks for data partitions in other processes. It assumes the filter is operated in a data parallel pipeline.

To create an instance of class vtkPOutlineCornerFilter, simply invoke its constructor as follows

```
obj = vtkPOutlineCornerFilter
```

38.43.2 Methods

The class vtkPOutlineCornerFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPOutlineCornerFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPOutlineCornerFilter = obj.NewInstance ()`
- `vtkPOutlineCornerFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetCornerFactor (double )` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `double = obj.GetCornerFactorMinValue ()` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `double = obj.GetCornerFactorMaxValue ()` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `double = obj.GetCornerFactor ()` - Set/Get the factor that controls the relative size of the corners to the length of the corresponding bounds
- `obj.SetController (vtkMultiProcessController )` - Set and get the controller.
- `vtkMultiProcessController = obj.GetController ()` - Set and get the controller.

38.44 vtkPOutlineFilter

38.44.1 Usage

vtkPOutlineFilter works like vtkOutlineFilter, but it looks for data partitions in other processes. It assumes the filter is operated in a data parallel pipeline.

To create an instance of class vtkPOutlineFilter, simply invoke its constructor as follows

```
obj = vtkPOutlineFilter
```

38.44.2 Methods

The class vtkPOutlineFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPOutlineFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPOutlineFilter = obj.NewInstance ()`
- `vtkPOutlineFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Set and get the controller.
- `vtkMultiProcessController = obj.GetController ()` - Set and get the controller.

38.45 vtkPPolyDataNormals

38.45.1 Usage

To create an instance of class vtkPPolyDataNormals, simply invoke its constructor as follows

```
obj = vtkPPolyDataNormals
```

38.45.2 Methods

The class vtkPPolyDataNormals has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPPolyDataNormals class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPPolyDataNormals = obj.NewInstance ()`
- `vtkPPolyDataNormals = obj.SafeDownCast (vtkObject o)`
- `obj.SetPieceInvariant (int )` - To get piece invariance, this filter has to request an extra ghost level. By default piece invariance is on.
- `int = obj.GetPieceInvariant ()` - To get piece invariance, this filter has to request an extra ghost level. By default piece invariance is on.
- `obj.PieceInvariantOn ()` - To get piece invariance, this filter has to request an extra ghost level. By default piece invariance is on.
- `obj.PieceInvariantOff ()` - To get piece invariance, this filter has to request an extra ghost level. By default piece invariance is on.

38.46 vtkPProbeFilter

38.46.1 Usage

To create an instance of class vtkPProbeFilter, simply invoke its constructor as follows

```
obj = vtkPProbeFilter
```

38.46.2 Methods

The class vtkPProbeFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPProbeFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPProbeFilter = obj.NewInstance ()`
- `vtkPProbeFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Set and get the controller.
- `vtkMultiProcessController = obj.GetController ()` - Set and get the controller.

38.47 vtkPReflectionFilter

38.47.1 Usage

vtkPReflectionFilter is a parallel version of vtkReflectionFilter which takes into consideration the full dataset bounds for performing the reflection.

To create an instance of class vtkPReflectionFilter, simply invoke its constructor as follows

```
obj = vtkPReflectionFilter
```

38.47.2 Methods

The class vtkPReflectionFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPReflectionFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPReflectionFilter = obj.NewInstance ()`
- `vtkPReflectionFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Get/Set the parallel controller.
- `vtkMultiProcessController = obj.GetController ()` - Get/Set the parallel controller.

38.48 vtkProcess

38.48.1 Usage

vtkProcess is an abstract class representing a process that can be launched by a vtkMultiProcessController. Concrete classes just have to implement Execute() method and make sure it set the proper value in ReturnValue.

.SECTION Example class MyProcess: public vtkProcess ... vtkMultiProcessController *c; MyProcess *p=new MyProcess::New(); p-¿SetArgs(argc,argv); // some parameters specific to the process p-¿SetX(10.0); // ... c-¿SetSingleProcess(p); c-¿SingleMethodExecute(); int returnValue=p-¿GetReturnValue();

To create an instance of class vtkProcess, simply invoke its constructor as follows

```
obj = vtkProcess
```

38.48.2 Methods

The class vtkProcess has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProcess class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProcess = obj.NewInstance ()`
- `vtkProcess = obj.SafeDownCast (vtkObject o)`
- `obj.Execute ()`

- `vtkMultiProcessController = obj.GetController ()` - Give access to the controller that launched the process. Initial value is NULL.
- `obj.SetController (vtkMultiProcessController aController)` - This method should not be called directly but set by the controller itself.
- `int = obj.GetReturnValue ()` - Value set at the end of a call to Execute.

38.49 vtkProcessGroup

38.49.1 Usage

This class is used for creating groups of processes. A vtkProcessGroup is initialized by passing the controller or communicator on which the group is based off of. You can then use the group to subset and reorder the the processes. Eventually, you can pass the group object to the CreateSubController method of vtkMultiProcessController to create a controller for the defined group of processes. You must use the same controller (or attached communicator) from which this group was initialized with.

To create an instance of class vtkProcessGroup, simply invoke its constructor as follows

```
obj = vtkProcessGroup
```

38.49.2 Methods

The class vtkProcessGroup has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProcessGroup class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProcessGroup = obj.NewInstance ()`
- `vtkProcessGroup = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (vtkMultiProcessController controller)` - Initialize the group to the given controller or communicator. The group will be set to contain all of the processes in the controller/communicator in the same order.
- `obj.Initialize (vtkCommunicator communicator)` - Initialize the group to the given controller or communicator. The group will be set to contain all of the processes in the controller/communicator in the same order.
- `vtkCommunicator = obj.GetCommunicator ()` - Get the communicator on which this group is based on.
- `obj.SetCommunicator (vtkCommunicator communicator)` - Set the communicator. This has the same effect as Initialize except that the contents of the group will not be modified (although they may be truncated if the new communicator is smaller than the current group). Note that this can lead to an invalid group if there are values in the group that are not valid in the new communicator.
- `int = obj.GetNumberOfProcessIds ()` - Returns the size of this group (the number of processes defined in it).
- `int = obj.GetProcessId (int pos)` - Get the process id for the local process (as defined by the group's communicator). Returns -1 if the local process is not in the group.

- `int = obj.GetLocalProcessId ()` - Get the process id for the local process (as defined by the group's communicator). Returns -1 if the local process is not in the group.
- `int = obj.FindProcessId (int processId)` - Given a process id in the communicator, this method returns its location in the group or -1 if it is not in the group. For example, if this group contains 6, 2, 8, 1, then FindProcessId(2) will return 1 and FindProcessId(3) will return -1.
- `int = obj.AddProcessId (int processId)` - Add a process id to the end of the group (if it is not already in the group). Returns the location where the id was stored.
- `int = obj.RemoveProcessId (int processId)` - Remove the given process id from the group (assuming it is in the group). All ids to the "right" of the removed id are shifted over. Returns 1 if the process id was removed, 0 if the process id was not in the group in the first place.
- `obj.RemoveAllProcessIds ()` - Removes all the processes ids from the group, leaving the group empty.
- `obj.Copy (vtkProcessGroup group)` - Copies the given group's communicator and process ids.

38.50 vtkProcessIdScalars

38.50.1 Usage

vtkProcessIdScalars is meant to display which processor owns which cells and points. It is useful for visualizing the partitioning for streaming or distributed pipelines.

To create an instance of class vtkProcessIdScalars, simply invoke its constructor as follows

```
obj = vtkProcessIdScalars
```

38.50.2 Methods

The class vtkProcessIdScalars has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProcessIdScalars class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProcessIdScalars = obj.NewInstance ()`
- `vtkProcessIdScalars = obj.SafeDownCast (vtkObject o)`
- `obj.SetScalarModeToCellData ()` - Option to centerate cell scalars of points scalars. Default is point scalars.
- `obj.SetScalarModeToPointData ()` - Option to centerate cell scalars of points scalars. Default is point scalars.
- `int = obj.GetScalarMode ()`
- `obj.SetRandomMode (int )`
- `int = obj.GetRandomMode ()`
- `obj.RandomModeOn ()`
- `obj.RandomModeOff ()`

- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.

- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.

38.51 vtkPSLACReader

38.51.1 Usage

Extends the vtkSLACReader to read in partitioned pieces. Due to the nature of the data layout, this reader only works in a data parallel mode where each process in a parallel job simultaneously attempts to read the piece corresponding to the local process id.

To create an instance of class vtkPSLACReader, simply invoke its constructor as follows

```
obj = vtkPSLACReader
```

38.51.2 Methods

The class vtkPSLACReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPSLACReader class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkPSLACReader = obj.NewInstance ()`

- `vtkPSLACReader = obj.SafeDownCast (vtkObject o)`

- `vtkMultiProcessController = obj.GetController ()` - The controller used to communicate partition data. The number of pieces requested must agree with the number of processes, the piece requested must agree with the local process id, and all process must invoke ProcessRequests of this filter simultaneously.

- `obj.SetController (vtkMultiProcessController )` - The controller used to communicate partition data. The number of pieces requested must agree with the number of processes, the piece requested must agree with the local process id, and all process must invoke ProcessRequests of this filter simultaneously.

38.52 vtkPStreamTracer

38.52.1 Usage

This class implements some necessary functionality used by distributed and parallel streamline generators. Note that all processes must have access to the WHOLE seed source, i.e. the source must be identical on all processes.

To create an instance of class vtkPStreamTracer, simply invoke its constructor as follows

```
obj = vtkPStreamTracer
```

38.52.2 Methods

The class vtkPStreamTracer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPStreamTracer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPStreamTracer = obj.NewInstance ()`
- `vtkPStreamTracer = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController controller)` - Set/Get the controller use in compositing (set to the global controller by default) If not using the default, this must be called before any other methods.
- `vtkMultiProcessController = obj.GetController ()` - Set/Get the controller use in compositing (set to the global controller by default) If not using the default, this must be called before any other methods.

38.53 vtkPTableToStructuredGrid

38.53.1 Usage

vtkPTableToStructuredGrid is vtkTableToStructuredGrid specialization which handles distribution of the input table. For starters, this assumes that the input table is only available on the root node.

To create an instance of class vtkPTableToStructuredGrid, simply invoke its constructor as follows

```
obj = vtkPTableToStructuredGrid
```

38.53.2 Methods

The class vtkPTableToStructuredGrid has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPTableToStructuredGrid class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPTableToStructuredGrid = obj.NewInstance ()`
- `vtkPTableToStructuredGrid = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Get/Set the controller.
- `vtkMultiProcessController = obj.GetController ()` - Get/Set the controller.

38.54 vtkRectilinearGridOutlineFilter

38.54.1 Usage

vtkRectilinearGridOutlineFilter works in parallel. There is no reason. to use this filter if you are not breaking the processing into pieces. With one piece you can simply use vtkOutlineFilter. This filter ignores internal edges when the extent is not the whole extent.

To create an instance of class vtkRectilinearGridOutlineFilter, simply invoke its constructor as follows

```
obj = vtkRectilinearGridOutlineFilter
```

38.54.2 Methods

The class vtkRectilinearGridOutlineFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearGridOutlineFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRectilinearGridOutlineFilter = obj.NewInstance ()`
- `vtkRectilinearGridOutlineFilter = obj.SafeDownCast (vtkObject o)`

38.55 vtkSocketCommunicator

38.55.1 Usage

This is a concrete implementation of vtkCommunicator which supports interprocess communication using BSD style sockets. It supports byte swapping for the communication of machines with different endianness.

To create an instance of class vtkSocketCommunicator, simply invoke its constructor as follows

```
obj = vtkSocketCommunicator
```

38.55.2 Methods

The class vtkSocketCommunicator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSocketCommunicator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSocketCommunicator = obj.NewInstance ()`
- `vtkSocketCommunicator = obj.SafeDownCast (vtkObject o)`
- `int = obj.WaitForConnection (int port)` - Wait for connection on a given port. These methods return 1 on success, 0 on error.
- `int = obj.WaitForConnection (vtkServerSocket socket, long msec)` - Wait for connection on a given port. These methods return 1 on success, 0 on error.
- `obj.CloseConnection ()` - Close a connection.
- `int = obj.ConnectTo (string hostName, int port)` - Open a connection to host.
- `int = obj.GetSwapBytesInReceivedData ()` - Returns 1 if bytes must be swapped in received ints, floats, etc
- `int = obj.GetIsConnected ()` - Is the communicator connected?.
- `obj.SetNumberOfProcesses (int num)` - Set the number of processes you will be using.
- `obj.Barrier ()` - This class foolishly breaks the conventions of the superclass, so this overload fixes the method.

- `obj.SetPerformHandshake (int )` - Set or get the PerformHandshake ivar. If it is on, the communicator will try to perform a handshake when connected. It is on by default.
- `int = obj.GetPerformHandshakeMinValue ()` - Set or get the PerformHandshake ivar. If it is on, the communicator will try to perform a handshake when connected. It is on by default.
- `int = obj.GetPerformHandshakeMaxValue ()` - Set or get the PerformHandshake ivar. If it is on, the communicator will try to perform a handshake when connected. It is on by default.
- `obj.PerformHandshakeOn ()` - Set or get the PerformHandshake ivar. If it is on, the communicator will try to perform a handshake when connected. It is on by default.
- `obj.PerformHandshakeOff ()` - Set or get the PerformHandshake ivar. If it is on, the communicator will try to perform a handshake when connected. It is on by default.
- `int = obj.GetPerformHandshake ()` - Set or get the PerformHandshake ivar. If it is on, the communicator will try to perform a handshake when connected. It is on by default.
- `int = obj.LogToFile (string name)` - Log messages to the given file. The file is truncated unless the second argument is non-zero (default is to truncate). If the file name is empty or NULL, logging is disabled. Returns 0 if the file failed to open, and 1 otherwise.
- `int = obj.LogToFile (string name, int append)` - Log messages to the given file. The file is truncated unless the second argument is non-zero (default is to truncate). If the file name is empty or NULL, logging is disabled. Returns 0 if the file failed to open, and 1 otherwise.
- `obj.SetReportErrors (int )` - If ReportErrors if false, all vtkErrorMacros are suppressed.
- `int = obj.GetReportErrors ()` - If ReportErrors if false, all vtkErrorMacros are suppressed.
- `vtkClientSocket = obj.GetSocket ()` - Get/Set the actual socket used for communication.
- `obj.SetSocket (vtkClientSocket )` - Get/Set the actual socket used for communication.
- `int = obj.Handshake ()` - Performs handshake. This uses vtkClientSocket::ConnectingSide to decide whether to perform ServerSideHandshake or ClientSideHandshake.
- `int = obj.ServerSideHandshake ()` - Performs ServerSide handshake. One should preferably use Handshake() which calls ServerSideHandshake or ClientSideHandshake as required.
- `int = obj.ClientSideHandshake ()` - Performs ClientSide handshake. One should preferably use Handshake() which calls ServerSideHandshake or ClientSideHandshake as required.
- `int = obj.GetIsServer ()` - Returns true if this side of the socket is the server. The result is invalid if the socket is not connected.

38.56 vtkSocketController

38.56.1 Usage

This is a concrete implementation of vtkMultiProcessController. It supports one-to-one communication using sockets. Note that process 0 will always correspond to self and process 1 to the remote process. This class is best used with ports.

To create an instance of class vtkSocketController, simply invoke its constructor as follows

```
obj = vtkSocketController
```

38.56.2 Methods

The class vtkSocketController has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSocketController class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSocketController = obj.NewInstance ()`
- `vtkSocketController = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Does not apply to sockets. Does nothing.
- `obj.Finalize ()` - Does not apply to sockets. Does nothing.
- `obj.Finalize (int )` - Does not apply to sockets. Does nothing.
- `obj.SingleMethodExecute ()` - Does not apply to sockets. Does nothing.
- `obj.MultipleMethodExecute ()` - Does not apply to sockets. Does nothing.
- `obj.CreateOutputWindow ()` - Does not apply to sockets. Does nothing.
- `int = obj.WaitForConnection (int port)` - Wait for connection on a given port, forwarded to the communicator
- `obj.CloseConnection ()` - Close a connection, forwarded to the communicator
- `int = obj.ConnectTo (string hostName, int port)` - Open a connection to a give machine, forwarded to the communicator
- `int = obj.GetSwapBytesInReceivedData ()`
- `obj.SetCommunicator (vtkSocketCommunicator comm)` - Set the communicator used in normal and rmi communications.
- `vtkMultiProcessController = obj.CreateCompliantController ()` - FOOLISH MORTALS! Thou hast forsaken the sacred laws of ad-hoc polymorphism when thou broke a critical assumption of the superclass (namely, each process has thine own id). The time frame to fix thy error has passed. Too much code has come to rely on this abhorrent behavior. Instead, we offer this gift: a method for creating an equivalent communicator with correct process id semantics. The calling code is responsible for deleting this controller.

38.57 vtkSubCommunicator

38.57.1 Usage

This class provides an implementation for communicating on process groups. In general, you should never use this class directly. Instead, use the vtkMultiProcessController::CreateSubController method.

.SECTION BUGS

Because all communication is delegated to the original communicator, any error will report process ids with respect to the original communicator, not this communicator that was actually used.

To create an instance of class vtkSubCommunicator, simply invoke its constructor as follows

```
obj = vtkSubCommunicator
```

38.57.2 Methods

The class vtkSubCommunicator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSubCommunicator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSubCommunicator = obj.NewInstance ()`
- `vtkSubCommunicator = obj.SafeDownCast (vtkObject o)`
- `vtkProcessGroup = obj.GetGroup ()` - Set/get the group on which communication will happen.
- `obj.SetGroup (vtkProcessGroup group)` - Set/get the group on which communication will happen.

38.58 vtkSubGroup

38.58.1 Usage

This class provides scalable broadcast, reduce, etc. using only a vtkMultiProcessController. It does not require MPI. Users are vtkPKdTree and vtkDistributedDataFilter.

.SECTION Note This class will be deprecated soon. Instead of using this class, use the collective and subgrouping operations now built into vtkMultiProcessController. The only reason this class is not deprecated already is because vtkPKdTree relies heavily on this class in ways that are not easy to work around. Since vtkPKdTree is due for a major overhaul anyway, we are leaving things the way they are for now.

To create an instance of class vtkSubGroup, simply invoke its constructor as follows

```
obj = vtkSubGroup
```

38.58.2 Methods

The class vtkSubGroup has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSubGroup class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSubGroup = obj.NewInstance ()`
- `vtkSubGroup = obj.SafeDownCast (vtkObject o)`
- `int = obj.Initialize (int p0, int p1, int me, int tag, vtkCommunicator c)`
- `int = obj.Gather (int data, int to, int length, int root)`
- `int = obj.Gather (string data, string to, int length, int root)`
- `int = obj.Gather (float data, float to, int length, int root)`
- `int = obj.Broadcast (float data, int length, int root)`
- `int = obj.Broadcast (double data, int length, int root)`
- `int = obj.Broadcast (int data, int length, int root)`

- `int = obj.Broadcast (string data, int length, int root)`
- `int = obj.ReduceSum (int data, int to, int length, int root)`
- `int = obj.ReduceMax (float data, float to, int length, int root)`
- `int = obj.ReduceMax (double data, double to, int length, int root)`
- `int = obj.ReduceMax (int data, int to, int length, int root)`
- `int = obj.ReduceMin (float data, float to, int length, int root)`
- `int = obj.ReduceMin (double data, double to, int length, int root)`
- `int = obj.ReduceMin (int data, int to, int length, int root)`
- `obj.setGatherPattern (int root, int length)`
- `int = obj.getLocalRank (int processID)`
- `int = obj.Barrier ()`
- `obj.PrintSubGroup () const`

38.59 vtkTemporalFractal

38.59.1 Usage

vtkTemporalFractal is a collection of uniform grids. All have the same dimensions. Each block has a different origin and spacing. It uses mandelbrot to create cell data. I scale the fractal array to look like a volme fraction. I may also add block id and level as extra cell arrays. This source produces a vtkHierarchicalBoxDataSet when GenerateRectilinearGrids is off, otherwise produces a vtkMultiBlockDataSet.

To create an instance of class vtkTemporalFractal, simply invoke its constructor as follows

```
obj = vtkTemporalFractal
```

38.59.2 Methods

The class vtkTemporalFractal has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalFractal class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTemporalFractal = obj.NewInstance ()`
- `vtkTemporalFractal = obj.SafeDownCast (vtkObject o)`
- `obj.SetFractalValue (float )` - Essentially the iso surface value. The fractal array is scaled to map this value to 0.5 for use as a volume fraction.
- `float = obj.GetFractalValue ()` - Essentially the iso surface value. The fractal array is scaled to map this value to 0.5 for use as a volume fraction.
- `obj.SetMaximumLevel (int )` - Any blocks touching a predefined line will be subdivided to this level. Other blocks are subdivided so that neighboring blocks only differ by one level.

- `int = obj.GetMaximumLevel ()` - Any blocks touching a predefined line will be subdivided to this level. Other blocks are subdivided so that neighboring blocks only differ by one level.
- `obj.SetDimensions (int )` - XYZ dimensions of cells.
- `int = obj.GetDimensions ()` - XYZ dimensions of cells.
- `obj.SetGhostLevels (int )` - For testing ghost levels.
- `int = obj.GetGhostLevels ()` - For testing ghost levels.
- `obj.GhostLevelsOn ()` - For testing ghost levels.
- `obj.GhostLevelsOff ()` - For testing ghost levels.
- `obj.SetGenerateRectilinearGrids (int )` - Generate either rectilinear grids either uniform grids. Default is false.
- `int = obj.GetGenerateRectilinearGrids ()` - Generate either rectilinear grids either uniform grids. Default is false.
- `obj.GenerateRectilinearGridsOn ()` - Generate either rectilinear grids either uniform grids. Default is false.
- `obj.GenerateRectilinearGridsOff ()` - Generate either rectilinear grids either uniform grids. Default is false.
- `obj.SetDiscreteTimeSteps (int )` - Limit this source to discrete integer time steps Default is off (continuous)
- `int = obj.GetDiscreteTimeSteps ()` - Limit this source to discrete integer time steps Default is off (continuous)
- `obj.DiscreteTimeStepsOn ()` - Limit this source to discrete integer time steps Default is off (continuous)
- `obj.DiscreteTimeStepsOff ()` - Limit this source to discrete integer time steps Default is off (continuous)
- `obj.SetTwoDimensional (int )` - Make a 2D data set to test.
- `int = obj.GetTwoDimensional ()` - Make a 2D data set to test.
- `obj.TwoDimensionalOn ()` - Make a 2D data set to test.
- `obj.TwoDimensionalOff ()` - Make a 2D data set to test.
- `obj.SetAsymetric (int )` - Test the case when the blocks do not have the same sizes. Adds 2 to the x extent of the far x blocks (level 1).
- `int = obj.GetAsymetric ()` - Test the case when the blocks do not have the same sizes. Adds 2 to the x extent of the far x blocks (level 1).
- `obj.SetAdaptiveSubdivision (int )` - Make the division adaptive or not, defaults to Adaptive
- `int = obj.GetAdaptiveSubdivision ()` - Make the division adaptive or not, defaults to Adaptive
- `obj.AdaptiveSubdivisionOn ()` - Make the division adaptive or not, defaults to Adaptive
- `obj.AdaptiveSubdivisionOff ()` - Make the division adaptive or not, defaults to Adaptive

38.60 vtkTemporalInterpolatedVelocityField

38.60.1 Usage

vtkTemporalInterpolatedVelocityField is a general purpose helper for the temporal particle tracing code (vtkTemporalStreamTracer)

It maintains two copies of vtkCachingInterpolatedVelocityField internally and uses them to obtain velocity values at time T0 and T1.

In fact the class does quite a bit more than this because when the geometry of the datasets is the same at T0 and T1, we can re-use cached cell Ids and weights used in the cell interpolation routines. Additionally, the same weights can be used when interpolating (point) scalar values and computing vorticity etc.

To create an instance of class vtkTemporalInterpolatedVelocityField, simply invoke its constructor as follows

```
obj = vtkTemporalInterpolatedVelocityField
```

38.60.2 Methods

The class vtkTemporalInterpolatedVelocityField has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalInterpolatedVelocityField class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTemporalInterpolatedVelocityField = obj.NewInstance ()`
- `vtkTemporalInterpolatedVelocityField = obj.SafeDownCast (vtkObject o)`
- `int = obj.FunctionValues (double x, double u)` - Evaluate the velocity field, f, at (x, y, z, t). For now, t is ignored.
- `int = obj.FunctionValuesAtT (int T, double x, double u)` - Evaluate the velocity field, f, at (x, y, z, t). For now, t is ignored.
- `obj.SelectVectors (string fieldName)` - In order to use this class, two sets of data must be supplied, corresponding to times T1 and T2. Data is added via this function.
- `obj.SetDataSetAtTime (int I, int N, double T, vtkDataSet dataset, bool staticdataset)` - In order to use this class, two sets of data must be supplied, corresponding to times T1 and T2. Data is added via this function.
- `obj.ClearCache ()` - Set the last cell id to -1 so that the next search does not start from the previous cell
- `int = obj.TestPoint (double x)` - A utility function which evaluates the point at T1, T2 to see if it is inside the data at both times or only one.
- `int = obj.QuickTestPoint (double x)` - A utility function which evaluates the point at T1, T2 to see if it is inside the data at both times or only one.
- `double = obj. GetLastGoodVelocity ()` - If an interpolation was successful, we can retrieve the last computed value from here. Initial value is (0.0,0.0,0.0)
- `double = obj.GetCurrentWeight ()` - Get the most recent weight between 0-¿1 from T1-¿T2. Initial value is 0.
- `bool = obj.InterpolatePoint (vtkPointData outPD1, vtkPointData outPD2, vtkIdType outIndex)`

- `bool = obj.InterpolatePoint (int T, vtkPointData outPD1, vtkIdType outIndex)`
- `obj.ShowCacheResults ()`
- `bool = obj.IsStatic (int datasetIndex)`
- `obj.AdvanceOneTimeStep ()`

38.61 vtkTemporalStreamTracer

38.61.1 Usage

vtkTemporalStreamTracer is a filter that integrates a vector field to generate

To create an instance of class vtkTemporalStreamTracer, simply invoke its constructor as follows

```
obj = vtkTemporalStreamTracer
```

38.61.2 Methods

The class vtkTemporalStreamTracer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTemporalStreamTracer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTemporalStreamTracer = obj.NewInstance ()`
- `vtkTemporalStreamTracer = obj.SafeDownCast (vtkObject o)`
- `obj.SetTimeStep (int )` - Set/Get the TimeStep. This is the primary means of advancing the particles. The TimeStep should be animated and this will drive the pipeline forcing timesteps to be fetched from upstream.
- `int = obj.GetTimeStep ()` - Set/Get the TimeStep. This is the primary means of advancing the particles. The TimeStep should be animated and this will drive the pipeline forcing timesteps to be fetched from upstream.
- `obj.SetIgnorePipelineTime (int )` - To get around problems with the Paraview Animation controls we can just animate the time step and ignore the TIME_ requests
- `int = obj.GetIgnorePipelineTime ()` - To get around problems with the Paraview Animation controls we can just animate the time step and ignore the TIME_ requests
- `obj.IgnorePipelineTimeOn ()` - To get around problems with the Paraview Animation controls we can just animate the time step and ignore the TIME_ requests
- `obj.IgnorePipelineTimeOff ()` - To get around problems with the Paraview Animation controls we can just animate the time step and ignore the TIME_ requests
- `obj.SetTimeStepResolution (double )` - If the data source does not have the correct time values present on each time step - setting this value to non unity can be used to adjust the time step size from 1s pre step to 1x_TimeStepResolution : Not functional in this version. Broke it @todo, put back time scaling
- `double = obj.GetTimeStepResolution ()` - If the data source does not have the correct time values present on each time step - setting this value to non unity can be used to adjust the time step size from 1s pre step to 1x_TimeStepResolution : Not functional in this version. Broke it @todo, put back time scaling

- `obj.SetForceReinjectionEveryNSteps (int )` - When animating particles, it is nice to inject new ones every Nth step to produce a continuous flow. Setting ForceReinjectionEveryNSteps to a non zero value will cause the particle source to reinject particles every Nth step even if it is otherwise unchanged. Note that if the particle source is also animated, this flag will be redundant as the particles will be reinjected whenever the source changes anyway

- `int = obj.GetForceReinjectionEveryNSteps ()` - When animating particles, it is nice to inject new ones every Nth step to produce a continuous flow. Setting ForceReinjectionEveryNSteps to a non zero value will cause the particle source to reinject particles every Nth step even if it is otherwise unchanged. Note that if the particle source is also animated, this flag will be redundant as the particles will be reinjected whenever the source changes anyway

- `obj.SetTerminationTime (double )` - Setting TerminationTime to a positive value will cause particles to terminate when the time is reached. Use a vlue of zero to diable termination. The units of time should be consistent with the primary time variable.

- `double = obj.GetTerminationTime ()` - Setting TerminationTime to a positive value will cause particles to terminate when the time is reached. Use a vlue of zero to diable termination. The units of time should be consistent with the primary time variable.

- `obj.SetTerminationTimeUnit (int )` - The units of TerminationTime may be actual 'Time' units as described by the data, or just TimeSteps of iteration.

- `int = obj.GetTerminationTimeUnit ()` - The units of TerminationTime may be actual 'Time' units as described by the data, or just TimeSteps of iteration.

- `obj.SetTerminationTimeUnitToTimeUnit ()` - The units of TerminationTime may be actual 'Time' units as described by the data, or just TimeSteps of iteration.

- `obj.SetTerminationTimeUnitToStepUnit ()` - The units of TerminationTime may be actual 'Time' units as described by the data, or just TimeSteps of iteration.

- `obj.SetStaticSeeds (int )` - if StaticSeeds is set and the mesh is static, then every time particles are injected we can re-use the same injection information. We classify particles according to processor just once before start. If StaticSeeds is set and a moving seed source is specified the motion will be ignored and results will not be as expected.

- `int = obj.GetStaticSeeds ()` - if StaticSeeds is set and the mesh is static, then every time particles are injected we can re-use the same injection information. We classify particles according to processor just once before start. If StaticSeeds is set and a moving seed source is specified the motion will be ignored and results will not be as expected.

- `obj.StaticSeedsOn ()` - if StaticSeeds is set and the mesh is static, then every time particles are injected we can re-use the same injection information. We classify particles according to processor just once before start. If StaticSeeds is set and a moving seed source is specified the motion will be ignored and results will not be as expected.

- `obj.StaticSeedsOff ()` - if StaticSeeds is set and the mesh is static, then every time particles are injected we can re-use the same injection information. We classify particles according to processor just once before start. If StaticSeeds is set and a moving seed source is specified the motion will be ignored and results will not be as expected.

- `obj.SetStaticMesh (int )` - if StaticMesh is set, many optimizations for cell caching can be assumed. if StaticMesh is not set, the algorithm will attempt to find out if optimizations can be used, but setting it to true will force all optimizations. Do not Set StaticMesh to true if a dynamic mesh is being used as this will invalidate all results.

- `int = obj.GetStaticMesh ()` - if StaticMesh is set, many optimizations for cell caching can be assumed. if StaticMesh is not set, the algorithm will attempt to find out if optimizations can be used, but setting it to true will force all optimizations. Do not Set StaticMesh to true if a dynamic mesh is being used as this will invalidate all results.
- `obj.StaticMeshOn ()` - if StaticMesh is set, many optimizations for cell caching can be assumed. if StaticMesh is not set, the algorithm will attempt to find out if optimizations can be used, but setting it to true will force all optimizations. Do not Set StaticMesh to true if a dynamic mesh is being used as this will invalidate all results.
- `obj.StaticMeshOff ()` - if StaticMesh is set, many optimizations for cell caching can be assumed. if StaticMesh is not set, the algorithm will attempt to find out if optimizations can be used, but setting it to true will force all optimizations. Do not Set StaticMesh to true if a dynamic mesh is being used as this will invalidate all results.
- `obj.SetController (vtkMultiProcessController controller)` - Set/Get the controller used when sending particles between processes The controller must be an instance of vtkMPIController. If VTK was compiled without VTK_USE_MPI on, then the Controller is simply ignored.
- `vtkMultiProcessController = obj.GetController ()` - Set/Get the controller used when sending particles between processes The controller must be an instance of vtkMPIController. If VTK was compiled without VTK_USE_MPI on, then the Controller is simply ignored.
- `obj.SetParticleWriter (vtkAbstractParticleWriter pw)` - Set/Get the Writer associated with this Particle Tracer Ideally a parallel IO capable vtkH5PartWriter should be used which will collect particles from all parallel processes and write them to a single HDF5 file.
- `vtkAbstractParticleWriter = obj.GetParticleWriter ()` - Set/Get the Writer associated with this Particle Tracer Ideally a parallel IO capable vtkH5PartWriter should be used which will collect particles from all parallel processes and write them to a single HDF5 file.
- `obj.SetParticleFileName (string )` - Set/Get the filename to be used with the particle writer when dumping particles to disk
- `string = obj.GetParticleFileName ()` - Set/Get the filename to be used with the particle writer when dumping particles to disk
- `obj.SetEnableParticleWriting (int )` - Set/Get the filename to be used with the particle writer when dumping particles to disk
- `int = obj.GetEnableParticleWriting ()` - Set/Get the filename to be used with the particle writer when dumping particles to disk
- `obj.EnableParticleWritingOn ()` - Set/Get the filename to be used with the particle writer when dumping particles to disk
- `obj.EnableParticleWritingOff ()` - Set/Get the filename to be used with the particle writer when dumping particles to disk
- `obj.AddSourceConnection (vtkAlgorithmOutput input)` - Provide support for multiple see sources
- `obj.RemoveAllSources ()` - Provide support for multiple see sources

38.62 vtkTransmitImageDataPiece

38.62.1 Usage

This filter updates the appropriate piece by requesting the piece from process 0. Process 0 always updates all of the data. It is important that Execute get called on all processes, otherwise the filter will deadlock.

To create an instance of class vtkTransmitImageDataPiece, simply invoke its constructor as follows

```
obj = vtkTransmitImageDataPiece
```

38.62.2 Methods

The class vtkTransmitImageDataPiece has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransmitImageDataPiece class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransmitImageDataPiece = obj.NewInstance ()`
- `vtkTransmitImageDataPiece = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `obj.SetCreateGhostCells (int )` - Turn on/off creating ghost cells (on by default).
- `int = obj.GetCreateGhostCells ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOn ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOff ()` - Turn on/off creating ghost cells (on by default).

38.63 vtkTransmitPolyDataPiece

38.63.1 Usage

This filter updates the appropriate piece by requesting the piece from process 0. Process 0 always updates all of the data. It is important that Execute get called on all processes, otherwise the filter will deadlock.

To create an instance of class vtkTransmitPolyDataPiece, simply invoke its constructor as follows

```
obj = vtkTransmitPolyDataPiece
```

38.63.2 Methods

The class vtkTransmitPolyDataPiece has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransmitPolyDataPiece class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransmitPolyDataPiece = obj.NewInstance ()`
- `vtkTransmitPolyDataPiece = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.

- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `obj.SetCreateGhostCells (int )` - Turn on/off creating ghost cells (on by default).
- `int = obj.GetCreateGhostCells ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOn ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOff ()` - Turn on/off creating ghost cells (on by default).

38.64 vtkTransmitRectilinearGridPiece

38.64.1 Usage

This filter updates the appropriate piece by requesting the piece from process 0. Process 0 always updates all of the data. It is important that Execute get called on all processes, otherwise the filter will deadlock.

To create an instance of class vtkTransmitRectilinearGridPiece, simply invoke its constructor as follows

```
obj = vtkTransmitRectilinearGridPiece
```

38.64.2 Methods

The class vtkTransmitRectilinearGridPiece has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransmitRectilinearGridPiece class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransmitRectilinearGridPiece = obj.NewInstance ()`
- `vtkTransmitRectilinearGridPiece = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `obj.SetCreateGhostCells (int )` - Turn on/off creating ghost cells (on by default).
- `int = obj.GetCreateGhostCells ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOn ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOff ()` - Turn on/off creating ghost cells (on by default).

38.65 vtkTransmitStructuredGridPiece

38.65.1 Usage

This filter updates the appropriate piece by requesting the piece from process 0. Process 0 always updates all of the data. It is important that Execute get called on all processes, otherwise the filter will deadlock.

To create an instance of class vtkTransmitStructuredGridPiece, simply invoke its constructor as follows

```
obj = vtkTransmitStructuredGridPiece
```

38.65.2 Methods

The class vtkTransmitStructuredGridPiece has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransmitStructuredGridPiece class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransmitStructuredGridPiece = obj.NewInstance ()`
- `vtkTransmitStructuredGridPiece = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `obj.SetCreateGhostCells (int )` - Turn on/off creating ghost cells (on by default).
- `int = obj.GetCreateGhostCells ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOn ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOff ()` - Turn on/off creating ghost cells (on by default).

38.66 vtkTransmitUnstructuredGridPiece

38.66.1 Usage

This filter updates the appropriate piece by requesting the piece from process 0. Process 0 always updates all of the data. It is important that Execute get called on all processes, otherwise the filter will deadlock.

To create an instance of class vtkTransmitUnstructuredGridPiece, simply invoke its constructor as follows

```
obj = vtkTransmitUnstructuredGridPiece
```

38.66.2 Methods

The class vtkTransmitUnstructuredGridPiece has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransmitUnstructuredGridPiece class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransmitUnstructuredGridPiece = obj.NewInstance ()`
- `vtkTransmitUnstructuredGridPiece = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `vtkMultiProcessController = obj.GetController ()` - By defualt this filter uses the global controller, but this method can be used to set another instead.
- `obj.SetCreateGhostCells (int )` - Turn on/off creating ghost cells (on by default).

- `int = obj.GetCreateGhostCells ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOn ()` - Turn on/off creating ghost cells (on by default).
- `obj.CreateGhostCellsOff ()` - Turn on/off creating ghost cells (on by default).

38.67 vtkTreeCompositer

38.67.1 Usage

vtkTreeCompositer operates in multiple processes. Each compositer has a render window. They use a vtkMultiProcessController to communicate the color and depth buffer to process 0's render window. It will not handle transparency well.

To create an instance of class vtkTreeCompositer, simply invoke its constructor as follows

```
obj = vtkTreeCompositer
```

38.67.2 Methods

The class vtkTreeCompositer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeCompositer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeCompositer = obj.NewInstance ()`
- `vtkTreeCompositer = obj.SafeDownCast (vtkObject o)`
- `obj.CompositeBuffer (vtkDataArray pBuf, vtkFloatArray zBuf, vtkDataArray pTmp, vtkFloatArray zTmp)`

38.68 vtkVPICReader

38.68.1 Usage

vtkDataReader is a helper superclass that reads the vtk data file header, dataset type, and attribute data (point and cell attributes such as scalars, vectors, normals, etc.) from a vtk data file. See text for the format of the various vtk file types.

To create an instance of class vtkVPICReader, simply invoke its constructor as follows

```
obj = vtkVPICReader
```

38.68.2 Methods

The class vtkVPICReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVPICReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVPICReader = obj.NewInstance ()`

- `vtkVPICReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify file name of VPIC data file to read.
- `string = obj.GetFileName ()` - Specify file name of VPIC data file to read.
- `obj.SetStride (int , int , int )` - Set the stride in each dimension
- `obj.SetStride (int  a[3])` - Set the stride in each dimension
- `int = obj. GetStride ()` - Set the stride in each dimension
- `obj.SetXExtent (int , int )` - Set the simulation file decomposition in each dimension
- `obj.SetXExtent (int  a[2])` - Set the simulation file decomposition in each dimension
- `obj.SetYExtent (int , int )` - Set the simulation file decomposition in each dimension
- `obj.SetYExtent (int  a[2])` - Set the simulation file decomposition in each dimension
- `obj.SetZExtent (int , int )` - Set the simulation file decomposition in each dimension
- `obj.SetZExtent (int  a[2])` - Set the simulation file decomposition in each dimension
- `int = obj. GetXLayout ()`
- `int = obj. GetYLayout ()`
- `int = obj. GetZLayout ()`
- `vtkImageData = obj.GetOutput ()` - Get the reader's output
- `vtkImageData = obj.GetOutput (int index)` - Get the reader's output
- `int = obj.GetNumberOfPointArrays ()` - The following methods allow selective reading of solutions fields. By default, ALL data fields on the nodes are read, but this can be modified.
- `string = obj.GetPointArrayName (int index)` - The following methods allow selective reading of solutions fields. By default, ALL data fields on the nodes are read, but this can be modified.
- `int = obj.GetPointArrayStatus (string name)` - The following methods allow selective reading of solutions fields. By default, ALL data fields on the nodes are read, but this can be modified.
- `obj.SetPointArrayStatus (string name, int status)` - The following methods allow selective reading of solutions fields. By default, ALL data fields on the nodes are read, but this can be modified.
- `obj.DisableAllPointArrays ()` - The following methods allow selective reading of solutions fields. By default, ALL data fields on the nodes are read, but this can be modified.
- `obj.EnableAllPointArrays ()` - The following methods allow selective reading of solutions fields. By default, ALL data fields on the nodes are read, but this can be modified.

38.69 vtkWindBladeReader

38.69.1 Usage

vtkWindBladeReader is a source object that reads WindBlade files which are block binary files with tags before and after each block giving the number of bytes within the block. The number of data variables dumped varies. The data is 3D rectilinear with irregular spacing on the Z dimension.

To create an instance of class vtkWindBladeReader, simply invoke its constructor as follows

```
obj = vtkWindBladeReader
```

38.69.2 Methods

The class vtkWindBladeReader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWindBladeReader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWindBladeReader = obj.NewInstance ()`
- `vtkWindBladeReader = obj.SafeDownCast (vtkObject o)`
- `obj.SetFilename (string )`
- `string = obj.GetFilename ()`
- `obj.SetWholeExtent (int , int , int , int , int , int )`
- `obj.SetWholeExtent (int  a[6])`
- `int = obj. GetWholeExtent ()`
- `obj.SetSubExtent (int , int , int , int , int , int )`
- `obj.SetSubExtent (int  a[6])`
- `int = obj. GetSubExtent ()`
- `vtkStructuredGrid = obj.GetFieldOutput ()` - Get the reader's output
- `vtkUnstructuredGrid = obj.GetBladeOutput ()` - Get the reader's output
- `int = obj.GetNumberOfPointArrays ()` - The following methods allow selective reading of solutions fields. By default, ALL data fields on the nodes are read, but this can be modified.
- `string = obj.GetPointArrayName (int index)` - The following methods allow selective reading of solutions fields. By default, ALL data fields on the nodes are read, but this can be modified.
- `int = obj.GetPointArrayStatus (string name)`
- `obj.SetPointArrayStatus (string name, int status)`
- `obj.DisableAllPointArrays ()`
- `obj.EnableAllPointArrays ()`

38.70 vtkXMLPHierarchicalBoxDataWriter

38.70.1 Usage

vtkXMLPCompositeDataWriter writes (in parallel or serially) the VTK XML multi-group, multi-block hierarchical and hierarchical box files. XML multi-group data files are meta-files that point to a list of serial VTK XML files.

To create an instance of class vtkXMLPHierarchicalBoxDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLPHierarchicalBoxDataWriter
```

38.70.2 Methods

The class vtkXMLPHierarchicalBoxDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPHierarchicalBoxDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPHierarchicalBoxDataWriter = obj.NewInstance ()`
- `vtkXMLPHierarchicalBoxDataWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Controller used to communicate data type of blocks. By default, the global controller is used. If you want another controller to be used, set it with this. If no controller is set, only the local blocks will be written to the meta-file.
- `vtkMultiProcessController = obj.GetController ()` - Controller used to communicate data type of blocks. By default, the global controller is used. If you want another controller to be used, set it with this. If no controller is set, only the local blocks will be written to the meta-file.
- `obj.SetWriteMetaFile (int flag)` - Set whether this instance will write the meta-file. WriteMetaFile is set to flag only on process 0 and all other processes have WriteMetaFile set to 0 by default.

38.71 vtkXMLPMultiBlockDataWriter

38.71.1 Usage

vtkXMLPCompositeDataWriter writes (in parallel or serially) the VTK XML multi-group, multi-block hierarchical and hierarchical box files. XML multi-group data files are meta-files that point to a list of serial VTK XML files.

To create an instance of class vtkXMLPMultiBlockDataWriter, simply invoke its constructor as follows

```
obj = vtkXMLPMultiBlockDataWriter
```

38.71.2 Methods

The class vtkXMLPMultiBlockDataWriter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXMLPMultiBlockDataWriter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXMLPMultiBlockDataWriter = obj.NewInstance ()`
- `vtkXMLPMultiBlockDataWriter = obj.SafeDownCast (vtkObject o)`
- `obj.SetController (vtkMultiProcessController )` - Controller used to communicate data type of blocks. By default, the global controller is used. If you want another controller to be used, set it with this. If no controller is set, only the local blocks will be written to the meta-file.
- `vtkMultiProcessController = obj.GetController ()` - Controller used to communicate data type of blocks. By default, the global controller is used. If you want another controller to be used, set it with this. If no controller is set, only the local blocks will be written to the meta-file.
- `obj.SetWriteMetaFile (int flag)` - Set whether this instance will write the meta-file. WriteMetaFile is set to flag only on process 0 and all other processes have WriteMetaFile set to 0 by default.

Chapter 39

Visualization Toolkit Rendering Classes

39.1 vtkAbstractMapper3D

39.1.1 Usage

vtkAbstractMapper3D is an abstract class to specify interface between 3D data and graphics primitives or software rendering techniques. Subclasses of vtkAbstractMapper3D can be used for rendering geometry or rendering volumetric data.

This class also defines an API to support hardware clipping planes (at most six planes can be defined). It also provides geometric data about the input data it maps, such as the bounding box and center.

To create an instance of class vtkAbstractMapper3D, simply invoke its constructor as follows

```
obj = vtkAbstractMapper3D
```

39.1.2 Methods

The class vtkAbstractMapper3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractMapper3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractMapper3D = obj.NewInstance ()`
- `vtkAbstractMapper3D = obj.SafeDownCast (vtkObject o)`
- `double = obj.GetBounds ()` - Return bounding box (array of six doubles) of data expressed as (xmin,xmax, ymin,ymax, zmin,zmax). Update this-¿Bounds as a side effect.
- `obj.GetBounds (double bounds[6])` - Get the bounds for this mapper as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax).
- `double = obj.GetCenter ()` - Return the Center of this mapper's data.
- `obj.GetCenter (double center[3])` - Return the diagonal length of this mappers bounding box.
- `double = obj.GetLength ()` - Return the diagonal length of this mappers bounding box.
- `int = obj.IsARayCastMapper ()` - Is this a "render into image" mapper? A subclass would return 1 if the mapper produces an image by rendering into a software image buffer.
- `int = obj.IsARenderIntoImageMapper ()`

39.2 vtkAbstractPicker

39.2.1 Usage

vtkAbstractPicker is an abstract superclass that defines a minimal API for its concrete subclasses. The minimum functionality of a picker is to return the x-y-z global coordinate position of a pick (the pick itself is defined in display coordinates).

The API to this class is to invoke the Pick() method with a selection point (in display coordinates - pixels) and a renderer. Then get the resulting pick position in global coordinates with the GetPickPosition() method.

vtkPicker fires events during the picking process. These events are StartPickEvent, PickEvent, and EndPickEvent which are invoked prior to picking, when something is picked, and after all picking candidates have been tested. Note that during the pick process the PickEvent of vtkProp (and its subclasses such as vtkActor) is fired prior to the PickEvent of vtkPicker.

To create an instance of class vtkAbstractPicker, simply invoke its constructor as follows

```
obj = vtkAbstractPicker
```

39.2.2 Methods

The class vtkAbstractPicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractPicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractPicker = obj.NewInstance ()`
- `vtkAbstractPicker = obj.SafeDownCast (vtkObject o)`
- `vtkRenderer = obj.GetRenderer ()` - Get the renderer in which pick event occurred.
- `double = obj. GetSelectionPoint ()` - Get the selection point in screen (pixel) coordinates. The third value is related to z-buffer depth. (Normally should be =0.)
- `double = obj. GetPickPosition ()` - Return position in global coordinates of pick point.
- `int = obj.Pick (double selectionX, double selectionY, double selectionZ, vtkRenderer renderer)` - Perform pick operation with selection point provided. Normally the first two values for the selection point are x-y pixel coordinate, and the third value is =0. Return non-zero if something was successfully picked.
- `int = obj.Pick (double selectionPt[3], vtkRenderer ren)` - provided. Normally the first two values for the selection point are x-y pixel coordinate, and the third value is =0. Return non-zero if something was successfully picked.
- `obj.SetPickFromList (int )` - Use these methods to control whether to limit the picking to this list (rather than renderer's actors). Make sure that the pick list contains actors that referred to by the picker's renderer.
- `int = obj.GetPickFromList ()` - Use these methods to control whether to limit the picking to this list (rather than renderer's actors). Make sure that the pick list contains actors that referred to by the picker's renderer.
- `obj.PickFromListOn ()` - Use these methods to control whether to limit the picking to this list (rather than renderer's actors). Make sure that the pick list contains actors that referred to by the picker's renderer.

- `obj.PickFromListOff ()` - Use these methods to control whether to limit the picking to this list (rather than renderer's actors). Make sure that the pick list contains actors that referred to by the picker's renderer.
- `obj.InitializePickList ()` - Initialize list of actors in pick list.
- `obj.AddPickList (vtkProp )` - Add an actor to the pick list.
- `obj.DeletePickList (vtkProp )` - Delete an actor from the pick list.
- `vtkPropCollection = obj.GetPickList ()`

39.3 vtkAbstractPropPicker

39.3.1 Usage

vtkAbstractPropPicker is an abstract superclass for pickers that can pick an instance of vtkProp. Some pickers, like vtkWorldPointPicker (not a subclass of this class), cannot identify the prop that is picked. Subclasses of vtkAbstractPropPicker return a prop in the form of a vtkAssemblyPath when a pick is invoked. Note that an vtkAssemblyPath contain a list of vtkAssemblyNodes, each of which in turn contains a reference to a vtkProp and a 4x4 transformation matrix. The path fully describes the entire pick path, so you can pick assemblies or portions of assemblies, or just grab the tail end of the vtkAssemblyPath (which is the picked prop).

To create an instance of class vtkAbstractPropPicker, simply invoke its constructor as follows

```
obj = vtkAbstractPropPicker
```

39.3.2 Methods

The class vtkAbstractPropPicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractPropPicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractPropPicker = obj.NewInstance ()`
- `vtkAbstractPropPicker = obj.SafeDownCast (vtkObject o)`
- `obj.SetPath (vtkAssemblyPath )` - Return the vtkAssemblyPath that has been picked. The assembly path lists all the vtkProps that form an assembly. If no assembly is present, then the assembly path will have one node (which is the picked prop). The set method is used internally to set the path. (Note: the structure of an assembly path is a collection of vtkAssemblyNode, each node pointing to a vtkProp and (possibly) a transformation matrix.)
- `vtkAssemblyPath = obj.GetPath ()` - Return the vtkAssemblyPath that has been picked. The assembly path lists all the vtkProps that form an assembly. If no assembly is present, then the assembly path will have one node (which is the picked prop). The set method is used internally to set the path. (Note: the structure of an assembly path is a collection of vtkAssemblyNode, each node pointing to a vtkProp and (possibly) a transformation matrix.)
- `vtkProp = obj.GetViewProp ()` - Return the vtkProp that has been picked. If NULL, nothing was picked. If anything at all was picked, this method will return something.
- `vtkProp3D = obj.GetProp3D ()` - Return the vtkProp that has been picked. If NULL, no vtkProp3D was picked.

- `vtkActor = obj.GetActor ()` - Return the vtkActor that has been picked. If NULL, no actor was picked.
- `vtkActor2D = obj.GetActor2D ()` - Return the vtkActor2D that has been picked. If NULL, no actor2D was picked.
- `vtkVolume = obj.GetVolume ()` - Return the vtkVolume that has been picked. If NULL, no volume was picked.
- `vtkAssembly = obj.GetAssembly ()` - Return the vtkAssembly that has been picked. If NULL, no assembly was picked. (Note: the returned assembly is the first node in the assembly path. If the path is one node long, then the assembly and the prop are the same, assuming that the first node is a vtkAssembly.)
- `vtkPropAssembly = obj.GetPropAssembly ()` - Return the vtkPropAssembly that has been picked. If NULL, no prop assembly was picked. (Note: the returned prop assembly is the first node in the assembly path. If the path is one node long, then the prop assembly and the prop are the same, assuming that the first node is a vtkPropAssembly.)
- `vtkProp = obj.GetProp ()` - @deprecated Replaced by vtkAbstractPicker::GetViewProp() as of VTK 5.0.

39.4 vtkAbstractVolumeMapper

39.4.1 Usage

vtkAbstractVolumeMapper is the abstract definition of a volume mapper. Specific subclasses deal with different specific types of data input

To create an instance of class vtkAbstractVolumeMapper, simply invoke its constructor as follows

```
obj = vtkAbstractVolumeMapper
```

39.4.2 Methods

The class vtkAbstractVolumeMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractVolumeMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAbstractVolumeMapper = obj.NewInstance ()`
- `vtkAbstractVolumeMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkDataSet )` - Set/Get the input data
- `vtkDataSet = obj.GetDataSetInput ()` - Set/Get the input data
- `vtkDataObject = obj.GetDataObjectInput ()` - Set/Get the input data
- `double = obj.GetBounds ()` - Return bounding box (array of six doubles) of data expressed as (xmin,xmax, ymin,ymax, zmin,zmax).
- `obj.GetBounds (double bounds[6])` - Return bounding box (array of six doubles) of data expressed as (xmin,xmax, ymin,ymax, zmin,zmax).

- `obj.SetScalarMode (int )` - Control how the mapper works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the mapper will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the mapper to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectScalarArray.

- `int = obj.GetScalarMode ()` - Control how the mapper works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the mapper will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the mapper to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectScalarArray.

- `obj.SetScalarModeToDefault ()` - Control how the mapper works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the mapper will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the mapper to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectScalarArray.

- `obj.SetScalarModeToUsePointData ()` - Control how the mapper works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the mapper will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the mapper to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectScalarArray.

- `obj.SetScalarModeToUseCellData ()` - Control how the mapper works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the mapper will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the mapper to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectScalarArray.

- `obj.SetScalarModeToUsePointFieldData ()` - Control how the mapper works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the mapper will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the mapper to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectScalarArray.

- `obj.SetScalarModeToUseCellFieldData ()` - Control how the mapper works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the mapper will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the mapper to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectScalarArray.

- `obj.SelectScalarArray (int arrayNum)` - When ScalarMode is set to UsePointFieldData or UseCellFieldData, you can specify which scalar array to use during rendering. The transfer function in the vtkVolumeProperty (attached to the calling vtkVolume) will decide how to convert vectors to colors.
- `obj.SelectScalarArray (string arrayName)` - When ScalarMode is set to UsePointFieldData or UseCellFieldData, you can specify which scalar array to use during rendering. The transfer function in the vtkVolumeProperty (attached to the calling vtkVolume) will decide how to convert vectors to colors.
- `string = obj.GetArrayName ()` - Get the array name or number and component to use for rendering.
- `int = obj.GetArrayId ()` - Get the array name or number and component to use for rendering.
- `int = obj.GetArrayAccessMode ()` - Return the method for obtaining scalar data.
- `string = obj.GetScalarModeAsString ()` - Return the method for obtaining scalar data.

39.5 vtkActor

39.5.1 Usage

vtkActor is used to represent an entity in a rendering scene. It inherits functions related to the actors position, and orientation from vtkProp. The actor also has scaling and maintains a reference to the defining geometry (i.e., the mapper), rendering properties, and possibly a texture map. vtkActor combines these instance variables into one 4x4 transformation matrix as follows: [x y z 1] = [x y z 1] Translate(-origin) Scale(scale) Rot(y) Rot(x) Rot (z) Trans(origin) Trans(position)

To create an instance of class vtkActor, simply invoke its constructor as follows

```
obj = vtkActor
```

39.5.2 Methods

The class vtkActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkActor = obj.NewInstance ()`
- `vtkActor = obj.SafeDownCast (vtkObject o)`
- `obj.GetActors (vtkPropCollection )` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.Render (vtkRenderer , vtkMapper )` - Shallow copy of an actor. Overloads the virtual vtkProp method.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of an actor. Overloads the virtual vtkProp method.

- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.

- `obj.SetProperty (vtkProperty lut)` - Set/Get the property object that controls this actors surface properties. This should be an instance of a vtkProperty object. Every actor must have a property associated with it. If one isn't specified, then one will be generated automatically. Multiple actors can share one property object.

- `vtkProperty = obj.GetProperty ()` - Set/Get the property object that controls this actors surface properties. This should be an instance of a vtkProperty object. Every actor must have a property associated with it. If one isn't specified, then one will be generated automatically. Multiple actors can share one property object.

- `vtkProperty = obj.MakeProperty ()` - Create a new property suitable for use with this type of Actor. For example, a vtkMesaActor should create a vtkMesaProperty in this function. The default is to just call vtkProperty::New.

- `obj.SetBackfaceProperty (vtkProperty lut)` - Set/Get the property object that controls this actors backface surface properties. This should be an instance of a vtkProperty object. If one isn't specified, then the front face properties will be used. Multiple actors can share one property object.

- `vtkProperty = obj.GetBackfaceProperty ()` - Set/Get the property object that controls this actors backface surface properties. This should be an instance of a vtkProperty object. If one isn't specified, then the front face properties will be used. Multiple actors can share one property object.

- `obj.SetTexture (vtkTexture )` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. An actor does not need to have an associated texture map and multiple actors can share one texture.

- `vtkTexture = obj.GetTexture ()` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. An actor does not need to have an associated texture map and multiple actors can share one texture.

- `obj.SetMapper (vtkMapper )` - This is the method that is used to connect an actor to the end of a visualization pipeline, i.e. the mapper. This should be a subclass of vtkMapper. Typically vtkPolyDataMapper and vtkDataSetMapper will be used.

- `vtkMapper = obj.GetMapper ()` - Returns the Mapper that this actor is getting its data from.

- `obj.GetBounds (double bounds[6])` - Get the bounds for this Actor as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax). (The method GetBounds(double bounds[6]) is available from the superclass.)

- `double = obj.GetBounds ()` - Get the bounds for this Actor as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax). (The method GetBounds(double bounds[6]) is available from the superclass.)

- `obj.ApplyProperties ()` - Get the actors mtime plus consider its properties and texture if set.

- `long = obj.GetMTime ()` - Get the actors mtime plus consider its properties and texture if set.

- `long = obj.GetRedrawMTime ()` - Return the mtime of anything that would cause the rendered image to appear differently. Usually this involves checking the mtime of the prop plus anything else it depends on such as properties, textures etc.

- `obj.InitPartTraversal ()` - The following methods are for compatibility. The methods will be deprecated in the near future. Use vtkProp::GetNextPath() (and related functionality) to get the parts in an assembly (or more correctly, the paths in the assembly).

- `vtkActor = obj.GetNextPart ()` - The following methods are for compatibility. The methods will be deprecated in the near future. Use vtkProp::GetNextPath() (and related functionality) to get the parts in an assembly (or more correctly, the paths in the assembly).
- `int = obj.GetNumberOfParts ()` - The following methods are for compatibility. The methods will be deprecated in the near future. Use vtkProp::GetNextPath() (and related functionality) to get the parts in an assembly (or more correctly, the paths in the assembly).
- `bool = obj.GetSupportsSelection ()` - WARNING: INTERNAL METHOD - NOT INTENDED FOR GENERAL USE DO NOT USE THIS METHOD OUTSIDE OF THE RENDERING PROCESS Used by vtkHardwareSelector to determine if the prop supports hardware selection.

39.6 vtkActorCollection

39.6.1 Usage

vtkActorCollection represents and provides methods to manipulate a list of actors (i.e., vtkActor and subclasses). The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkActorCollection, simply invoke its constructor as follows

```
obj = vtkActorCollection
```

39.6.2 Methods

The class vtkActorCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkActorCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkActorCollection = obj.NewInstance ()`
- `vtkActorCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkActor a)` - Add an actor to the list.
- `vtkActor = obj.GetNextActor ()` - Get the next actor in the list.
- `vtkActor = obj.GetLastActor ()` - Get the last actor in the list.
- `vtkActor = obj.GetNextItem ()` - Access routines that are provided for compatibility with previous version of VTK. Please use the GetNextActor(), GetLastActor() variants where possible.
- `vtkActor = obj.GetLastItem ()` - Access routines that are provided for compatibility with previous version of VTK. Please use the GetNextActor(), GetLastActor() variants where possible.
- `obj.ApplyProperties (vtkProperty p)` - Apply properties to all actors in this collection.

39.7 vtkAreaPicker

39.7.1 Usage

The vtkAreaPicker picks all vtkProp3Ds that lie behind the screen space rectangle from x0,y0 and x1,y1. The selection is based upon the bounding box of the prop and is thus not exact.

Like vtkPicker, a pick results in a list of Prop3Ds because many props may lie within the pick frustum. You can also get an AssemblyPath, which in this case is defined to be the path to the one particular prop in the Prop3D list that lies nearest to the near plane.

This picker also returns the selection frustum, defined as either a vtkPlanes, or a set of eight corner vertices in world space. The vtkPlanes version is an ImplicitFunction, which is suitable for use with the vtkExtractGeometry. The six frustum planes are in order: left, right, bottom, top, near, far

Because this picker picks everything within a volume, the world pick point result is ill-defined. Therefore if you ask this class for the world pick position, you will get the centroid of the pick frustum. This may be outside of all props in the prop list.

To create an instance of class vtkAreaPicker, simply invoke its constructor as follows

```
obj = vtkAreaPicker
```

39.7.2 Methods

The class vtkAreaPicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAreaPicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAreaPicker = obj.NewInstance ()`
- `vtkAreaPicker = obj.SafeDownCast (vtkObject o)`
- `obj.SetPickCoords (double x0, double y0, double x1, double y1)` - Set the default screen rectangle to pick in.
- `obj.SetRenderer (vtkRenderer )` - Set the default renderer to pick on.
- `int = obj.Pick ()` - Perform an AreaPick within the default screen rectangle and renderer.
- `int = obj.AreaPick (double x0, double y0, double x1, double y1, vtkRenderer rendererNULL)` - Perform pick operation in volume behind the given screen coordinates. Props intersecting the selection frustum will be accesible via GetProp3D. GetPlanes returns a vtkImplicitFunciton suitable for vtkExtractGeometry.
- `int = obj.Pick (double x0, double y0, double , vtkRenderer rendererNULL)` - Perform pick operation in volume behind the given screen coordinate. This makes a thin frustum around the selected pixel. Note: this ignores Z in order to pick everying in a volume from z=0 to z=1.
- `vtkAbstractMapper3D = obj.GetMapper ()` - Return mapper that was picked (if any).
- `vtkDataSet = obj.GetDataSet ()` - Get a pointer to the dataset that was picked (if any). If nothing was picked then NULL is returned.
- `vtkProp3DCollection = obj.GetProp3Ds ()` - Return a collection of all the prop 3D's that were intersected by the pick ray. This collection is not sorted.
- `vtkPlanes = obj.GetFrustum ()` - Return the six planes that define the selection frustum. The implicit function defined by the planes evaluates to negative inside and positive outside.
- `vtkPoints = obj.GetClipPoints ()` - Return eight points that define the selection frustum.

39.8 vtkAssembly

39.8.1 Usage

vtkAssembly is an object that groups vtkProp3Ds, its subclasses, and other assemblies into a tree-like hierarchy. The vtkProp3Ds and assemblies can then be transformed together by transforming just the root assembly of the hierarchy.

A vtkAssembly object can be used in place of an vtkProp3D since it is a subclass of vtkProp3D. The difference is that vtkAssembly maintains a list of vtkProp3D instances (its "parts") that form the assembly. Then, any operation that transforms (i.e., scales, rotates, translates) the parent assembly will transform all its parts. Note that this process is recursive: you can create groups consisting of assemblies and/or vtkProp3Ds to arbitrary depth.

To add an assembly to the renderer's list of props, you only need to add the root of the assembly. During rendering, the parts of the assembly are rendered during a hierarchical traversal process.

To create an instance of class vtkAssembly, simply invoke its constructor as follows

```
obj = vtkAssembly
```

39.8.2 Methods

The class vtkAssembly has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAssembly class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAssembly = obj.NewInstance ()`
- `vtkAssembly = obj.SafeDownCast (vtkObject o)`
- `obj.AddPart (vtkProp3D )` - Add a part to the list of parts.
- `obj.RemovePart (vtkProp3D )` - Remove a part from the list of parts,
- `vtkProp3DCollection = obj.GetParts ()` - Return the parts (direct descendants) of this assembly.
- `obj.GetActors (vtkPropCollection )` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.
- `obj.GetVolumes (vtkPropCollection )` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.
- `int = obj.RenderOpaqueGeometry (vtkViewport ren)` - Render this assembly and all its parts. The rendering process is recursive. Note that a mapper need not be defined. If not defined, then no geometry will be drawn for this assembly. This allows you to create "logical" assemblies; that is, assemblies that only serve to group and transform its parts.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport ren)` - Render this assembly and all its parts. The rendering process is recursive. Note that a mapper need not be defined. If not defined, then no geometry will be drawn for this assembly. This allows you to create "logical" assemblies; that is, assemblies that only serve to group and transform its parts.
- `int = obj.RenderVolumetricGeometry (vtkViewport ren)` - Render this assembly and all its parts. The rendering process is recursive. Note that a mapper need not be defined. If not defined, then no geometry will be drawn for this assembly. This allows you to create "logical" assemblies; that is, assemblies that only serve to group and transform its parts.

- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.InitPathTraversal ()` - Methods to traverse the parts of an assembly. Each part (starting from the root) will appear properly transformed and with the correct properties (depending upon the ApplyProperty and ApplyTransform ivars). Note that the part appears as an instance of vtkProp. These methods should be contrasted to those that traverse the list of parts using GetParts(). GetParts() returns a list of children of this assembly, not necessarily with the correct transformation or properties. To use the methods below - first invoke InitPathTraversal() followed by repeated calls to GetNextPath(). GetNextPath() returns a NULL pointer when the list is exhausted.
- `vtkAssemblyPath = obj.GetNextPath ()` - Methods to traverse the parts of an assembly. Each part (starting from the root) will appear properly transformed and with the correct properties (depending upon the ApplyProperty and ApplyTransform ivars). Note that the part appears as an instance of vtkProp. These methods should be contrasted to those that traverse the list of parts using GetParts(). GetParts() returns a list of children of this assembly, not necessarily with the correct transformation or properties. To use the methods below - first invoke InitPathTraversal() followed by repeated calls to GetNextPath(). GetNextPath() returns a NULL pointer when the list is exhausted.
- `int = obj.GetNumberOfPaths ()` - Methods to traverse the parts of an assembly. Each part (starting from the root) will appear properly transformed and with the correct properties (depending upon the ApplyProperty and ApplyTransform ivars). Note that the part appears as an instance of vtkProp. These methods should be contrasted to those that traverse the list of parts using GetParts(). GetParts() returns a list of children of this assembly, not necessarily with the correct transformation or properties. To use the methods below - first invoke InitPathTraversal() followed by repeated calls to GetNextPath(). GetNextPath() returns a NULL pointer when the list is exhausted.
- `obj.GetBounds (double bounds[6])` - Get the bounds for the assembly as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax).
- `double = obj.GetBounds ()` - Get the bounds for the assembly as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax).
- `long = obj.GetMTime ()` - Override default GetMTime method to also consider all of the assembly's parts.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of an assembly. Overloads the virtual vtkProp method.

39.9 vtkAxisActor2D

39.9.1 Usage

vtkAxisActor2D creates an axis with tick marks, labels, and/or a title, depending on the particular instance variable settings. vtkAxisActor2D is a 2D actor; that is, it is drawn on the overlay plane and is not occluded by 3D geometry. To use this class, you typically specify two points defining the start and end points of the line (x-y definition using vtkCoordinate class), the number of labels, and the data range (min,max). You can also control what parts of the axis are visible including the line, the tick marks, the labels, and the title. You can also specify the label format (a printf style format).

This class decides what font size to use and how to locate the labels. It also decides how to create reasonable tick marks and labels. The number of labels and the range of values may not match the number specified, but should be close.

Labels are drawn on the "right" side of the axis. The "right" side is the side of the axis on the right as you move from Position to Position2. The way the labels and title line up with the axis and tick marks depends on whether the line is considered horizontal or vertical.

The vtkActor2D instance variables Position and Position2 are instances of vtkCoordinate. Note that the Position2 is an absolute position in that class (it was by default relative to Position in vtkActor2D).

What this means is that you can specify the axis in a variety of coordinate systems. Also, the axis does not have to be either horizontal or vertical. The tick marks are created so that they are perpendicular to the axis.

Set the text property/attributes of the title and the labels through the vtkTextProperty objects associated to this actor.

To create an instance of class vtkAxisActor2D, simply invoke its constructor as follows

```
obj = vtkAxisActor2D
```

39.9.2 Methods

The class vtkAxisActor2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAxisActor2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkAxisActor2D = obj.NewInstance ()`
- `vtkAxisActor2D = obj.SafeDownCast (vtkObject o)`
- `vtkCoordinate = obj.GetPoint1Coordinate ()` - Specify the position of the first point defining the axis. Note: backward compatibility only, use vtkActor2D's Position instead.
- `obj.SetPoint1 (double x[2])` - Specify the position of the first point defining the axis. Note: backward compatibility only, use vtkActor2D's Position instead.
- `obj.SetPoint1 (double x, double y)` - Specify the position of the first point defining the axis. Note: backward compatibility only, use vtkActor2D's Position instead.
- `vtkCoordinate = obj.GetPoint2Coordinate ()` - Specify the position of the second point defining the axis. Note that the order from Point1 to Point2 controls which side the tick marks are drawn on (ticks are drawn on the right, if visible). Note: backward compatibility only, use vtkActor2D's Position2 instead.
- `obj.SetPoint2 (double x[2])` - Specify the position of the second point defining the axis. Note that the order from Point1 to Point2 controls which side the tick marks are drawn on (ticks are drawn on the right, if visible). Note: backward compatibility only, use vtkActor2D's Position2 instead.
- `obj.SetPoint2 (double x, double y)` - Specify the position of the second point defining the axis. Note that the order from Point1 to Point2 controls which side the tick marks are drawn on (ticks are drawn on the right, if visible). Note: backward compatibility only, use vtkActor2D's Position2 instead.
- `obj.SetRange (double , double )` - Specify the (min,max) axis range. This will be used in the generation of labels, if labels are visible.
- `obj.SetRange (double  a[2])` - Specify the (min,max) axis range. This will be used in the generation of labels, if labels are visible.
- `double = obj. GetRange ()` - Specify the (min,max) axis range. This will be used in the generation of labels, if labels are visible.
- `obj.SetNumberOfLabels (int )` - Set/Get the number of annotation labels to show.
- `int = obj.GetNumberOfLabelsMinValue ()` - Set/Get the number of annotation labels to show.

- `int = obj.GetNumberOfLabelsMaxValue ()` - Set/Get the number of annotation labels to show.
- `int = obj.GetNumberOfLabels ()` - Set/Get the number of annotation labels to show.
- `obj.SetLabelFormat (string )` - Set/Get the format with which to print the labels on the scalar bar.
- `string = obj.GetLabelFormat ()` - Set/Get the format with which to print the labels on the scalar bar.
- `obj.SetAdjustLabels (int )` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.
- `int = obj.GetAdjustLabels ()` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.
- `obj.AdjustLabelsOn ()` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.
- `obj.AdjustLabelsOff ()` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.
- `obj.GetAdjustedRange (double \_arg[2])` - Set/Get the flag that controls whether the labels and ticks are adjusted for "nice" numerical values to make it easier to read the labels. The adjustment is based in the Range instance variable. Call GetAdjustedRange and GetAdjustedNumberOfLabels to get the adjusted range and number of labels.
- `int = obj.GetAdjustedNumberOfLabels ()` - Set/Get the title of the scalar bar actor,
- `obj.SetTitle (string )` - Set/Get the title of the scalar bar actor,
- `string = obj.GetTitle ()` - Set/Get the title of the scalar bar actor,
- `obj.SetTitleTextProperty (vtkTextProperty p)` - Set/Get the title text property.
- `vtkTextProperty = obj.GetTitleTextProperty ()` - Set/Get the title text property.
- `obj.SetLabelTextProperty (vtkTextProperty p)` - Set/Get the labels text property.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - Set/Get the labels text property.
- `obj.SetTickLength (int )` - Set/Get the length of the tick marks (expressed in pixels or display coordinates).
- `int = obj.GetTickLengthMinValue ()` - Set/Get the length of the tick marks (expressed in pixels or display coordinates).
- `int = obj.GetTickLengthMaxValue ()` - Set/Get the length of the tick marks (expressed in pixels or display coordinates).
- `int = obj.GetTickLength ()` - Set/Get the length of the tick marks (expressed in pixels or display coordinates).

- `obj.SetNumberOfMinorTicks (int )` - Number of minor ticks to be displayed between each tick. Default is 0.
- `int = obj.GetNumberOfMinorTicksMinValue ()` - Number of minor ticks to be displayed between each tick. Default is 0.
- `int = obj.GetNumberOfMinorTicksMaxValue ()` - Number of minor ticks to be displayed between each tick. Default is 0.
- `int = obj.GetNumberOfMinorTicks ()` - Number of minor ticks to be displayed between each tick. Default is 0.
- `obj.SetMinorTickLength (int )` - Set/Get the length of the minor tick marks (expressed in pixels or display coordinates).
- `int = obj.GetMinorTickLengthMinValue ()` - Set/Get the length of the minor tick marks (expressed in pixels or display coordinates).
- `int = obj.GetMinorTickLengthMaxValue ()` - Set/Get the length of the minor tick marks (expressed in pixels or display coordinates).
- `int = obj.GetMinorTickLength ()` - Set/Get the length of the minor tick marks (expressed in pixels or display coordinates).
- `obj.SetTickOffset (int )` - Set/Get the offset of the labels (expressed in pixels or display coordinates). The offset is the distance of labels from tick marks or other objects.
- `int = obj.GetTickOffsetMinValue ()` - Set/Get the offset of the labels (expressed in pixels or display coordinates). The offset is the distance of labels from tick marks or other objects.
- `int = obj.GetTickOffsetMaxValue ()` - Set/Get the offset of the labels (expressed in pixels or display coordinates). The offset is the distance of labels from tick marks or other objects.
- `int = obj.GetTickOffset ()` - Set/Get the offset of the labels (expressed in pixels or display coordinates). The offset is the distance of labels from tick marks or other objects.
- `obj.SetAxisVisibility (int )` - Set/Get visibility of the axis line.
- `int = obj.GetAxisVisibility ()` - Set/Get visibility of the axis line.
- `obj.AxisVisibilityOn ()` - Set/Get visibility of the axis line.
- `obj.AxisVisibilityOff ()` - Set/Get visibility of the axis line.
- `obj.SetTickVisibility (int )` - Set/Get visibility of the axis tick marks.
- `int = obj.GetTickVisibility ()` - Set/Get visibility of the axis tick marks.
- `obj.TickVisibilityOn ()` - Set/Get visibility of the axis tick marks.
- `obj.TickVisibilityOff ()` - Set/Get visibility of the axis tick marks.
- `obj.SetLabelVisibility (int )` - Set/Get visibility of the axis labels.
- `int = obj.GetLabelVisibility ()` - Set/Get visibility of the axis labels.
- `obj.LabelVisibilityOn ()` - Set/Get visibility of the axis labels.
- `obj.LabelVisibilityOff ()` - Set/Get visibility of the axis labels.
- `obj.SetTitleVisibility (int )` - Set/Get visibility of the axis title.
- `int = obj.GetTitleVisibility ()` - Set/Get visibility of the axis title.

- `obj.TitleVisibilityOn ()` - Set/Get visibility of the axis title.
- `obj.TitleVisibilityOff ()` - Set/Get visibility of the axis title.
- `obj.SetTitlePosition (double )` - Set/Get position of the axis title. 0 is at the start of the axis whereas 1 is at the end.
- `double = obj.GetTitlePosition ()` - Set/Get position of the axis title. 0 is at the start of the axis whereas 1 is at the end.
- `obj.SetFontFactor (double )` - Set/Get the factor that controls the overall size of the fonts used to label and title the axes. This ivar used in conjunction with the LabelFactor can be used to control font sizes.
- `double = obj.GetFontFactorMinValue ()` - Set/Get the factor that controls the overall size of the fonts used to label and title the axes. This ivar used in conjunction with the LabelFactor can be used to control font sizes.
- `double = obj.GetFontFactorMaxValue ()` - Set/Get the factor that controls the overall size of the fonts used to label and title the axes. This ivar used in conjunction with the LabelFactor can be used to control font sizes.
- `double = obj.GetFontFactor ()` - Set/Get the factor that controls the overall size of the fonts used to label and title the axes. This ivar used in conjunction with the LabelFactor can be used to control font sizes.
- `obj.SetLabelFactor (double )` - Set/Get the factor that controls the relative size of the axis labels to the axis title.
- `double = obj.GetLabelFactorMinValue ()` - Set/Get the factor that controls the relative size of the axis labels to the axis title.
- `double = obj.GetLabelFactorMaxValue ()` - Set/Get the factor that controls the relative size of the axis labels to the axis title.
- `double = obj.GetLabelFactor ()` - Set/Get the factor that controls the relative size of the axis labels to the axis title.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Draw the axis.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Draw the axis.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Does this prop have some translucent polygonal geometry?
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.SetSizeFontRelativeToAxis (int )` - Specify whether to size the fonts relative to the viewport or relative to length of the axis. By default, fonts are resized relative to the axis.
- `int = obj.GetSizeFontRelativeToAxis ()` - Specify whether to size the fonts relative to the viewport or relative to length of the axis. By default, fonts are resized relative to the axis.
- `obj.SizeFontRelativeToAxisOn ()` - Specify whether to size the fonts relative to the viewport or relative to length of the axis. By default, fonts are resized relative to the axis.

- `obj.SizeFontRelativeToAxisOff ()` - Specify whether to size the fonts relative to the viewport or relative to length of the axis. By default, fonts are resized relative to the axis.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of an axis actor. Overloads the virtual vtkProp method.

39.10 vtkCamera

39.10.1 Usage

vtkCamera is a virtual camera for 3D rendering. It provides methods to position and orient the view point and focal point. Convenience methods for moving about the focal point also are provided. More complex methods allow the manipulation of the computer graphics model including view up vector, clipping planes, and camera perspective.

To create an instance of class vtkCamera, simply invoke its constructor as follows

```
obj = vtkCamera
```

39.10.2 Methods

The class vtkCamera has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCamera class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCamera = obj.NewInstance ()`
- `vtkCamera = obj.SafeDownCast (vtkObject o)`
- `obj.SetPosition (double x, double y, double z)` - Set/Get the position of the camera in world coordinates. The default position is (0,0,1).
- `obj.SetPosition (double a[3])` - Set/Get the position of the camera in world coordinates. The default position is (0,0,1).
- `double = obj. GetPosition ()` - Set/Get the position of the camera in world coordinates. The default position is (0,0,1).
- `obj.SetFocalPoint (double x, double y, double z)` - Set/Get the focal of the camera in world coordinates. The default focal point is the origin.
- `obj.SetFocalPoint (double a[3])` - Set/Get the focal of the camera in world coordinates. The default focal point is the origin.
- `double = obj. GetFocalPoint ()` - Set/Get the focal of the camera in world coordinates. The default focal point is the origin.
- `obj.SetViewUp (double vx, double vy, double vz)` - Set/Get the view up direction for the camera. The default is (0,1,0).
- `obj.SetViewUp (double a[3])` - Set/Get the view up direction for the camera. The default is (0,1,0).
- `double = obj. GetViewUp ()` - Set/Get the view up direction for the camera. The default is (0,1,0).
- `obj.OrthogonalizeViewUp ()` - Recompute the ViewUp vector to force it to be perpendicular to camera-¿focalpoint vector. Unless you are going to use Yaw or Azimuth on the camera, there is no need to do this.

- `obj.SetDistance (double )` - Move the focal point so that it is the specified distance from the camera position. This distance must be positive.
- `double = obj.GetDistance ()` - Return the distance from the camera position to the focal point. This distance is positive.
- `double = obj. GetDirectionOfProjection ()` - Get the vector in the direction from the camera position to the focal point. This is usually the opposite of the ViewPlaneNormal, the vector perpendicular to the screen, unless the view is oblique.
- `obj.Dolly (double value)` - Divide the camera's distance from the focal point by the given dolly value. Use a value greater than one to dolly-in toward the focal point, and use a value less than one to dolly-out away from the focal point.
- `obj.SetRoll (double angle)` - Set the roll angle of the camera about the direction of projection.
- `double = obj.GetRoll ()` - Set the roll angle of the camera about the direction of projection.
- `obj.Roll (double angle)` - Rotate the camera about the direction of projection. This will spin the camera about its axis.
- `obj.Azimuth (double angle)` - Rotate the camera about the view up vector centered at the focal point. Note that the view up vector is whatever was set via SetViewUp, and is not necessarily perpendicular to the direction of projection. The result is a horizontal rotation of the camera.
- `obj.Yaw (double angle)` - Rotate the focal point about the view up vector, using the camera's position as the center of rotation. Note that the view up vector is whatever was set via SetViewUp, and is not necessarily perpendicular to the direction of projection. The result is a horizontal rotation of the scene.
- `obj.Elevation (double angle)` - Rotate the camera about the cross product of the negative of the direction of projection and the view up vector, using the focal point as the center of rotation. The result is a vertical rotation of the scene.
- `obj.Pitch (double angle)` - Rotate the focal point about the cross product of the view up vector and the direction of projection, using the camera's position as the center of rotation. The result is a vertical rotation of the camera.
- `obj.SetParallelProjection (int flag)` - Set/Get the value of the ParallelProjection instance variable. This determines if the camera should do a perspective or parallel projection.
- `int = obj.GetParallelProjection ()` - Set/Get the value of the ParallelProjection instance variable. This determines if the camera should do a perspective or parallel projection.
- `obj.ParallelProjectionOn ()` - Set/Get the value of the ParallelProjection instance variable. This determines if the camera should do a perspective or parallel projection.
- `obj.ParallelProjectionOff ()` - Set/Get the value of the ParallelProjection instance variable. This determines if the camera should do a perspective or parallel projection.
- `obj.SetUseHorizontalViewAngle (int flag)` - Set/Get the value of the UseHorizontalViewAngle instance variable. If set, the camera's view angle represents a horizontal view angle, rather than the default vertical view angle. This is useful if the application uses a display device which whose specs indicate a particular horizontal view angle, or if the application varies the window height but wants to keep the perspective transform unchanges.
- `int = obj.GetUseHorizontalViewAngle ()` - Set/Get the value of the UseHorizontalViewAngle instance variable. If set, the camera's view angle represents a horizontal view angle, rather than the default vertical view angle. This is useful if the application uses a display device which whose specs indicate a particular horizontal view angle, or if the application varies the window height but wants to keep the perspective transform unchanges.

- `obj.UseHorizontalViewAngleOn ()` - Set/Get the value of the UseHorizontalViewAngle instance variable. If set, the camera's view angle represents a horizontal view angle, rather than the default vertical view angle. This is useful if the application uses a display device which whose specs indicate a particular horizontal view angle, or if the application varies the window height but wants to keep the perspective transform unchanges.

- `obj.UseHorizontalViewAngleOff ()` - Set/Get the value of the UseHorizontalViewAngle instance variable. If set, the camera's view angle represents a horizontal view angle, rather than the default vertical view angle. This is useful if the application uses a display device which whose specs indicate a particular horizontal view angle, or if the application varies the window height but wants to keep the perspective transform unchanges.

- `obj.SetViewAngle (double angle)` - Set/Get the camera view angle, which is the angular height of the camera view measured in degrees. The default angle is 30 degrees. This method has no effect in parallel projection mode. The formula for setting the angle up for perfect perspective viewing is: angle = 2*atan((h/2)/d) where h is the height of the RenderWindow (measured by holding a ruler up to your screen) and d is the distance from your eyes to the screen.

- `double = obj.GetViewAngle ()` - Set/Get the camera view angle, which is the angular height of the camera view measured in degrees. The default angle is 30 degrees. This method has no effect in parallel projection mode. The formula for setting the angle up for perfect perspective viewing is: angle = 2*atan((h/2)/d) where h is the height of the RenderWindow (measured by holding a ruler up to your screen) and d is the distance from your eyes to the screen.

- `obj.SetParallelScale (double scale)` - Set/Get the scaling used for a parallel projection, i.e. the height of the viewport in world-coordinate distances. The default is 1. Note that the "scale" parameter works as an "inverse scale" — larger numbers produce smaller images. This method has no effect in perspective projection mode.

- `double = obj.GetParallelScale ()` - Set/Get the scaling used for a parallel projection, i.e. the height of the viewport in world-coordinate distances. The default is 1. Note that the "scale" parameter works as an "inverse scale" — larger numbers produce smaller images. This method has no effect in perspective projection mode.

- `obj.Zoom (double factor)` - In perspective mode, decrease the view angle by the specified factor. In parallel mode, decrease the parallel scale by the specified factor. A value greater than 1 is a zoom-in, a value less than 1 is a zoom-out.

- `obj.SetClippingRange (double dNear, double dFar)` - Set/Get the location of the near and far clipping planes along the direction of projection. Both of these values must be positive. How the clipping planes are set can have a large impact on how well z-buffering works. In particular the front clipping plane can make a very big difference. Setting it to 0.01 when it really could be 1.0 can have a big impact on your z-buffer resolution farther away. The default clipping range is (0.1,1000).

- `obj.SetClippingRange (double a[2])` - Set/Get the location of the near and far clipping planes along the direction of projection. Both of these values must be positive. How the clipping planes are set can have a large impact on how well z-buffering works. In particular the front clipping plane can make a very big difference. Setting it to 0.01 when it really could be 1.0 can have a big impact on your z-buffer resolution farther away. The default clipping range is (0.1,1000).

- `double = obj. GetClippingRange ()` - Set/Get the location of the near and far clipping planes along the direction of projection. Both of these values must be positive. How the clipping planes are set can have a large impact on how well z-buffering works. In particular the front clipping plane can make a very big difference. Setting it to 0.01 when it really could be 1.0 can have a big impact on your z-buffer resolution farther away. The default clipping range is (0.1,1000).

- `obj.SetThickness (double )` - Set the distance between clipping planes. This method adjusts the far clipping plane to be set a distance 'thickness' beyond the near clipping plane.

- `double = obj.GetThickness ()` - Set the distance between clipping planes. This method adjusts the far clipping plane to be set a distance 'thickness' beyond the near clipping plane.

- `obj.SetWindowCenter (double x, double y)` - Set/Get the center of the window in viewport coordinates. The viewport coordinate range is ([-1,+1],[-1,+1]). This method is for if you have one window which consists of several viewports, or if you have several screens which you want to act together as one large screen.

- `double = obj. GetWindowCenter ()` - Set/Get the center of the window in viewport coordinates. The viewport coordinate range is ([-1,+1],[-1,+1]). This method is for if you have one window which consists of several viewports, or if you have several screens which you want to act together as one large screen.

- `obj.SetObliqueAngles (double alpha, double beta)` - Get/Set the oblique viewing angles. The first angle, alpha, is the angle (measured from the horizontal) that rays along the direction of projection will follow once projected onto the 2D screen. The second angle, beta, is the angle between the view plane and the direction of projection. This creates a shear transform x' = x + dz*cos(alpha)/tan(beta), y' = dz*sin(alpha)/tan(beta) where dz is the distance of the point from the focal plane. The angles are (45,90) by default. Oblique projections commonly use (30,63.435).

- `obj.ApplyTransform (vtkTransform t)` - Apply a transform to the camera. The camera position, focal-point, and view-up are re-calculated using the transform's matrix to multiply the old points by the new transform.

- `double = obj. GetViewPlaneNormal ()` - Get the ViewPlaneNormal. This vector will point opposite to the direction of projection, unless you have created an sheared output view using SetViewShear/SetObliqueAngles.

- `obj.SetViewShear (double dxdz, double dydz, double center)` - Set/get the shear transform of the viewing frustum. Parameters are dx/dz, dy/dz, and center. center is a factor that describes where to shear around. The distance dshear from the camera where no shear occurs is given by (dshear = center * FocalDistance).

- `obj.SetViewShear (double d[3])` - Set/get the shear transform of the viewing frustum. Parameters are dx/dz, dy/dz, and center. center is a factor that describes where to shear around. The distance dshear from the camera where no shear occurs is given by (dshear = center * FocalDistance).

- `double = obj. GetViewShear ()` - Set/get the shear transform of the viewing frustum. Parameters are dx/dz, dy/dz, and center. center is a factor that describes where to shear around. The distance dshear from the camera where no shear occurs is given by (dshear = center * FocalDistance).

- `obj.SetEyeAngle (double )` - Set/Get the separation between eyes (in degrees). This is used when generating stereo images.

- `double = obj.GetEyeAngle ()` - Set/Get the separation between eyes (in degrees). This is used when generating stereo images.

- `obj.SetFocalDisk (double )` - Set the size of the cameras lens in world coordinates. This is only used when the renderer is doing focal depth rendering. When that is being done the size of the focal disk will effect how significant the depth effects will be.

- `double = obj.GetFocalDisk ()` - Set the size of the cameras lens in world coordinates. This is only used when the renderer is doing focal depth rendering. When that is being done the size of the focal disk will effect how significant the depth effects will be.

- `vtkMatrix4x4 = obj.GetViewTransformMatrix ()` - Return the matrix of the view transform. The ViewTransform depends on only three ivars: the Position, the FocalPoint, and the ViewUp vector. All the other methods are there simply for the sake of the users' convenience.

- `vtkTransform = obj.GetViewTransformObject ()` - Return the projection transform matrix, which converts from camera coordinates to viewport coordinates. The 'aspect' is the width/height for the viewport, and the nearz and farz are the Z-buffer values that map to the near and far clipping planes. The viewport coordinates of a point located inside the frustum are in the range ([-1,+1],[-1,+1],[nearz,farz]). WARNING: the name of the method is wrong, it should be GetProjectionTransformMatrix() (it is used also in parallel projection) @deprecated Replaced by GetProjectionTransformMatrix() as of VTK 5.4.

- `vtkMatrix4x4 = obj.GetPerspectiveTransformMatrix (double aspect, double nearz, double farz)` - Return the projection transform matrix, which converts from camera coordinates to viewport coordinates. The 'aspect' is the width/height for the viewport, and the nearz and farz are the Z-buffer values that map to the near and far clipping planes. The viewport coordinates of a point located inside the frustum are in the range ([-1,+1],[-1,+1],[nearz,farz]). WARNING: the name of the method is wrong, it should be GetProjectionTransformMatrix() (it is used also in parallel projection) @deprecated Replaced by GetProjectionTransformMatrix() as of VTK 5.4.

- `vtkMatrix4x4 = obj.GetProjectionTransformMatrix (double aspect, double nearz, double farz)` - Return the projection transform matrix, which converts from camera coordinates to viewport coordinates. The 'aspect' is the width/height for the viewport, and the nearz and farz are the Z-buffer values that map to the near and far clipping planes. The viewport coordinates of a point located inside the frustum are in the range ([-1,+1],[-1,+1],[nearz,farz]).

- `vtkPerspectiveTransform = obj.GetProjectionTransformObject (double aspect, double nearz, double far` - Return the projection transform matrix, which converts from camera coordinates to viewport coordinates. The 'aspect' is the width/height for the viewport, and the nearz and farz are the Z-buffer values that map to the near and far clipping planes. The viewport coordinates of a point located inside the frustum are in the range ([-1,+1],[-1,+1],[nearz,farz]).

- `vtkMatrix4x4 = obj.GetCompositePerspectiveTransformMatrix (double aspect, double nearz, double farz` - Return the concatenation of the ViewTransform and the ProjectionTransform. This transform will convert world coordinates to viewport coordinates. The 'aspect' is the width/height for the viewport, and the nearz and farz are the Z-buffer values that map to the near and far clipping planes. The viewport coordinates of a point located inside the frustum are in the range ([-1,+1],[-1,+1],[nearz,farz]). WARNING: the name of the method is wrong, it should be GetCompositeProjectionTransformMatrix() (it is used also in parallel projection) @deprecated Replaced by GetCompositeProjectionTransformMatrix() as of VTK 5.4.

- `vtkMatrix4x4 = obj.GetCompositeProjectionTransformMatrix (double aspect, double nearz, double farz)` - Return the concatenation of the ViewTransform and the ProjectionTransform. This transform will convert world coordinates to viewport coordinates. The 'aspect' is the width/height for the viewport, and the nearz and farz are the Z-buffer values that map to the near and far clipping planes. The viewport coordinates of a point located inside the frustum are in the range ([-1,+1],[-1,+1],[nearz,farz]).

- `obj.SetUserViewTransform (vtkHomogeneousTransform transform)` - In addition to the instance variables such as position and orientation, you can add an additional transformation for your own use. This transformation is concatenated to the camera's ViewTransform

- `vtkHomogeneousTransform = obj.GetUserViewTransform ()` - In addition to the instance variables such as position and orientation, you can add an additional transformation for your own use. This transformation is concatenated to the camera's ViewTransform

- `obj.SetUserTransform (vtkHomogeneousTransform transform)` - In addition to the instance variables such as position and orientation, you can add an additional transformation for your own use. This transformation is concatenated to the camera's ProjectionTransform

- `vtkHomogeneousTransform = obj.GetUserTransform ()` - In addition to the instance variables such as position and orientation, you can add an additional transformation for your own use. This transformation is concatenated to the camera's ProjectionTransform

- `obj.Render (vtkRenderer )` - Return the MTime that concerns recomputing the view rays of the camera.
- `long = obj.GetViewingRaysMTime ()` - Return the MTime that concerns recomputing the view rays of the camera.
- `obj.ViewingRaysModified ()` - Mark that something has changed which requires the view rays to be recomputed.
- `obj.GetFrustumPlanes (double aspect, double planes[24])` - Get the plane equations that bound the view frustum. The plane normals point inward. The planes array contains six plane equations of the form (Ax+By+Cz+D=0), the first four values are (A,B,C,D) which repeats for each of the planes. The planes are given in the following order: -x,+x,-y,+y,-z,+z. Warning: it means left,right,bottom,top,far,near (NOT near,far) The aspect of the viewport is needed to correctly compute the planes
- `double = obj.GetOrientation ()` - Get the orientation of the camera.
- `double = obj.GetOrientationWXYZ ()` - Get the orientation of the camera.
- `obj.SetViewPlaneNormal (double x, double y, double z)` - @deprecated The view plane normal is automatically set from the DirectionOfProjection according to the ViewShear.
- `obj.SetViewPlaneNormal (double a[3])` - @deprecated The view plane normal is automatically set from the DirectionOfProjection according to the ViewShear.
- `obj.ComputeViewPlaneNormal ()` - This method is called automatically whenever necessary, it should never be used outside of vtkCamera.cxx.
- `vtkMatrix4x4 = obj.GetCameraLightTransformMatrix ()` - Returns a transformation matrix for a coordinate frame attached to the camera, where the camera is located at (0, 0, 1) looking at the focal point at (0, 0, 0), with up being (0, 1, 0).
- `obj.UpdateViewport (vtkRenderer )` - Set the Left Eye setting
- `obj.SetLeftEye (int )` - Set the Left Eye setting
- `int = obj.GetLeftEye ()` - Set the Left Eye setting
- `obj.ShallowCopy (vtkCamera source)` - Copy the properties of 'source' into 'this'. Copy pointers of matrices.
- `obj.DeepCopy (vtkCamera source)` - Copy the properties of 'source' into 'this'. Copy the contents of the matrices.

39.11 vtkCameraActor

39.11.1 Usage

vtkCameraActor is an actor used to represent a camera by its wireframe frustum.

To create an instance of class vtkCameraActor, simply invoke its constructor as follows

```
obj = vtkCameraActor
```

39.11.2 Methods

The class vtkCameraActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCameraActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCameraActor = obj.NewInstance ()`
- `vtkCameraActor = obj.SafeDownCast (vtkObject o)`
- `obj.SetCamera (vtkCamera camera)` - The camera to represent. Initial value is NULL.
- `vtkCamera = obj.GetCamera ()` - The camera to represent. Initial value is NULL.
- `obj.SetWidthByHeightRatio (double )` - Ratio between the width and the height of the frustum. Initial value is 1.0 (square)
- `double = obj.GetWidthByHeightRatio ()` - Ratio between the width and the height of the frustum. Initial value is 1.0 (square)
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry? No.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `long = obj.GetMTime ()` - Get the actors mtime plus consider its properties and texture if set.
- `vtkProperty = obj.GetProperty ()` - Get property of the internal actor.
- `obj.SetProperty (vtkProperty p)` - Set property of the internal actor.

39.12 vtkCameraInterpolator

39.12.1 Usage

This class is used to interpolate a series of cameras to update a specified camera. Either linear interpolation or spline interpolation may be used. The instance variables currently interpolated include position, focal point, view up, view angle, parallel scale, and clipping range.

To use this class, specify the type of interpolation to use, and add a series of cameras at various times "t" to the list of cameras from which to interpolate. Then to interpolate in between cameras, simply invoke the function InterpolateCamera(t,camera) where "camera" is the camera to be updated with interpolated values. Note that "t" should be in the range (min,max) times specified with the AddCamera() method. If outside this range, the interpolation is clamped. This class copies the camera information (as compared to referencing the cameras) so you do not need to keep separate instances of the camera around for each camera added to the list of cameras to interpolate.

To create an instance of class vtkCameraInterpolator, simply invoke its constructor as follows

```
obj = vtkCameraInterpolator
```

39.12.2 Methods

The class vtkCameraInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCameraInterpolator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCameraInterpolator = obj.NewInstance ()`
- `vtkCameraInterpolator = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfCameras ()` - Return the number of cameras in the list of cameras.
- `double = obj.GetMinimumT ()` - Obtain some information about the interpolation range. The numbers returned are undefined if the list of cameras is empty.
- `double = obj.GetMaximumT ()` - Obtain some information about the interpolation range. The numbers returned are undefined if the list of cameras is empty.
- `obj.Initialize ()` - Clear the list of cameras.
- `obj.AddCamera (double t, vtkCamera camera)` - Add another camera to the list of cameras defining the camera function. Note that using the same time t value more than once replaces the previous camera value at t. At least one camera must be added to define a function.
- `obj.RemoveCamera (double t)` - Delete the camera at a particular parameter t. If there is no camera defined at location t, then the method does nothing.
- `obj.InterpolateCamera (double t, vtkCamera camera)` - Interpolate the list of cameras and determine a new camera (i.e., fill in the camera provided). If t is outside the range of (min,max) values, then t is clamped to lie within this range.
- `obj.SetInterpolationType (int )` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the instance variable interpolators (i.e., position, focal point, clipping range, orientation, etc.) interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.
- `int = obj.GetInterpolationTypeMinValue ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the instance variable interpolators (i.e., position, focal point, clipping range, orientation, etc.) interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.
- `int = obj.GetInterpolationTypeMaxValue ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the instance variable interpolators (i.e., position, focal point, clipping range, orientation, etc.) interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `int = obj.GetInterpolationType ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the instance variable interpolators (i.e., position, focal point, clipping range, orientation, etc.) interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `obj.SetInterpolationTypeToLinear ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the instance variable interpolators (i.e., position, focal point, clipping range, orientation, etc.) interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `obj.SetInterpolationTypeToSpline ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the instance variable interpolators (i.e., position, focal point, clipping range, orientation, etc.) interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `obj.SetInterpolationTypeToManual ()` - Set/Get the tuple interpolator used to interpolate the position portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.

- `obj.SetPositionInterpolator (vtkTupleInterpolator )` - Set/Get the tuple interpolator used to interpolate the position portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.

- `vtkTupleInterpolator = obj.GetPositionInterpolator ()` - Set/Get the tuple interpolator used to interpolate the position portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.

- `obj.SetFocalPointInterpolator (vtkTupleInterpolator )` - Set/Get the tuple interpolator used to interpolate the focal point portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.

- `vtkTupleInterpolator = obj.GetFocalPointInterpolator ()` - Set/Get the tuple interpolator used to interpolate the focal point portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.

- `obj.SetViewUpInterpolator (vtkTupleInterpolator )` - Set/Get the tuple interpolator used to interpolate the view up portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.

- `vtkTupleInterpolator = obj.GetViewUpInterpolator ()` - Set/Get the tuple interpolator used to interpolate the view up portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.

- `obj.SetViewAngleInterpolator (vtkTupleInterpolator )` - Set/Get the tuple interpolator used to interpolate the view angle portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.

- `vtkTupleInterpolator = obj.GetViewAngleInterpolator ()` - Set/Get the tuple interpolator used to interpolate the view angle portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.
- `obj.SetParallelScaleInterpolator (vtkTupleInterpolator )` - Set/Get the tuple interpolator used to interpolate the parallel scale portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.
- `vtkTupleInterpolator = obj.GetParallelScaleInterpolator ()` - Set/Get the tuple interpolator used to interpolate the parallel scale portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.
- `obj.SetClippingRangeInterpolator (vtkTupleInterpolator )` - Set/Get the tuple interpolator used to interpolate the clipping range portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.
- `vtkTupleInterpolator = obj.GetClippingRangeInterpolator ()` - Set/Get the tuple interpolator used to interpolate the clipping range portion of the camera. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances directly.
- `long = obj.GetMTime ()` - Override GetMTime() because we depend on the interpolators which may be modified outside of this class.

39.13 vtkCameraPass

39.13.1 Usage

Render the camera.

It setups the projection and modelview matrices and can clear the background It calls its delegate once. After its delegate returns, it restore the modelview matrix stack.

Its delegate is usually set to a vtkSequencePass with a vtkLigthsPass and a list of passes for the geometry.

To create an instance of class vtkCameraPass, simply invoke its constructor as follows

```
obj = vtkCameraPass
```

39.13.2 Methods

The class vtkCameraPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCameraPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCameraPass = obj.NewInstance ()`
- `vtkCameraPass = obj.SafeDownCast (vtkObject o)`
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Release graphics resources and ask components to release their own resources.

- `vtkRenderPass = obj.GetDelegatePass ()` - Delegate for rendering the geometry. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkSequencePass with a vtkLigthsPass and a list of passes for the geometry. Initial value is a NULL pointer.

- `obj.SetDelegatePass (vtkRenderPass delegatePass)` - Delegate for rendering the geometry. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkSequencePass with a vtkLigthsPass and a list of passes for the geometry. Initial value is a NULL pointer.

39.14 vtkCellCenterDepthSort

39.14.1 Usage

vtkCellCenterDepthSort is a simple and fast implementation of depth sort, but it only provides approximate results. The sorting algorithm finds the centroids of all the cells. It then performs the dot product of the centroids against a vector pointing in the direction of the camera transformed into object space. It then performs an ordinary sort on the result.

To create an instance of class vtkCellCenterDepthSort, simply invoke its constructor as follows

```
obj = vtkCellCenterDepthSort
```

39.14.2 Methods

The class vtkCellCenterDepthSort has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellCenterDepthSort class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkCellCenterDepthSort = obj.NewInstance ()`

- `vtkCellCenterDepthSort = obj.SafeDownCast (vtkObject o)`

- `obj.InitTraversal ()`

- `vtkIdTypeArray = obj.GetNextCells ()`

39.15 vtkCellPicker

39.15.1 Usage

vtkCellPicker will shoot a ray into a 3D scene and return information about the first object that the ray hits. It works for all Prop3Ds. For vtkVolume objects, it shoots a ray into the volume and returns the point where the ray intersects an isosurface of a chosen opacity. For vtkImageActor objects, it intersects the ray with the displayed slice. For vtkActor objects, it intersects the actor's polygons. If the object's mapper has ClippingPlanes, then it takes the clipping into account, and will return the Id of the clipping plane that was intersected. For all prop types, it returns point and cell information, plus the normal of the surface that was intersected at the pick position. For volumes and images, it also returns (i,j,k) coordinates for the point and the cell that were picked.

To create an instance of class vtkCellPicker, simply invoke its constructor as follows

```
obj = vtkCellPicker
```

39.15.2 Methods

The class vtkCellPicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCellPicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCellPicker = obj.NewInstance ()`
- `vtkCellPicker = obj.SafeDownCast (vtkObject o)`
- `int = obj.Pick (double selectionX, double selectionY, double selectionZ, vtkRenderer renderer)` - Perform pick operation with selection point provided. Normally the first two values are the (x,y) pixel coordinates for the pick, and the third value is z=0. The return value will be non-zero if something was successfully picked.
- `obj.AddLocator (vtkAbstractCellLocator locator)` - Add a locator for one of the data sets that will be included in the scene. You must set up the locator with exactly the same data set that was input to the mapper of one or more of the actors in the scene. As well, you must either build the locator before doing the pick, or you must turn on LazyEvaluation in the locator to make it build itself on the first pick. Note that if you try to add the same locator to the picker twice, the second addition will be ignored.
- `obj.RemoveLocator (vtkAbstractCellLocator locator)` - Remove a locator that was previously added. If you try to remove a nonexistent locator, then nothing will happen and no errors will be raised.
- `obj.RemoveAllLocators ()` - Remove all locators associated with this picker.
- `obj.SetVolumeOpacityIsovalue (double )` - Set the opacity isovalue to use for defining volume surfaces. The pick will occur at the location along the pick ray where the opacity of the volume is equal to this isovalue. If you want to do the pick based on an actual data isovalue rather than the opacity, then pass the data value through the scalar opacity function before using this method.
- `double = obj.GetVolumeOpacityIsovalue ()` - Set the opacity isovalue to use for defining volume surfaces. The pick will occur at the location along the pick ray where the opacity of the volume is equal to this isovalue. If you want to do the pick based on an actual data isovalue rather than the opacity, then pass the data value through the scalar opacity function before using this method.
- `obj.SetUseVolumeGradientOpacity (int )` - Use the product of the scalar and gradient opacity functions when computing the opacity isovalue, instead of just using the scalar opacity. This parameter is only relevant to volume picking and is off by default.
- `obj.UseVolumeGradientOpacityOn ()` - Use the product of the scalar and gradient opacity functions when computing the opacity isovalue, instead of just using the scalar opacity. This parameter is only relevant to volume picking and is off by default.
- `obj.UseVolumeGradientOpacityOff ()` - Use the product of the scalar and gradient opacity functions when computing the opacity isovalue, instead of just using the scalar opacity. This parameter is only relevant to volume picking and is off by default.
- `int = obj.GetUseVolumeGradientOpacity ()` - Use the product of the scalar and gradient opacity functions when computing the opacity isovalue, instead of just using the scalar opacity. This parameter is only relevant to volume picking and is off by default.

- `obj.SetPickClippingPlanes (int )` - The PickClippingPlanes setting controls how clipping planes are handled by the pick. If it is On, then the clipping planes become pickable objects, even though they are usually invisible. This means that if the pick ray intersects a clipping plane before it hits anything else, the pick will stop at that clipping plane. The GetProp3D() and GetMapper() methods will return the Prop3D and Mapper that the clipping plane belongs to. The GetClippingPlaneId() method will return the index of the clipping plane so that you can retrieve it from the mapper, or -1 if no clipping plane was picked. The picking of vtkImageActors is not influenced by this setting, since they have no clipping planes.

- `obj.PickClippingPlanesOn ()` - The PickClippingPlanes setting controls how clipping planes are handled by the pick. If it is On, then the clipping planes become pickable objects, even though they are usually invisible. This means that if the pick ray intersects a clipping plane before it hits anything else, the pick will stop at that clipping plane. The GetProp3D() and GetMapper() methods will return the Prop3D and Mapper that the clipping plane belongs to. The GetClippingPlaneId() method will return the index of the clipping plane so that you can retrieve it from the mapper, or -1 if no clipping plane was picked. The picking of vtkImageActors is not influenced by this setting, since they have no clipping planes.

- `obj.PickClippingPlanesOff ()` - The PickClippingPlanes setting controls how clipping planes are handled by the pick. If it is On, then the clipping planes become pickable objects, even though they are usually invisible. This means that if the pick ray intersects a clipping plane before it hits anything else, the pick will stop at that clipping plane. The GetProp3D() and GetMapper() methods will return the Prop3D and Mapper that the clipping plane belongs to. The GetClippingPlaneId() method will return the index of the clipping plane so that you can retrieve it from the mapper, or -1 if no clipping plane was picked. The picking of vtkImageActors is not influenced by this setting, since they have no clipping planes.

- `int = obj.GetPickClippingPlanes ()` - The PickClippingPlanes setting controls how clipping planes are handled by the pick. If it is On, then the clipping planes become pickable objects, even though they are usually invisible. This means that if the pick ray intersects a clipping plane before it hits anything else, the pick will stop at that clipping plane. The GetProp3D() and GetMapper() methods will return the Prop3D and Mapper that the clipping plane belongs to. The GetClippingPlaneId() method will return the index of the clipping plane so that you can retrieve it from the mapper, or -1 if no clipping plane was picked. The picking of vtkImageActors is not influenced by this setting, since they have no clipping planes.

- `int = obj.GetClippingPlaneId ()` - Get the index of the clipping plane that was intersected during the pick. This will be set regardless of whether PickClippingPlanes is On, all that is required is that the pick intersected a clipping plane of the Prop3D that was picked. The result will be -1 if the Prop3D that was picked has no clipping planes, or if the ray didn't intersect the planes.

- `double = obj. GetPickNormal ()` - Return the normal of the picked surface at the PickPosition. If no surface was picked, then a vector pointing back at the camera is returned.

- `double = obj. GetMapperNormal ()` - Return the normal of the surface at the PickPosition in mapper coordinates. The result is undefined if no prop was picked.

- `int = obj. GetPointIJK ()` - Get the structured coordinates of the point at the PickPosition. Only valid for image actors and volumes with vtkImageData.

- `int = obj. GetCellIJK ()` - Get the structured coordinates of the cell at the PickPosition. Only valid for image actors and volumes with vtkImageData. Combine this with the PCoords to get the position within the cell.

- `vtkIdType = obj.GetPointId ()` - Get the id of the picked point. If PointId = -1, nothing was picked. This point will be a member of any cell that is picked.

- `vtkIdType = obj.GetCellId ()` - Get the id of the picked cell. If CellId = -1, nothing was picked.

- `int = obj.GetSubId ()` - Get the subId of the picked cell. This is useful, for example, if the data is made of triangle strips. If SubId = -1, nothing was picked.

- `double = obj. GetPCoords ()` - Get the parametric coordinates of the picked cell. Only valid if a prop was picked. The PCoords can be used to compute the weights that are needed to interpolate data values within the cell.

- `vtkTexture = obj.GetTexture ()` - Get the texture that was picked. This will always be set if the picked prop has a texture, and will always be null otherwise.

- `obj.SetPickTextureData (int )` - If this is "On" and if the picked prop has a texture, then the data returned by GetDataSet() will be the texture's data instead of the mapper's data. The GetPointId(), GetCellId(), GetPCoords() etc. will all return information for use with the texture's data. If the picked prop does not have any texture, then GetDataSet() will return the mapper's data instead and GetPointId() etc. will return information related to the mapper's data. The default value of PickTextureData is "Off".

- `obj.PickTextureDataOn ()` - If this is "On" and if the picked prop has a texture, then the data returned by GetDataSet() will be the texture's data instead of the mapper's data. The GetPointId(), GetCellId(), GetPCoords() etc. will all return information for use with the texture's data. If the picked prop does not have any texture, then GetDataSet() will return the mapper's data instead and GetPointId() etc. will return information related to the mapper's data. The default value of PickTextureData is "Off".

- `obj.PickTextureDataOff ()` - If this is "On" and if the picked prop has a texture, then the data returned by GetDataSet() will be the texture's data instead of the mapper's data. The GetPointId(), GetCellId(), GetPCoords() etc. will all return information for use with the texture's data. If the picked prop does not have any texture, then GetDataSet() will return the mapper's data instead and GetPointId() etc. will return information related to the mapper's data. The default value of PickTextureData is "Off".

- `int = obj.GetPickTextureData ()` - If this is "On" and if the picked prop has a texture, then the data returned by GetDataSet() will be the texture's data instead of the mapper's data. The GetPointId(), GetCellId(), GetPCoords() etc. will all return information for use with the texture's data. If the picked prop does not have any texture, then GetDataSet() will return the mapper's data instead and GetPointId() etc. will return information related to the mapper's data. The default value of PickTextureData is "Off".

39.16 vtkChooserPainter

39.16.1 Usage

This painter does not actually do any painting. Instead, it picks other painters based on the current state of itself and its poly data. It then delegates the work to these other painters.

To create an instance of class vtkChooserPainter, simply invoke its constructor as follows

```
obj = vtkChooserPainter
```

39.16.2 Methods

The class vtkChooserPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkChooserPainter class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkChooserPainter = obj.NewInstance ()`
- `vtkChooserPainter = obj.SafeDownCast (vtkObject o)`
- `obj.SetVertPainter (vtkPolyDataPainter )`
- `obj.SetLinePainter (vtkPolyDataPainter )`
- `obj.SetPolyPainter (vtkPolyDataPainter )`
- `obj.SetStripPainter (vtkPolyDataPainter )`
- `obj.SetUseLinesPainterForWireframes (int )` - When set, the lines painter is used for drawing wireframes (off by default, except on Mac, where it's on by default).
- `int = obj.GetUseLinesPainterForWireframes ()` - When set, the lines painter is used for drawing wireframes (off by default, except on Mac, where it's on by default).
- `obj.UseLinesPainterForWireframesOn ()` - When set, the lines painter is used for drawing wireframes (off by default, except on Mac, where it's on by default).
- `obj.UseLinesPainterForWireframesOff ()` - When set, the lines painter is used for drawing wireframes (off by default, except on Mac, where it's on by default).

39.17 vtkClearZPass

39.17.1 Usage

Clear the depth buffer with a given value.

To create an instance of class vtkClearZPass, simply invoke its constructor as follows

```
obj = vtkClearZPass
```

39.17.2 Methods

The class vtkClearZPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkClearZPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkClearZPass = obj.NewInstance ()`
- `vtkClearZPass = obj.SafeDownCast (vtkObject o)`
- `obj.SetDepth (double )` - Set/Get the depth value. Initial value is 1.0 (farest).
- `double = obj.GetDepthMinValue ()` - Set/Get the depth value. Initial value is 1.0 (farest).
- `double = obj.GetDepthMaxValue ()` - Set/Get the depth value. Initial value is 1.0 (farest).
- `double = obj.GetDepth ()` - Set/Get the depth value. Initial value is 1.0 (farest).

39.18 vtkCoincidentTopologyResolutionPainter

39.18.1 Usage

Provides the ability to shift the z-buffer to resolve coincident topology. For example, if you'd like to draw a mesh with some edges a different color, and the edges lie on the mesh, this feature can be useful to get nice looking lines.

To create an instance of class vtkCoincidentTopologyResolutionPainter, simply invoke its constructor as follows

```
obj = vtkCoincidentTopologyResolutionPainter
```

39.18.2 Methods

The class vtkCoincidentTopologyResolutionPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCoincidentTopologyResolutionPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCoincidentTopologyResolutionPainter = obj.NewInstance ()`
- `vtkCoincidentTopologyResolutionPainter = obj.SafeDownCast (vtkObject o)`

39.19 vtkColorMaterialHelper

39.19.1 Usage

vtkColorMaterialHelper is a helper to assist in simulating the ColorMaterial behaviour of the default OpenGL pipeline. Look at vtkColorMaterialHelper_s for available GLSL functions.

To create an instance of class vtkColorMaterialHelper, simply invoke its constructor as follows

```
obj = vtkColorMaterialHelper
```

39.19.2 Methods

The class vtkColorMaterialHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkColorMaterialHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkColorMaterialHelper = obj.NewInstance ()`
- `vtkColorMaterialHelper = obj.SafeDownCast (vtkObject o)`
- `obj.PrepareForRendering ()` - Prepares the shader i.e. reads color material paramters state from OpenGL. This must be called before the shader is bound.
- `obj.Render ()` - Uploads any uniforms needed. This must be called only after the shader has been bound, but before rendering the geometry.

39.20 vtkCompositePainter

39.20.1 Usage

vtkCompositePainter iterates over the leaves in a composite datasets. This painter can also handle the case when the dataset is not a composite dataset.

To create an instance of class vtkCompositePainter, simply invoke its constructor as follows

```
obj = vtkCompositePainter
```

39.20.2 Methods

The class vtkCompositePainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositePainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositePainter = obj.NewInstance ()`
- `vtkCompositePainter = obj.SafeDownCast (vtkObject o)`
- `vtkDataObject = obj.GetOutput ()` - Get the output data object from this painter. The default implementation simply forwards the input data object as the output.

39.21 vtkCompositePolyDataMapper

39.21.1 Usage

This class uses a set of vtkPolyDataMappers to render input data which may be hierarchical. The input to this mapper may be either vtkPolyData or a vtkCompositeDataSet built from polydata. If something other than vtkPolyData is encountered, an error message will be produced.

To create an instance of class vtkCompositePolyDataMapper, simply invoke its constructor as follows

```
obj = vtkCompositePolyDataMapper
```

39.21.2 Methods

The class vtkCompositePolyDataMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositePolyDataMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositePolyDataMapper = obj.NewInstance ()`
- `vtkCompositePolyDataMapper = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer ren, vtkActor a)` - Standard method for rendering a mapper. This method will be called by the actor.
- `double = obj.GetBounds ()` - Standard vtkProp method to get 3D bounds of a 3D prop
- `obj.GetBounds (double bounds[6])` - Standard vtkProp method to get 3D bounds of a 3D prop
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release the underlying resources associated with this mapper

39.22 vtkCompositePolyDataMapper2

39.22.1 Usage

vtkCompositePolyDataMapper2 is similar to vtkCompositePolyDataMapper except that instead of creating individual mapper for each block in the composite dataset, it iterates over the blocks internally.

To create an instance of class vtkCompositePolyDataMapper2, simply invoke its constructor as follows

```
obj = vtkCompositePolyDataMapper2
```

39.22.2 Methods

The class vtkCompositePolyDataMapper2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCompositePolyDataMapper2 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCompositePolyDataMapper2 = obj.NewInstance ()`
- `vtkCompositePolyDataMapper2 = obj.SafeDownCast (vtkObject o)`
- `obj.RenderPiece (vtkRenderer ren, vtkActor act)` - Implemented by sub classes. Actual rendering is done here.
- `double = obj.GetBounds ()` - Standard vtkProp method to get 3D bounds of a 3D prop
- `obj.GetBounds (double bounds[6])` - Standard vtkProp method to get 3D bounds of a 3D prop
- `obj.Render (vtkRenderer ren, vtkActor act)` - This calls RenderPiece (in a for loop is streaming is necessary). Basically a reimplementation for vtkPolyDataMapper::Render() since we don't want it to give up when vtkCompositeDataSet is encountered.
- `obj.SetColorBlocks (int )` - When set, each block is colored with a different color. Note that scalar coloring will be ignored.
- `int = obj.GetColorBlocks ()` - When set, each block is colored with a different color. Note that scalar coloring will be ignored.

39.23 vtkCuller

39.23.1 Usage

A culler has a cull method called by the vtkRenderer. The cull method is called before any rendering is performed, and it allows the culler to do some processing on the props and to modify their AllocatedRenderTime and re-order them in the prop list.

To create an instance of class vtkCuller, simply invoke its constructor as follows

```
obj = vtkCuller
```

39.23.2 Methods

The class vtkCuller has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCuller class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCuller = obj.NewInstance ()`
- `vtkCuller = obj.SafeDownCast (vtkObject o)`

39.24 vtkCullerCollection

39.24.1 Usage

vtkCullerCollection represents and provides methods to manipulate a list of Cullers (i.e., vtkCuller and subclasses). The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkCullerCollection, simply invoke its constructor as follows

```
obj = vtkCullerCollection
```

39.24.2 Methods

The class vtkCullerCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCullerCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkCullerCollection = obj.NewInstance ()`
- `vtkCullerCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkCuller a)` - Get the next Culler in the list.
- `vtkCuller = obj.GetNextItem ()` - Get the last Culler in the list.
- `vtkCuller = obj.GetLastItem ()` - Get the last Culler in the list.

39.25 vtkDataSetMapper

39.25.1 Usage

vtkDataSetMapper is a mapper to map data sets (i.e., vtkDataSet and all derived classes) to graphics primitives. The mapping procedure is as follows: all 0D, 1D, and 2D cells are converted into points, lines, and polygons/triangle strips and then mapped to the graphics system. The 2D faces of 3D cells are mapped only if they are used by only one cell, i.e., on the boundary of the data set.

To create an instance of class vtkDataSetMapper, simply invoke its constructor as follows

```
obj = vtkDataSetMapper
```

39.25.2 Methods

The class vtkDataSetMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataSetMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataSetMapper = obj.NewInstance ()`
- `vtkDataSetMapper = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer ren, vtkActor act)`
- `vtkPolyDataMapper = obj.GetPolyDataMapper ()` - Get the internal poly data mapper used to map data set to graphics system.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this mapper. The parameter window could be used to determine which graphic resources to release.
- `long = obj.GetMTime ()` - Get the mtime also considering the lookup table.
- `obj.SetInput (vtkDataSet input)` - Set the Input of this mapper.
- `vtkDataSet = obj.GetInput ()` - Set the Input of this mapper.

39.26 vtkDataTransferHelper

39.26.1 Usage

vtkDataTransferHelper is a helper class that aids in transferring data between the CPU memory and the GPU memory. The data in GPU memory is stored as textures which that in CPU memory is stored as vtkDataArray. vtkDataTransferHelper provides API to transfer only a sub-extent of CPU structured data to/from the GPU.

To create an instance of class vtkDataTransferHelper, simply invoke its constructor as follows

```
obj = vtkDataTransferHelper
```

39.26.2 Methods

The class vtkDataTransferHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataTransferHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataTransferHelper = obj.NewInstance ()`
- `vtkDataTransferHelper = obj.SafeDownCast (vtkObject o)`
- `obj.SetContext (vtkRenderWindow context)` - Get/Set the context. Context must be a vtkOpenGLRenderWindow. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions.

- `vtkRenderWindow = obj.GetContext ()` - Get/Set the context. Context must be a vtkOpenGLRenderWindow. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions.

- `obj.SetCPUExtent (int , int , int , int , int , int )` - Set the CPU data extent. The extent matches the vtkDataArray size. If the vtkDataArray comes from an vtkImageData and it is part of the point data, it is usually the vtkImageData extent. It can be on cell data too, but in this case it does not match the vtkImageData extent. If the vtkDataArray comes from a vtkDataSet, just set it to a one-dimenstional extent equal to the number of tuples. Initial value is (0,0,0,0,0,0), a valid one tuple array.

- `obj.SetCPUExtent (int a[6])` - Set the CPU data extent. The extent matches the vtkDataArray size. If the vtkDataArray comes from an vtkImageData and it is part of the point data, it is usually the vtkImageData extent. It can be on cell data too, but in this case it does not match the vtkImageData extent. If the vtkDataArray comes from a vtkDataSet, just set it to a one-dimenstional extent equal to the number of tuples. Initial value is (0,0,0,0,0,0), a valid one tuple array.

- `int = obj. GetCPUExtent ()` - Set the CPU data extent. The extent matches the vtkDataArray size. If the vtkDataArray comes from an vtkImageData and it is part of the point data, it is usually the vtkImageData extent. It can be on cell data too, but in this case it does not match the vtkImageData extent. If the vtkDataArray comes from a vtkDataSet, just set it to a one-dimenstional extent equal to the number of tuples. Initial value is (0,0,0,0,0,0), a valid one tuple array.

- `obj.SetGPUExtent (int , int , int , int , int , int )` - Set the GPU data extent. This is the sub-extent to copy from or to the GPU. This extent matches the size of the data to transfer. GPUExtent and TextureExtent don't have to match (GPUExtent can be 1D whereas TextureExtent is 2D) but the number of elements have to match. Initial value is (0,0,0,0,0,0), a valid one tuple array.

- `obj.SetGPUExtent (int a[6])` - Set the GPU data extent. This is the sub-extent to copy from or to the GPU. This extent matches the size of the data to transfer. GPUExtent and TextureExtent don't have to match (GPUExtent can be 1D whereas TextureExtent is 2D) but the number of elements have to match. Initial value is (0,0,0,0,0,0), a valid one tuple array.

- `int = obj. GetGPUExtent ()` - Set the GPU data extent. This is the sub-extent to copy from or to the GPU. This extent matches the size of the data to transfer. GPUExtent and TextureExtent don't have to match (GPUExtent can be 1D whereas TextureExtent is 2D) but the number of elements have to match. Initial value is (0,0,0,0,0,0), a valid one tuple array.

- `obj.SetTextureExtent (int , int , int , int , int , int )` - Set the texture data extent. This is the extent of the texture image that will receive the data. This extent matches the size of the data to transfer. If it is set to an invalid extent, GPUExtent is used. See more comment on GPUExtent. Initial value is an invalid extent.

- `obj.SetTextureExtent (int a[6])` - Set the texture data extent. This is the extent of the texture image that will receive the data. This extent matches the size of the data to transfer. If it is set to an invalid extent, GPUExtent is used. See more comment on GPUExtent. Initial value is an invalid extent.

- `int = obj. GetTextureExtent ()` - Set the texture data extent. This is the extent of the texture image that will receive the data. This extent matches the size of the data to transfer. If it is set to an invalid extent, GPUExtent is used. See more comment on GPUExtent. Initial value is an invalid extent.

- `bool = obj.GetExtentIsValid (int extent)` - Tells if the given extent (6 int) is valid. True if min extent¡=max extent.

- `bool = obj.GetCPUExtentIsValid ()` - Tells if CPUExtent is valid. True if min extent¡=max extent.

- `bool = obj.GetGPUExtentIsValid ()` - Tells if GPUExtent is valid. True if min extent¡=max extent.

- `bool = obj.GetTextureExtentIsValid ()` - Tells if TextureExtent is valid. True if min extent¡=max extent.

- `obj.SetMinTextureDimension (int )` - Define the minimal dimension of the texture regardless of the dimensions of the TextureExtent. Initial value is 1. A texture extent can have a given dimension 0D (one value), 1D, 2D or 3D. By default 0D and 1D are translated into a 1D texture, 2D is translated into a 2D texture, 3D is translated into a 3D texture. To make life easier when writting GLSL code and use only one type of sampler (ex: sampler2d), the default behavior can be changed by forcing a type of texture with this ivar. 1: default behavior. Initial value. 2: force 0D and 1D to be in a 2D texture 3: force 0D, 1D and 2D texture to be in a 3D texture.

- `int = obj.GetMinTextureDimension ()` - Define the minimal dimension of the texture regardless of the dimensions of the TextureExtent. Initial value is 1. A texture extent can have a given dimension 0D (one value), 1D, 2D or 3D. By default 0D and 1D are translated into a 1D texture, 2D is translated into a 2D texture, 3D is translated into a 3D texture. To make life easier when writting GLSL code and use only one type of sampler (ex: sampler2d), the default behavior can be changed by forcing a type of texture with this ivar. 1: default behavior. Initial value. 2: force 0D and 1D to be in a 2D texture 3: force 0D, 1D and 2D texture to be in a 3D texture.

- `vtkDataArray = obj.GetArray ()` - Get/Set the CPU data buffer. Initial value is 0.

- `obj.SetArray (vtkDataArray array)` - Get/Set the CPU data buffer. Initial value is 0.

- `vtkTextureObject = obj.GetTexture ()` - Get/Set the GPU data buffer. Initial value is 0.

- `obj.SetTexture (vtkTextureObject texture)` - Get/Set the GPU data buffer. Initial value is 0.

- `bool = obj.Upload (int components, int componentListNULL)` - Old comment. Upload Extent from CPU data buffer to GPU. The WholeExtent must match the Array size. New comment. Upload GPUExtent from CPU vtkDataArray to GPU texture. It is possible to send a subset of the components or to specify and order of components or both. If components=0, componentList is ignored and all components are passed, a texture cannot have more than 4 components.

- `bool = obj.Download ()` - old comment: Downlad Extent from GPU data buffer to CPU. GPU data size must exactly match Extent. CPU data buffer will be resized to match WholeExtent in which only the Extent will be filled with the GPU data. new comment: Download GPUExtent from GPU texture to CPU vtkDataArray. If Array is not provided, it will be created with the size of CPUExtent. But only the tuples covered by GPUExtent will be download. In this case, if GPUExtent does not cover all GPUExtent, some of the vtkDataArray will be uninitialized. Reminder: A=¿B ¡=¿ !A——B

- `bool = obj.DownloadAsync1 ()` - Splits the download in two operations * Asynchronously download from texture memory to PBO (DownloadAsync1()). * Copy from pbo to user array (DownloadAsync2()).

- `bool = obj.DownloadAsync2 ()` - Splits the download in two operations * Asynchronously download from texture memory to PBO (DownloadAsync1()). * Copy from pbo to user array (DownloadAsync2()).

- `bool = obj.GetShaderSupportsTextureInt ()`

- `obj.SetShaderSupportsTextureInt (bool value)`

39.27 vtkDefaultPainter

39.27.1 Usage

This painter does not do any actual rendering. Sets up a default pipeline of painters to mimick the behaiour of old vtkPolyDataMapper. The chain is as follows: input–¿ vtkScalarsToColorsPainter –¿ vtkClipPlanesPainter –¿ vtkDisplayListPainter –¿ vtkCompositePainter –¿ vtkCoincidentTopologyResolutionPainter –¿ vtkLightingPainter –¿ vtkRepresentationPainter –¿ Delegate of vtkDefaultPainter. Typically, the delegate of the default painter be one that is capable of r rendering graphics primitives or a vtkChooserPainter which can select appropriate painters to do the rendering.

To create an instance of class vtkDefaultPainter, simply invoke its constructor as follows

```
obj = vtkDefaultPainter
```

39.27.2 Methods

The class vtkDefaultPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDefaultPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDefaultPainter = obj.NewInstance ()`
- `vtkDefaultPainter = obj.SafeDownCast (vtkObject o)`
- `obj.SetScalarsToColorsPainter (vtkScalarsToColorsPainter )` - Get/Set the painter that maps scalars to colors.
- `vtkScalarsToColorsPainter = obj.GetScalarsToColorsPainter ()` - Get/Set the painter that maps scalars to colors.
- `obj.SetClipPlanesPainter (vtkClipPlanesPainter )` - Get/Set the painter that handles clipping.
- `vtkClipPlanesPainter = obj.GetClipPlanesPainter ()` - Get/Set the painter that handles clipping.
- `obj.SetDisplayListPainter (vtkDisplayListPainter )` - Get/Set the painter that builds display lists.
- `vtkDisplayListPainter = obj.GetDisplayListPainter ()` - Get/Set the painter that builds display lists.
- `obj.SetCompositePainter (vtkCompositePainter )` - Get/Set the painter used to handle composite datasets.
- `vtkCompositePainter = obj.GetCompositePainter ()` - Get/Set the painter used to handle composite datasets.
- `obj.SetCoincidentTopologyResolutionPainter (vtkCoincidentTopologyResolutionPainter )` - Painter used to resolve coincident topology.
- `vtkCoincidentTopologyResolutionPainter = obj.GetCoincidentTopologyResolutionPainter ()` - Painter used to resolve coincident topology.
- `obj.SetLightingPainter (vtkLightingPainter )` - Get/Set the painter that controls lighting.
- `vtkLightingPainter = obj.GetLightingPainter ()` - Get/Set the painter that controls lighting.

- `obj.SetRepresentationPainter (vtkRepresentationPainter )` - Painter used to convert polydata to Wireframe/Points representation.
- `vtkRepresentationPainter = obj.GetRepresentationPainter ()` - Painter used to convert polydata to Wireframe/Points representation.
- `obj.SetDelegatePainter (vtkPainter )` - Set/Get the painter to which this painter should propagare its draw calls. These methods are overridden so that the delegate is set to the end of the Painter Chain.
- `vtkPainter = obj.GetDelegatePainter ()` - Overridden to setup the chain of painter depending on the actor representation. The chain is rebuilt if this-¿MTime has changed since last BuildPainterChain(); Building of the chain does not depend on input polydata, hence it does not check if the input has changed at all.
- `obj.Render (vtkRenderer renderer, vtkActor actor, long typeflags, bool forceCompileOnly)` - Overridden to setup the chain of painter depending on the actor representation. The chain is rebuilt if this-¿MTime has changed since last BuildPainterChain(); Building of the chain does not depend on input polydata, hence it does not check if the input has changed at all.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this painter. The parameter window could be used to determine which graphic resources to release. The call is propagated to the delegate painter, if any.
- `obj.UpdateBounds (double bounds[6])` - Expand or shrink the estimated bounds based on the geometric transformations applied in the painter. The bounds are left unchanged if the painter does not change the geometry.

39.28 vtkDefaultPass

39.28.1 Usage

vtkDefaultPass implements the basic standard render passes of VTK. Subclasses can easily be implemented by reusing some parts of the basic implementation.

It implements classic Render operations as well as versions with property key checking.

This pass expects an initialized depth buffer and color buffer. Initialized buffers means they have been cleared with farest z-value and background color/gradient/transparent color.

To create an instance of class vtkDefaultPass, simply invoke its constructor as follows

```
obj = vtkDefaultPass
```

39.28.2 Methods

The class vtkDefaultPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDefaultPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDefaultPass = obj.NewInstance ()`
- `vtkDefaultPass = obj.SafeDownCast (vtkObject o)`

39.29 vtkDepthPeelingPass

39.29.1 Usage

Render the translucent polygonal geometry of a scene without sorting polygons in the view direction.

This pass expects an initialized depth buffer and color buffer. Initialized buffers means they have been cleared with farest z-value and background color/gradient/transparent color. An opaque pass may have been performed right after the initialization.

The depth peeling algorithm works by rendering the translucent polygonal geometry multiple times (once for each peel). The actually rendering of the translucent polygonal geometry is peformed by its delegate TranslucentPass. This delegate is therefore used multiple times.

Its delegate is usually set to a vtkTranslucentPass.

To create an instance of class vtkDepthPeelingPass, simply invoke its constructor as follows

```
obj = vtkDepthPeelingPass
```

39.29.2 Methods

The class vtkDepthPeelingPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDepthPeelingPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDepthPeelingPass = obj.NewInstance ()`
- `vtkDepthPeelingPass = obj.SafeDownCast (vtkObject o)`
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Release graphics resources and ask components to release their own resources.
- `vtkRenderPass = obj.GetTranslucentPass ()` - Delegate for rendering the translucent polygonal geometry. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkTranslucentPass. Initial value is a NULL pointer.
- `obj.SetTranslucentPass (vtkRenderPass translucentPass)` - Delegate for rendering the translucent polygonal geometry. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkTranslucentPass. Initial value is a NULL pointer.
- `obj.SetOcclusionRatio (double )` - In case of use of depth peeling technique for rendering translucent material, define the threshold under which the algorithm stops to iterate over peel layers. This is the ratio of the number of pixels that have been touched by the last layer over the total number of pixels of the viewport area. Initial value is 0.0, meaning rendering have to be exact. Greater values may speed-up the rendering with small impact on the quality.
- `double = obj.GetOcclusionRatioMinValue ()` - In case of use of depth peeling technique for rendering translucent material, define the threshold under which the algorithm stops to iterate over peel layers. This is the ratio of the number of pixels that have been touched by the last layer over the total number of pixels of the viewport area. Initial value is 0.0, meaning rendering have to be exact. Greater values may speed-up the rendering with small impact on the quality.
- `double = obj.GetOcclusionRatioMaxValue ()` - In case of use of depth peeling technique for rendering translucent material, define the threshold under which the algorithm stops to iterate over peel layers. This is the ratio of the number of pixels that have been touched by the last layer over the total number of pixels of the viewport area. Initial value is 0.0, meaning rendering have to be exact. Greater values may speed-up the rendering with small impact on the quality.

- `double = obj.GetOcclusionRatio ()` - In case of use of depth peeling technique for rendering translucent material, define the threshold under which the algorithm stops to iterate over peel layers. This is the ratio of the number of pixels that have been touched by the last layer over the total number of pixels of the viewport area. Initial value is 0.0, meaning rendering have to be exact. Greater values may speed-up the rendering with small impact on the quality.
- `obj.SetMaximumNumberOfPeels (int )` - In case of depth peeling, define the maximum number of peeling layers. Initial value is 4. A special value of 0 means no maximum limit. It has to be a positive value.
- `int = obj.GetMaximumNumberOfPeels ()` - In case of depth peeling, define the maximum number of peeling layers. Initial value is 4. A special value of 0 means no maximum limit. It has to be a positive value.
- `bool = obj.GetLastRenderingUsedDepthPeeling ()` - Tells if the last time this pass was executed, the depth peeling algorithm was actually used. Initial value is false.

39.30 vtkDistanceToCamera

39.30.1 Usage

This filter adds a double array containing the distance from each point to the camera. If Scaling is on, it will use the values in the input array to process in order to scale the size of the points. ScreenSize sets the size in screen pixels that you would want a rendered rectangle at that point to be, if it was scaled by the output array.

To create an instance of class vtkDistanceToCamera, simply invoke its constructor as follows

```
obj = vtkDistanceToCamera
```

39.30.2 Methods

The class vtkDistanceToCamera has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDistanceToCamera class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDistanceToCamera = obj.NewInstance ()`
- `vtkDistanceToCamera = obj.SafeDownCast (vtkObject o)`
- `obj.SetRenderer (vtkRenderer ren)` - The renderer which will ultimately render these points.
- `vtkRenderer = obj.GetRenderer ()` - The renderer which will ultimately render these points.
- `obj.SetScreenSize (double )` - The desired screen size obtained by scaling glyphs by the distance array. It assumes the glyph at each point will be unit size.
- `double = obj.GetScreenSize ()` - The desired screen size obtained by scaling glyphs by the distance array. It assumes the glyph at each point will be unit size.
- `obj.SetScaling (bool )` - Whether to scale the distance by the input array to process.
- `bool = obj.GetScaling ()` - Whether to scale the distance by the input array to process.
- `obj.ScalingOn ()` - Whether to scale the distance by the input array to process.
- `obj.ScalingOff ()` - Whether to scale the distance by the input array to process.
- `long = obj.GetMTime ()` - The modified time of this filter.

39.31 vtkDummyGPUInfoList

39.31.1 Usage

vtkDummyGPUInfoList implements Probe() by just setting the count of GPUs to be zero. Useful when an OS specific implementation is not available.

To create an instance of class vtkDummyGPUInfoList, simply invoke its constructor as follows

```
obj = vtkDummyGPUInfoList
```

39.31.2 Methods

The class vtkDummyGPUInfoList has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDummyGPUInfoList class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDummyGPUInfoList = obj.NewInstance ()`
- `vtkDummyGPUInfoList = obj.SafeDownCast (vtkObject o)`
- `obj.Probe ()` - Build the list of vtkInfoGPU if not done yet.

39.32 vtkDynamic2DLabelMapper

39.32.1 Usage

vtkDynamic2DLabelMapper is a mapper that renders text at dataset points such that the labels do not overlap. Various items can be labeled including point ids, scalars, vectors, normals, texture coordinates, tensors, and field data components. This mapper assumes that the points are located on the x-y plane and that the camera remains perpendicular to that plane with a y-up axis (this can be constrained using vtkImageInteractor). On the first render, the mapper computes the visiblility of all labels at all scales, and queries this information on successive renders. This causes the first render to be much slower. The visibility algorithm is a greedy approach based on the point id, so the label for a point will be drawn unless the label for a point with lower id overlaps it.

To create an instance of class vtkDynamic2DLabelMapper, simply invoke its constructor as follows

```
obj = vtkDynamic2DLabelMapper
```

39.32.2 Methods

The class vtkDynamic2DLabelMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDynamic2DLabelMapper class.

- `string = obj.GetClassName ()` - Instantiate object with are labeled.
- `int = obj.IsA (string name)` - Instantiate object with are labeled.
- `vtkDynamic2DLabelMapper = obj.NewInstance ()` - Instantiate object with are labeled.
- `vtkDynamic2DLabelMapper = obj.SafeDownCast (vtkObject o)` - Instantiate object with are labeled.

- `obj.SetPriorityArrayName (string name)` - Set the points array name to use to give priority to labels. Defaults to "priority".
- `obj.SetReversePriority (bool )` - Whether to reverse the priority order (i.e. low values have high priority). Default is off.
- `bool = obj.GetReversePriority ()` - Whether to reverse the priority order (i.e. low values have high priority). Default is off.
- `obj.ReversePriorityOn ()` - Whether to reverse the priority order (i.e. low values have high priority). Default is off.
- `obj.ReversePriorityOff ()` - Whether to reverse the priority order (i.e. low values have high priority). Default is off.
- `obj.SetLabelHeightPadding (float )` - Set the label height padding as a percentage. The percentage is a percentage of your label height. Default is 50
- `float = obj.GetLabelHeightPadding ()` - Set the label height padding as a percentage. The percentage is a percentage of your label height. Default is 50
- `obj.SetLabelWidthPadding (float )` - Set the label width padding as a percentage. The percentage is a percentage of your label $\hat{h}$eight$\hat{(}$yes, not a typo). Default is 50
- `float = obj.GetLabelWidthPadding ()` - Set the label width padding as a percentage. The percentage is a percentage of your label $\hat{h}$eight$\hat{(}$yes, not a typo). Default is 50
- `obj.RenderOpaqueGeometry (vtkViewport viewport, vtkActor2D actor)` - Draw non-overlapping labels to the screen.
- `obj.RenderOverlay (vtkViewport viewport, vtkActor2D actor)` - Draw non-overlapping labels to the screen.

39.33 vtkExporter

39.33.1 Usage

vtkExporter is an abstract class that exports a scene to a file. It is very similar to vtkWriter except that a writer only writes out the geometric and topological data for an object, where an exporter can write out material properties, lighting, camera parameters etc. The concrete subclasses of this class may not write out all of this information. For example vtkOBJExporter writes out Wavefront obj files which do not include support for camera parameters.

vtkExporter provides the convenience methods StartWrite() and EndWrite(). These methods are executed before and after execution of the Write() method. You can also specify arguments to these methods. This class defines SetInput and GetInput methods which take or return a vtkRenderWindow.

To create an instance of class vtkExporter, simply invoke its constructor as follows

```
obj = vtkExporter
```

39.33.2 Methods

The class vtkExporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkExporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkExporter = obj.NewInstance ()`
- `vtkExporter = obj.SafeDownCast (vtkObject o)`
- `obj.Write ()` - Write data to output. Method executes subclasses WriteData() method, as well as StartWrite() and EndWrite() methods.
- `obj.Update ()` - Convenient alias for Write() method.
- `obj.SetRenderWindow (vtkRenderWindow )` - Set/Get the rendering window that contains the scene to be written.
- `vtkRenderWindow = obj.GetRenderWindow ()` - Set/Get the rendering window that contains the scene to be written.
- `obj.SetInput (vtkRenderWindow renWin)` - These methods are provided for backward compatibility. Will disappear soon.
- `vtkRenderWindow = obj.GetInput ()` - These methods are provided for backward compatibility. Will disappear soon.
- `long = obj.GetMTime ()` - Returns the MTime also considering the RenderWindow.

39.34 vtkFollower

39.34.1 Usage

vtkFollower is a subclass of vtkActor that always follows its specified camera. More specifically it will not change its position or scale, but it will continually update its orientation so that it is right side up and facing the camera. This is typically used for text labels in a scene. All of the adjustments that can be made to an actor also will take effect with a follower. So, if you change the orientation of the follower by 90 degrees, then it will follow the camera, but be off by 90 degrees.

To create an instance of class vtkFollower, simply invoke its constructor as follows

```
obj = vtkFollower
```

39.34.2 Methods

The class vtkFollower has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFollower class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFollower = obj.NewInstance ()`
- `vtkFollower = obj.SafeDownCast (vtkObject o)`
- `obj.SetCamera (vtkCamera )` - Set/Get the camera to follow. If this is not set, then the follower won't know who to follow.
- `vtkCamera = obj.GetCamera ()` - Set/Get the camera to follow. If this is not set, then the follower won't know who to follow.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - This causes the actor to be rendered. It in turn will render the actor's property, texture map and then mapper. If a property hasn't been assigned, then the actor will create one automatically.

- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - This causes the actor to be rendered. It in turn will render the actor's property, texture map and then mapper. If a property hasn't been assigned, then the actor will create one automatically.
- `obj.Render (vtkRenderer ren)` - This causes the actor to be rendered. It in turn will render the actor's property, texture map and then mapper. If a property hasn't been assigned, then the actor will create one automatically.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.GetMatrix (vtkMatrix4x4 m)` - Copy the follower's composite 4x4 matrix into the matrix provided.
- `obj.GetMatrix (double m[16])` - Copy the follower's composite 4x4 matrix into the matrix provided.
- `vtkMatrix4x4 = obj.GetMatrix ()` - Shallow copy of a follower. Overloads the virtual vtkProp method.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of a follower. Overloads the virtual vtkProp method.

39.35 vtkFrameBufferObject

39.35.1 Usage

Encapsulates an OpenGL Frame Buffer Object. For use by vtkOpenGLFBORenderWindow, not to be used directly.

To create an instance of class vtkFrameBufferObject, simply invoke its constructor as follows

```
obj = vtkFrameBufferObject
```

39.35.2 Methods

The class vtkFrameBufferObject has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFrameBufferObject class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFrameBufferObject = obj.NewInstance ()`
- `vtkFrameBufferObject = obj.SafeDownCast (vtkObject o)`
- `obj.SetContext (vtkRenderWindow context)` - Get/Set the context. Context must be a vtkOpenGLRenderWindow. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions.
- `vtkRenderWindow = obj.GetContext ()` - Get/Set the context. Context must be a vtkOpenGLRenderWindow. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions.
- `bool = obj.Start (int width, int height, bool shaderSupportsTextureInt)` - User must take care that width/height match the dimensions of the user defined texture attachments. This method makes the "active buffers" the buffers that will get drawn into by subsequent drawing calls. Note that this does not clear the render buffers i.e. no glClear() calls are made by either of these methods. It's up to the caller to clear the buffers if needed.

- `bool = obj.StartNonOrtho (int width, int height, bool shaderSupportsTextureInt)` - User must take care that width/height match the dimensions of the user defined texture attachments. This method makes the "active buffers" the buffers that will get drawn into by subsequent drawing calls. Note that this does not clear the render buffers i.e. no glClear() calls are made by either of these methods. It's up to the caller to clear the buffers if needed.

- `obj.RenderQuad (int minX, int maxX, int minY, int maxY)` - Renders a quad at the given location with pixel coordinates. This method is provided as a convenience, since we often render quads in a FBO.

- `obj.Bind ()` - Save the current framebuffer and make the frame buffer active. Multiple calls to Bind has no effect.

- `obj.UnBind ()` - Restore the framebuffer saved with the call to Bind(). Multiple calls to UnBind has no effect.

- `obj.SetActiveBuffer (int index)` - Choose the buffer to render into. This is available only if the GL_ARB_draw_buffers extension is supported by the card.

- `obj.SetActiveBuffers (int numbuffers, int indices[])` - Choose the buffer to render into. This is available only if the GL_ARB_draw_buffers extension is supported by the card.

- `obj.SetColorBuffer (int index, vtkTextureObject texture, int zslice)`

- `vtkTextureObject = obj.GetColorBuffer (int index)`

- `obj.RemoveColorBuffer (int index)`

- `obj.RemoveAllColorBuffers ()`

- `obj.SetDepthBuffer (vtkTextureObject depthTexture)` - Set the texture to use as depth buffer.

- `obj.RemoveDepthBuffer ()` - Set the texture to use as depth buffer.

- `obj.SetDepthBufferNeeded (bool )` - If true, the frame buffer object will be initialized with a depth buffer. Initial value is true.

- `bool = obj.GetDepthBufferNeeded ()` - If true, the frame buffer object will be initialized with a depth buffer. Initial value is true.

- `obj.SetNumberOfRenderTargets (int )` - Set/Get the number of render targets to render into at once.

- `int = obj.GetNumberOfRenderTargets ()` - Set/Get the number of render targets to render into at once.

- `int = obj.GetMaximumNumberOfActiveTargets ()` - Returns the maximum number of targets that can be rendered to at one time. This limits the active targets set by SetActiveTargets(). The return value is valid only if GetContext is non-null.

- `int = obj.GetMaximumNumberOfRenderTargets ()` - Returns the maximum number of render targets available. This limits the available attachement points for SetColorAttachment(). The return value is valid only if GetContext is non-null.

- `int = obj. GetLastSize ()` - Dimensions in pixels of the framebuffer.

39.36 vtkFreeTypeLabelRenderStrategy

39.36.1 Usage

Uses the FreeType to render labels and compute label sizes. This strategy may be used with vtkLabelPlacementMapper.

To create an instance of class vtkFreeTypeLabelRenderStrategy, simply invoke its constructor as follows

```
obj = vtkFreeTypeLabelRenderStrategy
```

39.36.2 Methods

The class vtkFreeTypeLabelRenderStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFreeTypeLabelRenderStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFreeTypeLabelRenderStrategy = obj.NewInstance ()`
- `vtkFreeTypeLabelRenderStrategy = obj.SafeDownCast (vtkObject o)`
- `bool = obj.SupportsRotation ()` - The free type render strategy currently does not support bounded size labels.
- `bool = obj.SupportsBoundedSize ()` - Release any graphics resources that are being consumed by this strategy. The parameter window could be used to determine which graphic resources to release.
- `obj.ReleaseGraphicsResources (vtkWindow window)` - Release any graphics resources that are being consumed by this strategy. The parameter window could be used to determine which graphic resources to release.

39.37 vtkFrustumCoverageCuller

39.37.1 Usage

vtkFrustumCoverageCuller will cull props based on the coverage in the view frustum. The coverage is computed by enclosing the prop in a bounding sphere, projecting that to the viewing coordinate system, then taking a slice through the view frustum at the center of the sphere. This results in a circle on the plane slice through the view frustum. This circle is enclosed in a squared, and the fraction of the plane slice that this square covers is the coverage. This is a number between 0 and 1. If the number is less than the MinumumCoverage, the allocated render time for that prop is set to zero. If it is greater than the MaximumCoverage, the allocated render time is set to 1.0. In between, a linear ramp is used to convert coverage into allocated render time.

To create an instance of class vtkFrustumCoverageCuller, simply invoke its constructor as follows

```
obj = vtkFrustumCoverageCuller
```

39.37.2 Methods

The class vtkFrustumCoverageCuller has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFrustumCoverageCuller class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFrustumCoverageCuller = obj.NewInstance ()`
- `vtkFrustumCoverageCuller = obj.SafeDownCast (vtkObject o)`
- `obj.SetMinimumCoverage (double )` - Set/Get the minimum coverage - props with less coverage than this are given no time to render (they are culled)
- `double = obj.GetMinimumCoverage ()` - Set/Get the minimum coverage - props with less coverage than this are given no time to render (they are culled)
- `obj.SetMaximumCoverage (double )` - Set/Get the maximum coverage - props with more coverage than this are given an allocated render time of 1.0 (the maximum)
- `double = obj.GetMaximumCoverage ()` - Set/Get the maximum coverage - props with more coverage than this are given an allocated render time of 1.0 (the maximum)
- `obj.SetSortingStyle (int )` - Set the sorting style - none, front-to-back or back-to-front The default is none
- `int = obj.GetSortingStyleMinValue ()` - Set the sorting style - none, front-to-back or back-to-front The default is none
- `int = obj.GetSortingStyleMaxValue ()` - Set the sorting style - none, front-to-back or back-to-front The default is none
- `int = obj.GetSortingStyle ()` - Set the sorting style - none, front-to-back or back-to-front The default is none
- `obj.SetSortingStyleToNone ()` - Set the sorting style - none, front-to-back or back-to-front The default is none
- `obj.SetSortingStyleToBackToFront ()` - Set the sorting style - none, front-to-back or back-to-front The default is none
- `obj.SetSortingStyleToFrontToBack ()` - Set the sorting style - none, front-to-back or back-to-front The default is none
- `string = obj.GetSortingStyleAsString (void )` - Set the sorting style - none, front-to-back or back-to-front The default is none

39.38 vtkGaussianBlurPass

39.38.1 Usage

Blur the image renderered by its delegate. Blurring uses a Gaussian low-pass filter with a 5x5 kernel.

This pass expects an initialized depth buffer and color buffer. Initialized buffers means they have been cleared with farest z-value and background color/gradient/transparent color. An opaque pass may have been performed right after the initialization.

The delegate is used once.

Its delegate is usually set to a vtkCameraPass or to a post-processing pass.

This pass requires a OpenGL context that supports texture objects (TO), framebuffer objects (FBO) and GLSL. If not, it will emit an error message and will render its delegate and return.

.SECTION Implementation As the filter is separable, it first blurs the image horizontally and then vertically. This reduces the number of texture sampling to 5 per pass. In addition, as texture sampling can already blend texel values in linear mode, by adjusting the texture coordinate accordingly, only 3 texture

sampling are actually necessary. Reference: OpenGL Bloom Toturial by Philip Rideout, section Exploit Hardware Filtering http://prideout.net/bloom/index.php#Sneaky

To create an instance of class vtkGaussianBlurPass, simply invoke its constructor as follows

```
obj = vtkGaussianBlurPass
```

39.38.2 Methods

The class vtkGaussianBlurPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGaussianBlurPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGaussianBlurPass = obj.NewInstance ()`
- `vtkGaussianBlurPass = obj.SafeDownCast (vtkObject o)`
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Release graphics resources and ask components to release their own resources.

39.39 vtkGenericRenderWindowInteractor

39.39.1 Usage

vtkGenericRenderWindowInteractor provides a way to translate native mouse and keyboard events into vtk Events. By calling the methods on this class, vtk events will be invoked. This will allow scripting languages to use vtkInteractorStyles and 3D widgets.

To create an instance of class vtkGenericRenderWindowInteractor, simply invoke its constructor as follows

```
obj = vtkGenericRenderWindowInteractor
```

39.39.2 Methods

The class vtkGenericRenderWindowInteractor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericRenderWindowInteractor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenericRenderWindowInteractor = obj.NewInstance ()`
- `vtkGenericRenderWindowInteractor = obj.SafeDownCast (vtkObject o)`
- `obj.MouseMoveEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.RightButtonPressEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.RightButtonReleaseEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.

- `obj.LeftButtonPressEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.LeftButtonReleaseEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.MiddleButtonPressEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.MiddleButtonReleaseEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.MouseWheelForwardEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.MouseWheelBackwardEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.ExposeEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.ConfigureEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.EnterEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.LeaveEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.TimerEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.KeyPressEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.KeyReleaseEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.CharEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.ExitEvent ()` - Fire various events. SetEventInformation should be called just prior to calling any of these methods. These methods will Invoke the corresponding vtk event.
- `obj.SetTimerEventResetsTimer (int )` - Flag that indicates whether the TimerEvent method should call ResetTimer to simulate repeating timers with an endless stream of one shot timers. By default this flag is on and all repeating timers are implemented as a stream of sequential one shot timers. If the observer of CreateTimerEvent actually creates a "natively repeating" timer, setting this flag to off will prevent (perhaps many many) unnecessary calls to ResetTimer. Having the flag on by default means that "natively one shot" timers can be either one shot or repeating timers with no additional work. Also, "natively repeating" timers still work with the default setting, but with potentially many create and destroy calls.
- `int = obj.GetTimerEventResetsTimer ()` - Flag that indicates whether the TimerEvent method should call ResetTimer to simulate repeating timers with an endless stream of one shot timers. By default this flag is on and all repeating timers are implemented as a stream of sequential one shot timers. If the observer of CreateTimerEvent actually creates a "natively repeating" timer, setting this flag to off will prevent (perhaps many many) unnecessary calls to ResetTimer. Having the flag on by default means that "natively one shot" timers can be either one shot or repeating timers with

no additional work. Also, "natively repeating" timers still work with the default setting, but with potentially many create and destroy calls.

- `obj.TimerEventResetsTimerOn ()` - Flag that indicates whether the TimerEvent method should call ResetTimer to simulate repeating timers with an endless stream of one shot timers. By default this flag is on and all repeating timers are implemented as a stream of sequential one shot timers. If the observer of CreateTimerEvent actually creates a "natively repeating" timer, setting this flag to off will prevent (perhaps many many) unnecessary calls to ResetTimer. Having the flag on by default means that "natively one shot" timers can be either one shot or repeating timers with no additional work. Also, "natively repeating" timers still work with the default setting, but with potentially many create and destroy calls.

- `obj.TimerEventResetsTimerOff ()` - Flag that indicates whether the TimerEvent method should call ResetTimer to simulate repeating timers with an endless stream of one shot timers. By default this flag is on and all repeating timers are implemented as a stream of sequential one shot timers. If the observer of CreateTimerEvent actually creates a "natively repeating" timer, setting this flag to off will prevent (perhaps many many) unnecessary calls to ResetTimer. Having the flag on by default means that "natively one shot" timers can be either one shot or repeating timers with no additional work. Also, "natively repeating" timers still work with the default setting, but with potentially many create and destroy calls.

39.40 vtkGenericVertexAttributeMapping

39.40.1 Usage

vtkGenericVertexAttributeMapping stores mapping between data arrays and generic vertex attributes. It is used by vtkPainterPolyDataMapper to pass the mappings to the painter which rendering the attributes. .SECTION Thanks Support for generic vertex attributes in VTK was contributed in collaboration with Stephane Ploix at EDF.

To create an instance of class vtkGenericVertexAttributeMapping, simply invoke its constructor as follows

```
obj = vtkGenericVertexAttributeMapping
```

39.40.2 Methods

The class vtkGenericVertexAttributeMapping has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGenericVertexAttributeMapping class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGenericVertexAttributeMapping = obj.NewInstance ()`
- `vtkGenericVertexAttributeMapping = obj.SafeDownCast (vtkObject o)`
- `obj.AddMapping (string attributeName, string arrayName, int fieldAssociation, int component)` - Select a data array from the point/cell data and map it to a generic vertex attribute. Note that indices change when a mapping is added/removed.
- `obj.AddMapping (int unit, string arrayName, int fieldAssociation, int component)` - Select a data array and use it as multitexture texture coordinates. Note the texture unit parameter should correspond to the texture unit set on the texture.
- `bool = obj.RemoveMapping (string attributeName)` - Remove a vertex attribute mapping.

- `obj.RemoveAllMappings ()` - Remove all mappings.
- `int = obj.GetNumberOfMappings ()` - Get number of mapppings.
- `string = obj.GetAttributeName (int index)` - Get the attribute name at the given index.
- `string = obj.GetArrayName (int index)` - Get the array name at the given index.
- `int = obj.GetFieldAssociation (int index)` - Get the field association at the given index.
- `int = obj.GetComponent (int index)` - Get the component no. at the given index.
- `int = obj.GetTextureUnit (int index)` - Get the component no. at the given index.

39.41 vtkGL2PSExporter

39.41.1 Usage

vtkGL2PSExporter is a concrete subclass of vtkExporter that writes high quality vector PostScript (PS/EPS), PDF or SVG files by using GL2PS. GL2PS can be obtained at: http://www.geuz.org/gl2ps/. This can be very useful when one requires publication quality pictures. This class works best with simple 3D scenes and most 2D plots. Please note that GL2PS has its limitations since PostScript is not an ideal language to represent complex 3D scenes. However, this class does allow one to write mixed vector/raster files by using the Write3DPropsAsRasterImage ivar. Please do read the caveats section of this documentation.

By default vtkGL2PSExporter generates Encapsulated PostScript (EPS) output along with the text in portrait orientation with the background color of the window being drawn. The generated output is also compressed using zlib. The various other options are set to sensible defaults.

The output file format (FileFormat) can be either PostScript (PS), Encapsulated PostScript (EPS), PDF, SVG or TeX. The file extension is generated automatically depending on the FileFormat. The default is EPS. When TeX output is chosen, only the text strings in the plot are generated and put into a picture environment. One can turn on and off the text when generating PS/EPS/PDF/SVG files by using the Text boolean variable. By default the text is drawn. The background color of the renderwindow is drawn by default. To make the background white instead use the DrawBackgroundOff function. Landscape figures can be generated by using the LandscapeOn function. Portrait orientation is used by default. Several of the GL2PS options can be set. The names of the ivars for these options are similar to the ones that GL2PS provides. Compress, SimpleLineOffset, Silent, BestRoot, PS3Shading and OcclusionCull are similar to the options provided by GL2PS. Please read the function documentation or the GL2PS documentation for more details. The ivar Write3DPropsAsRasterImage allows one to generate mixed vector/raster images. All the 3D props in the scene will be written as a raster image and all 2D actors will be written as vector graphic primitives. This makes it possible to handle transparency and complex 3D scenes. This ivar is set to Off by default. When drawing lines and points the OpenGL point size and line width are multiplied by a factor in order to generate PostScript lines and points of the right size. The Get/SetGlobalPointSizeFactor and Get/SetGlobalLineWidthFactor let one customize this ratio. The default value is such that the PostScript output looks close to what is seen on screen.

To use this class you need to turn on VTK_USE_GL2PS when configuring VTK.

To create an instance of class vtkGL2PSExporter, simply invoke its constructor as follows

```
obj = vtkGL2PSExporter
```

39.41.2 Methods

The class vtkGL2PSExporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGL2PSExporter class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkGL2PSExporter = obj.NewInstance ()`
- `vtkGL2PSExporter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFilePrefix (string )` - Specify the prefix of the files to write out. The resulting filenames will have .ps or .eps or .tex appended to them depending on the other options chosen.
- `string = obj.GetFilePrefix ()` - Specify the prefix of the files to write out. The resulting filenames will have .ps or .eps or .tex appended to them depending on the other options chosen.
- `obj.SetFileFormat (int )` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `int = obj.GetFileFormatMinValue ()` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `int = obj.GetFileFormatMaxValue ()` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `int = obj.GetFileFormat ()` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `obj.SetFileFormatToPS ()` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `obj.SetFileFormatToEPS ()` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `obj.SetFileFormatToPDF ()` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `obj.SetFileFormatToTeX ()` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `obj.SetFileFormatToSVG ()` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `string = obj.GetFileFormatAsString ()` - Specify the format of file to write out. This can be one of: PS_FILE, EPS_FILE, PDF_FILE, TEX_FILE. Defaults to EPS_FILE. Depending on the option chosen it generates the appropriate file (with correct extension) when the Write function is called.
- `obj.SetSort (int )` - Set the the type of sorting algorithm to order primitives from back to front. Successive algorithms are more memory intensive. Simple is the default but BSP is perhaps the best.
- `int = obj.GetSortMinValue ()` - Set the the type of sorting algorithm to order primitives from back to front. Successive algorithms are more memory intensive. Simple is the default but BSP is perhaps the best.

- `int = obj.GetSortMaxValue ()` - Set the the type of sorting algorithm to order primitives from back to front. Successive algorithms are more memory intensive. Simple is the default but BSP is perhaps the best.
- `int = obj.GetSort ()` - Set the the type of sorting algorithm to order primitives from back to front. Successive algorithms are more memory intensive. Simple is the default but BSP is perhaps the best.
- `obj.SetSortToOff ()` - Set the the type of sorting algorithm to order primitives from back to front. Successive algorithms are more memory intensive. Simple is the default but BSP is perhaps the best.
- `obj.SetSortToSimple ()` - Set the the type of sorting algorithm to order primitives from back to front. Successive algorithms are more memory intensive. Simple is the default but BSP is perhaps the best.
- `obj.SetSortToBSP ()` - Set the the type of sorting algorithm to order primitives from back to front. Successive algorithms are more memory intensive. Simple is the default but BSP is perhaps the best.
- `string = obj.GetSortAsString ()` - Set the the type of sorting algorithm to order primitives from back to front. Successive algorithms are more memory intensive. Simple is the default but BSP is perhaps the best.
- `obj.SetCompress (int )` - Turn on/off compression when generating PostScript or PDF output. By default compression is on.
- `int = obj.GetCompress ()` - Turn on/off compression when generating PostScript or PDF output. By default compression is on.
- `obj.CompressOn ()` - Turn on/off compression when generating PostScript or PDF output. By default compression is on.
- `obj.CompressOff ()` - Turn on/off compression when generating PostScript or PDF output. By default compression is on.
- `obj.SetDrawBackground (int )` - Turn on/off drawing the background frame. If off the background is treated as white. By default the background is drawn.
- `int = obj.GetDrawBackground ()` - Turn on/off drawing the background frame. If off the background is treated as white. By default the background is drawn.
- `obj.DrawBackgroundOn ()` - Turn on/off drawing the background frame. If off the background is treated as white. By default the background is drawn.
- `obj.DrawBackgroundOff ()` - Turn on/off drawing the background frame. If off the background is treated as white. By default the background is drawn.
- `obj.SetSimpleLineOffset (int )` - Turn on/off the GL2PS_SIMPLE_LINE_OFFSET option. When enabled a small offset is added in the z-buffer to all the lines in the plot. This results in an anti-aliasing like solution. Defaults to on.
- `int = obj.GetSimpleLineOffset ()` - Turn on/off the GL2PS_SIMPLE_LINE_OFFSET option. When enabled a small offset is added in the z-buffer to all the lines in the plot. This results in an anti-aliasing like solution. Defaults to on.
- `obj.SimpleLineOffsetOn ()` - Turn on/off the GL2PS_SIMPLE_LINE_OFFSET option. When enabled a small offset is added in the z-buffer to all the lines in the plot. This results in an anti-aliasing like solution. Defaults to on.
- `obj.SimpleLineOffsetOff ()` - Turn on/off the GL2PS_SIMPLE_LINE_OFFSET option. When enabled a small offset is added in the z-buffer to all the lines in the plot. This results in an anti-aliasing like solution. Defaults to on.

- `obj.SetSilent (int )` - Turn on/off GL2PS messages sent to stderr (GL2PS_SILENT). When enabled GL2PS messages are suppressed. Defaults to off.
- `int = obj.GetSilent ()` - Turn on/off GL2PS messages sent to stderr (GL2PS_SILENT). When enabled GL2PS messages are suppressed. Defaults to off.
- `obj.SilentOn ()` - Turn on/off GL2PS messages sent to stderr (GL2PS_SILENT). When enabled GL2PS messages are suppressed. Defaults to off.
- `obj.SilentOff ()` - Turn on/off GL2PS messages sent to stderr (GL2PS_SILENT). When enabled GL2PS messages are suppressed. Defaults to off.
- `obj.SetBestRoot (int )` - Turn on/off the GL2PS_BEST_ROOT option. When enabled the construction of the BSP tree is optimized by choosing the root primitives leading to the minimum number of splits. Defaults to on.
- `int = obj.GetBestRoot ()` - Turn on/off the GL2PS_BEST_ROOT option. When enabled the construction of the BSP tree is optimized by choosing the root primitives leading to the minimum number of splits. Defaults to on.
- `obj.BestRootOn ()` - Turn on/off the GL2PS_BEST_ROOT option. When enabled the construction of the BSP tree is optimized by choosing the root primitives leading to the minimum number of splits. Defaults to on.
- `obj.BestRootOff ()` - Turn on/off the GL2PS_BEST_ROOT option. When enabled the construction of the BSP tree is optimized by choosing the root primitives leading to the minimum number of splits. Defaults to on.
- `obj.SetText (int )` - Turn on/off drawing the text. If on (default) the text is drawn. If the FileFormat is set to TeX output then a LaTeX picture is generated with the text strings. If off text output is suppressed.
- `int = obj.GetText ()` - Turn on/off drawing the text. If on (default) the text is drawn. If the FileFormat is set to TeX output then a LaTeX picture is generated with the text strings. If off text output is suppressed.
- `obj.TextOn ()` - Turn on/off drawing the text. If on (default) the text is drawn. If the FileFormat is set to TeX output then a LaTeX picture is generated with the text strings. If off text output is suppressed.
- `obj.TextOff ()` - Turn on/off drawing the text. If on (default) the text is drawn. If the FileFormat is set to TeX output then a LaTeX picture is generated with the text strings. If off text output is suppressed.
- `obj.SetLandscape (int )` - Turn on/off landscape orientation. If off (default) the orientation is set to portrait.
- `int = obj.GetLandscape ()` - Turn on/off landscape orientation. If off (default) the orientation is set to portrait.
- `obj.LandscapeOn ()` - Turn on/off landscape orientation. If off (default) the orientation is set to portrait.
- `obj.LandscapeOff ()` - Turn on/off landscape orientation. If off (default) the orientation is set to portrait.
- `obj.SetPS3Shading (int )` - Turn on/off the GL2PS_PS3_SHADING option. When enabled the shfill PostScript level 3 operator is used. Read the GL2PS documentation for more details. Defaults to on.

- `int = obj.GetPS3Shading ()` - Turn on/off the GL2PS_PS3_SHADING option. When enabled the shfill PostScript level 3 operator is used. Read the GL2PS documentation for more details. Defaults to on.
- `obj.PS3ShadingOn ()` - Turn on/off the GL2PS_PS3_SHADING option. When enabled the shfill PostScript level 3 operator is used. Read the GL2PS documentation for more details. Defaults to on.
- `obj.PS3ShadingOff ()` - Turn on/off the GL2PS_PS3_SHADING option. When enabled the shfill PostScript level 3 operator is used. Read the GL2PS documentation for more details. Defaults to on.
- `obj.SetOcclusionCull (int )` - Turn on/off culling of occluded polygons (GL2PS_OCCLUSION_CULL). When enabled hidden polygons are removed. This reduces file size considerably. Defaults to on.
- `int = obj.GetOcclusionCull ()` - Turn on/off culling of occluded polygons (GL2PS_OCCLUSION_CULL). When enabled hidden polygons are removed. This reduces file size considerably. Defaults to on.
- `obj.OcclusionCullOn ()` - Turn on/off culling of occluded polygons (GL2PS_OCCLUSION_CULL). When enabled hidden polygons are removed. This reduces file size considerably. Defaults to on.
- `obj.OcclusionCullOff ()` - Turn on/off culling of occluded polygons (GL2PS_OCCLUSION_CULL). When enabled hidden polygons are removed. This reduces file size considerably. Defaults to on.
- `obj.SetWrite3DPropsAsRasterImage (int )` - Turn on/off writing 3D props as raster images. 2D props are rendered using vector graphics primitives. If you have hi-res actors and are using transparency you probably need to turn this on. Defaults to Off.
- `int = obj.GetWrite3DPropsAsRasterImage ()` - Turn on/off writing 3D props as raster images. 2D props are rendered using vector graphics primitives. If you have hi-res actors and are using transparency you probably need to turn this on. Defaults to Off.
- `obj.Write3DPropsAsRasterImageOn ()` - Turn on/off writing 3D props as raster images. 2D props are rendered using vector graphics primitives. If you have hi-res actors and are using transparency you probably need to turn this on. Defaults to Off.
- `obj.Write3DPropsAsRasterImageOff ()` - Turn on/off writing 3D props as raster images. 2D props are rendered using vector graphics primitives. If you have hi-res actors and are using transparency you probably need to turn this on. Defaults to Off.

39.42 vtkGLSLShader

39.42.1 Usage

vtkGLSLShader is a concrete class that creates and compiles hardware shaders written in the OpenGL Shadering Language (GLSL, OpenGL2.0). While step linking a vertex and a fragment shader is performed by vtkGLSLShaderProgram, all shader parameters are initialized in this class.

.Section vtkOpenGLExtensionManager All OpenGL calls are made through vtkOpenGLExtensionManager.

.Section Supported Basic Shader Types:

Scalar Types uniform float uniform int uniform int – boolean scalar not yet tested

Vector Types: uniform vec2—3—4 uniform ivec2—3—4 uniform bvec2—3—4 – boolean vector not yet tested

Matrix Types: uniform mat2—3—4

Texture Samplers: sample1D – Not yet implemented in this cless. sample2D – Not yet implemented in this class. sample3D – Not yet implemented in this class. sampler1DShadow – Not yet implemented in this class. sampler1DShadow – Not yet implemented in this class.

User-Defined structures: uniform struct NOTE: these must be defined and declared outside of the 'main' shader function.

.SECTION Thanks Shader support in VTK includes key contributions by Gary Templet at Sandia National Labs.

To create an instance of class vtkGLSLShader, simply invoke its constructor as follows

```
obj = vtkGLSLShader
```

39.42.2 Methods

The class vtkGLSLShader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGLSLShader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGLSLShader = obj.NewInstance ()`
- `vtkGLSLShader = obj.SafeDownCast (vtkObject o)`
- `int = obj.Compile ()` - Called to compile the shader code. The subclasses must only compile the code in this method. Returns if the compile was successful. Subclasses should compile the code only if it was not already compiled.
- `int = obj.GetHandle ()` - The Shader needs the id of the ShaderProgram to obtain uniform variable locations. This is set by vtkGLSLShaderProgram.
- `obj.SetProgram (int )` - The Shader needs the id of the ShaderProgram to obtain uniform variable locations. This is set by vtkGLSLShaderProgram.
- `int = obj.GetProgram ()` - The Shader needs the id of the ShaderProgram to obtain uniform variable locations. This is set by vtkGLSLShaderProgram.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.

39.43 vtkGLSLShaderDeviceAdapter

39.43.1 Usage

vtkShaderDeviceAdapter subclass for GLSL. .SECTION Thanks Support for generic vertex attributes in VTK was contributed in collaboration with Stephane Ploix at EDF.

To create an instance of class vtkGLSLShaderDeviceAdapter, simply invoke its constructor as follows

```
obj = vtkGLSLShaderDeviceAdapter
```

39.43.2 Methods

The class vtkGLSLShaderDeviceAdapter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGLSLShaderDeviceAdapter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkGLSLShaderDeviceAdapter = obj.NewInstance ()`
- `vtkGLSLShaderDeviceAdapter = obj.SafeDownCast (vtkObject o)`
- `obj.PrepareForRender ()`

39.44 vtkGLSLShaderDeviceAdapter2

39.44.1 Usage

vtkShaderDeviceAdapter subclass for vtkShaderProgram2.

To create an instance of class vtkGLSLShaderDeviceAdapter2, simply invoke its constructor as follows

```
obj = vtkGLSLShaderDeviceAdapter2
```

39.44.2 Methods

The class vtkGLSLShaderDeviceAdapter2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGLSLShaderDeviceAdapter2 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGLSLShaderDeviceAdapter2 = obj.NewInstance ()`
- `vtkGLSLShaderDeviceAdapter2 = obj.SafeDownCast (vtkObject o)`
- `obj.PrepareForRender ()`

39.45 vtkGLSLShaderProgram

39.45.1 Usage

vtkGLSLShaderProgram is a concerete implementation of vtkShaderProgram. It's main function is to 'Link' a vertex and a fragment shader together and install them into the rendering pipeline by calling OpenGL2.0.

Initialization of shader parameters is delegated to instances of vtkShader (vtkGLSLShader in this case). .SECTION Thanks Shader support in VTK includes key contributions by Gary Templet at Sandia National Labs.

To create an instance of class vtkGLSLShaderProgram, simply invoke its constructor as follows

```
obj = vtkGLSLShaderProgram
```

39.45.2 Methods

The class vtkGLSLShaderProgram has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGLSLShaderProgram class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGLSLShaderProgram = obj.NewInstance ()`

- `vtkGLSLShaderProgram = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkActor actor, vtkRenderer renderer)`
- `obj.PostRender (vtkActor , vtkRenderer )` - Called to unload the shaders after the actor has been rendered.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `int = obj.GetProgram ()`

39.46 vtkGPUInfo

39.46.1 Usage

vtkGPUInfo stores information about GPU Video RAM. An host can have several GPUs. The values are set by vtkGPUInfoList.

To create an instance of class vtkGPUInfo, simply invoke its constructor as follows

```
obj = vtkGPUInfo
```

39.46.2 Methods

The class vtkGPUInfo has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGPUInfo class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGPUInfo = obj.NewInstance ()`
- `vtkGPUInfo = obj.SafeDownCast (vtkObject o)`
- `obj.SetDedicatedVideoMemory (vtkIdType )` - Set/Get dedicated video memory in bytes. Initial value is 0. Usually the fastest one. If it is not null, it should be take into account first and DedicatedSystemMemory or SharedSystemMemory should be ignored.
- `vtkIdType = obj.GetDedicatedVideoMemory ()` - Set/Get dedicated video memory in bytes. Initial value is 0. Usually the fastest one. If it is not null, it should be take into account first and DedicatedSystemMemory or SharedSystemMemory should be ignored.
- `obj.SetDedicatedSystemMemory (vtkIdType )` - Set/Get dedicated system memory in bytes. Initial value is 0. This is slow memory. If it is not null, this value should be taken into account only if there is no DedicatedVideoMemory and SharedSystemMemory should be ignored.
- `vtkIdType = obj.GetDedicatedSystemMemory ()` - Set/Get dedicated system memory in bytes. Initial value is 0. This is slow memory. If it is not null, this value should be taken into account only if there is no DedicatedVideoMemory and SharedSystemMemory should be ignored.
- `obj.SetSharedSystemMemory (vtkIdType )` - Set/Get shared system memory in bytes. Initial value is 0. Slowest memory. This value should be taken into account only if there is neither DedicatedVideoMemory nor DedicatedSystemMemory.
- `vtkIdType = obj.GetSharedSystemMemory ()` - Set/Get shared system memory in bytes. Initial value is 0. Slowest memory. This value should be taken into account only if there is neither DedicatedVideoMemory nor DedicatedSystemMemory.

39.47 vtkGPUInfoList

39.47.1 Usage

vtkGPUInfoList stores a list of vtkGPUInfo. An host can have several GPUs. It creates and sets the list by probing the host with system calls. This an abstract class. Concrete classes are OS specific.

To create an instance of class vtkGPUInfoList, simply invoke its constructor as follows

```
obj = vtkGPUInfoList
```

39.47.2 Methods

The class vtkGPUInfoList has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGPUInfoList class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGPUInfoList = obj.NewInstance ()`
- `vtkGPUInfoList = obj.SafeDownCast (vtkObject o)`
- `obj.Probe ()` - Build the list of vtkInfoGPU if not done yet. Default implementation created an empty list. Useful if there is no implementation available for a given architecture yet.
- `bool = obj.IsProbed ()` - Tells if the operating system has been probed. Initial value is false.
- `int = obj.GetNumberOfGPUs ()` - Return the number of GPUs.
- `vtkGPUInfo = obj.GetGPUInfo (int i)` - Return information about GPU i.

39.48 vtkGraphicsFactory

39.48.1 Usage

To create an instance of class vtkGraphicsFactory, simply invoke its constructor as follows

```
obj = vtkGraphicsFactory
```

39.48.2 Methods

The class vtkGraphicsFactory has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphicsFactory class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphicsFactory = obj.NewInstance ()`
- `vtkGraphicsFactory = obj.SafeDownCast (vtkObject o)`

39.49 vtkGraphMapper

39.49.1 Usage

vtkGraphMapper is a mapper to map vtkGraph (and all derived classes) to graphics primitives.

To create an instance of class vtkGraphMapper, simply invoke its constructor as follows

```
obj = vtkGraphMapper
```

39.49.2 Methods

The class vtkGraphMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphMapper = obj.NewInstance ()`
- `vtkGraphMapper = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer ren, vtkActor act)`
- `obj.SetVertexColorArrayName (string name)` - The array to use for coloring vertices. Default is "color".
- `string = obj.GetVertexColorArrayName ()` - The array to use for coloring vertices. Default is "color".
- `obj.SetColorVertices (bool vis)` - Whether to color vertices. Default is off.
- `bool = obj.GetColorVertices ()` - Whether to color vertices. Default is off.
- `obj.ColorVerticesOn ()` - Whether to color vertices. Default is off.
- `obj.ColorVerticesOff ()` - Whether to color vertices. Default is off.
- `obj.SetScaledGlyphs (bool arg)` - Whether scaled glyphs are on or not. Default is off. By default this mapper uses vertex glyphs that do not scale. If you turn this option on you will get circles at each vertex and they will scale as you zoom in/out.
- `bool = obj.GetScaledGlyphs ()` - Whether scaled glyphs are on or not. Default is off. By default this mapper uses vertex glyphs that do not scale. If you turn this option on you will get circles at each vertex and they will scale as you zoom in/out.
- `obj.ScaledGlyphsOn ()` - Whether scaled glyphs are on or not. Default is off. By default this mapper uses vertex glyphs that do not scale. If you turn this option on you will get circles at each vertex and they will scale as you zoom in/out.
- `obj.ScaledGlyphsOff ()` - Whether scaled glyphs are on or not. Default is off. By default this mapper uses vertex glyphs that do not scale. If you turn this option on you will get circles at each vertex and they will scale as you zoom in/out.
- `obj.SetScalingArrayName (string )` - Glyph scaling array name. Default is "scale"
- `string = obj.GetScalingArrayName ()` - Glyph scaling array name. Default is "scale"
- `obj.SetEdgeVisibility (bool vis)` - Whether to show edges or not. Default is on.

- `bool = obj.GetEdgeVisibility ()` - Whether to show edges or not. Default is on.
- `obj.EdgeVisibilityOn ()` - Whether to show edges or not. Default is on.
- `obj.EdgeVisibilityOff ()` - Whether to show edges or not. Default is on.
- `obj.SetEdgeColorArrayName (string name)` - The array to use for coloring edges. Default is "color".
- `string = obj.GetEdgeColorArrayName ()` - The array to use for coloring edges. Default is "color".
- `obj.SetColorEdges (bool vis)` - Whether to color edges. Default is off.
- `bool = obj.GetColorEdges ()` - Whether to color edges. Default is off.
- `obj.ColorEdgesOn ()` - Whether to color edges. Default is off.
- `obj.ColorEdgesOff ()` - Whether to color edges. Default is off.
- `obj.SetEnabledEdgesArrayName (string )` - The array to use for coloring edges. Default is "color".
- `string = obj.GetEnabledEdgesArrayName ()` - The array to use for coloring edges. Default is "color".
- `obj.SetEnableEdgesByArray (int )` - Whether to enable/disable edges using array values. Default is off.
- `int = obj.GetEnableEdgesByArray ()` - Whether to enable/disable edges using array values. Default is off.
- `obj.EnableEdgesByArrayOn ()` - Whether to enable/disable edges using array values. Default is off.
- `obj.EnableEdgesByArrayOff ()` - Whether to enable/disable edges using array values. Default is off.
- `obj.SetEnabledVerticesArrayName (string )` - The array to use for coloring edges. Default is "color".
- `string = obj.GetEnabledVerticesArrayName ()` - The array to use for coloring edges. Default is "color".
- `obj.SetEnableVerticesByArray (int )` - Whether to enable/disable vertices using array values. Default is off.
- `int = obj.GetEnableVerticesByArray ()` - Whether to enable/disable vertices using array values. Default is off.
- `obj.EnableVerticesByArrayOn ()` - Whether to enable/disable vertices using array values. Default is off.
- `obj.EnableVerticesByArrayOff ()` - Whether to enable/disable vertices using array values. Default is off.
- `obj.SetIconArrayName (string name)` - The array to use for assigning icons.
- `string = obj.GetIconArrayName ()` - The array to use for assigning icons.
- `obj.AddIconType (string type, int index)` - Associate the icon at index "index" in the vtkTexture to all vertices containing "type" as a value in the vertex attribute array specified by IconArrayName.
- `obj.ClearIconTypes ()` - Clear all icon mappings.
- `obj.SetIconSize (int size)` - Specify the Width and Height, in pixels, of an icon in the icon sheet.

- `obj.SetIconAlignment (int alignment)` - Specify where the icons should be placed in relation to the vertex. See vtkIconGlyphFilter.h for possible values.
- `vtkTexture = obj.GetIconTexture ()` - The texture containing the icon sheet.
- `obj.SetIconTexture (vtkTexture texture)` - The texture containing the icon sheet.
- `obj.SetIconVisibility (bool vis)` - Whether to show icons. Default is off.
- `bool = obj.GetIconVisibility ()` - Whether to show icons. Default is off.
- `obj.IconVisibilityOn ()` - Whether to show icons. Default is off.
- `obj.IconVisibilityOff ()` - Whether to show icons. Default is off.
- `float = obj.GetVertexPointSize ()` - Get/Set the vertex point size
- `obj.SetVertexPointSize (float size)` - Get/Set the vertex point size
- `float = obj.GetEdgeLineWidth ()` - Get/Set the edge line width
- `obj.SetEdgeLineWidth (float width)` - Get/Set the edge line width
- `obj.ApplyViewTheme (vtkViewTheme theme)` - Apply the theme to this view.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this mapper. The parameter window could be used to determine which graphic resources to release.
- `long = obj.GetMTime ()` - Get the mtime also considering the lookup table.
- `obj.SetInput (vtkGraph input)` - Set the Input of this mapper.
- `vtkGraph = obj.GetInput ()` - Set the Input of this mapper.
- `double = obj.GetBounds ()` - Return bounding box (array of six doubles) of data expressed as (xmin,xmax, ymin,ymax, zmin,zmax).
- `obj.GetBounds (double bounds)` - Access to the lookup tables used by the vertex and edge mappers.
- `vtkLookupTable = obj.GetEdgeLookupTable ()` - Access to the lookup tables used by the vertex and edge mappers.
- `vtkLookupTable = obj.GetVertexLookupTable ()` - Access to the lookup tables used by the vertex and edge mappers.

39.50 vtkGraphToGlyphs

39.50.1 Usage

Converts a vtkGraph to a vtkPolyData containing a glyph for each vertex. This assumes that the points of the graph have already been filled (perhaps by vtkGraphLayout). The glyphs will automatically be scaled to be the same size in screen coordinates. To do this the filter requires a pointer to the renderer into which the glyphs will be rendered.

To create an instance of class vtkGraphToGlyphs, simply invoke its constructor as follows

```
obj = vtkGraphToGlyphs
```

39.50.2 Methods

The class vtkGraphToGlyphs has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphToGlyphs class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphToGlyphs = obj.NewInstance ()`
- `vtkGraphToGlyphs = obj.SafeDownCast (vtkObject o)`
- `obj.SetGlyphType (int )` - The glyph type, specified as one of the enumerated values in this class. VERTEX is a special glyph that cannot be scaled, but instead is rendered as an OpenGL vertex primitive. This may appear as a box or circle depending on the hardware.
- `int = obj.GetGlyphType ()` - The glyph type, specified as one of the enumerated values in this class. VERTEX is a special glyph that cannot be scaled, but instead is rendered as an OpenGL vertex primitive. This may appear as a box or circle depending on the hardware.
- `obj.SetFilled (bool )` - Whether to fill the glyph, or to just render the outline.
- `bool = obj.GetFilled ()` - Whether to fill the glyph, or to just render the outline.
- `obj.FilledOn ()` - Whether to fill the glyph, or to just render the outline.
- `obj.FilledOff ()` - Whether to fill the glyph, or to just render the outline.
- `obj.SetScreenSize (double )` - Set the desired screen size of each glyph. If you are using scaling, this will be the size of the glyph when rendering an object with scaling value 1.0.
- `double = obj.GetScreenSize ()` - Set the desired screen size of each glyph. If you are using scaling, this will be the size of the glyph when rendering an object with scaling value 1.0.
- `obj.SetRenderer (vtkRenderer ren)` - The renderer in which the glyphs will be placed.
- `vtkRenderer = obj.GetRenderer ()` - The renderer in which the glyphs will be placed.
- `obj.SetScaling (bool b)` - Whether to use the input array to process in order to scale the vertices.
- `bool = obj.GetScaling ()` - Whether to use the input array to process in order to scale the vertices.
- `long = obj.GetMTime ()` - The modified time of this filter.

39.51 vtkHardwareSelectionPolyDataPainter

39.51.1 Usage

vtkHardwareSelectionPolyDataPainter is a painter for polydata used when rendering hardware selection passes.

To create an instance of class vtkHardwareSelectionPolyDataPainter, simply invoke its constructor as follows

```
obj = vtkHardwareSelectionPolyDataPainter
```

39.51.2 Methods

The class vtkHardwareSelectionPolyDataPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHardwareSelectionPolyDataPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHardwareSelectionPolyDataPainter = obj.NewInstance ()`
- `vtkHardwareSelectionPolyDataPainter = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnableSelection (int )` - Enable/Disable vtkHardwareSelector class. Useful when using this painter as an internal painter. Default is enabled.
- `int = obj.GetEnableSelection ()` - Enable/Disable vtkHardwareSelector class. Useful when using this painter as an internal painter. Default is enabled.
- `obj.EnableSelectionOn ()` - Enable/Disable vtkHardwareSelector class. Useful when using this painter as an internal painter. Default is enabled.
- `obj.EnableSelectionOff ()` - Enable/Disable vtkHardwareSelector class. Useful when using this painter as an internal painter. Default is enabled.

39.52 vtkHardwareSelector

39.52.1 Usage

vtkHardwareSelector is a helper that orchestrates color buffer based selection. This relies on OpenGL. vtkHardwareSelector can be used to select visible cells or points within a given rectangle of the RenderWindow. To use it, call in order: SetRenderer() - to select the renderer in which we want to select the cells/points. SetArea() - to set the rectangular region in the render window to select in. SetFieldAssociation() - to select the attribute to select i.e. cells/points etc. Finally, call Select(). Select will cause the attached vtkRenderer to render in a special color mode, where each cell/point is given it own color so that later inspection of the Rendered Pixels can determine what cells are visible. Select() returns a new vtkSelection instance with the cells/points selected.

Limitations: Antialiasing will break this class. If your graphics card settings force their use this class will return invalid results.

Currently only cells from PolyDataMappers can be selected from. When vtkRenderer::Selector is non-null vtkPainterPolyDataMapper uses the vtkHardwareSelectionPolyDataPainter which make appropriate calls to BeginRenderProp(), EndRenderProp(), RenderAttributeId() to render colors correctly. Until alternatives to vtkHardwareSelectionPolyDataPainter exist that can do a similar coloration of other vtkDataSet types, only polygonal data can be selected. If you need to select other data types, consider using vtkDataSetMapper and turning on it's PassThroughCellIds feature, or using vtkFrustumExtractor.

Only Opaque geometry in Actors is selected from. Assemblies and LODMappers are not currently supported.

During selection, visible datasets that can not be selected from are temporarily hidden so as not to produce invalid indices from their colors.

To create an instance of class vtkHardwareSelector, simply invoke its constructor as follows

```
obj = vtkHardwareSelector
```

39.52.2 Methods

The class vtkHardwareSelector has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHardwareSelector class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHardwareSelector = obj.NewInstance ()`
- `vtkHardwareSelector = obj.SafeDownCast (vtkObject o)`
- `obj.SetRenderer (vtkRenderer )` - Get/Set the renderer to perform the selection on.
- `vtkRenderer = obj.GetRenderer ()` - Get/Set the renderer to perform the selection on.
- `obj.SetArea (int , int , int , int )` - Get/Set the area to select as (xmin, ymin, xmax, ymax).
- `obj.SetArea (int  a[4])` - Get/Set the area to select as (xmin, ymin, xmax, ymax).
- `int = obj. GetArea ()` - Get/Set the area to select as (xmin, ymin, xmax, ymax).
- `obj.SetFieldAssociation (int )` - Set the field type to select. Valid values are vtkDataObject::FIELD_ASSOCIATIO vtkDataObject::FIELD_ASSOCIATION_CELLS vtkDataObject::FIELD_ASSOCIATION_VERTICES vtkDataObject::FIELD_ASSOCIATION_EDGES vtkDataObject::FIELD_ASSOCIATION_ROWS Currently only FIELD_ASSOCIATION_POINTS and FIELD_ASSOCIATION_CELLS are supported.
- `int = obj.GetFieldAssociation ()` - Set the field type to select. Valid values are vtkDataObject::FIELD_ASSOCIATION_POINTS vtkDataObject::FIELD_ASSOCIATION_CELLS vtkDataObject::FIELD_ASSO vtkDataObject::FIELD_ASSOCIATION_EDGES vtkDataObject::FIELD_ASSOCIATION_ROWS Currently only FIELD_ASSOCIATION_POINTS and FIELD_ASSOCIATION_CELLS are supported.
- `vtkSelection = obj.Select ()` - Perform the selection. Returns a new instance of vtkSelection containing the selection on success.
- `bool = obj.CaptureBuffers ()` - It is possible to use the vtkHardwareSelector for a custom picking. (Look at vtkScenePicker). In that case instead of Select() on can use CaptureBuffers() to render the selection buffers and then get information about pixel locations suing GetPixelInformation(). Use ClearBuffers() to clear buffers after one's done with the scene. The optional final parameter maxDist will look for a cell within the specified number of pixels from display_position.
- `obj.ClearBuffers ()` - Called by any vtkMapper or vtkProp subclass to render an attribute's id.
- `obj.RenderAttributeId (vtkIdType attribid)` - Called by any vtkMapper or vtkProp subclass to render an attribute's id.
- `obj.BeginRenderProp ()` - Called by the mapper (vtkHardwareSelectionPolyDataPainter) before and after rendering each prop.
- `obj.EndRenderProp ()` - Called by the mapper (vtkHardwareSelectionPolyDataPainter) before and after rendering each prop.
- `obj.SetProcessID (int )` - Get/Set the process id. If process id ¡ 0 (default -1), then the PROCESS_PASS is not rendered.
- `int = obj.GetProcessID ()` - Get/Set the process id. If process id ¡ 0 (default -1), then the PROCESS_PASS is not rendered.
- `int = obj.GetCurrentPass ()` - Get the current pass number.

- `vtkSelection = obj.GenerateSelection ()` - Generates the vtkSelection from pixel buffers. Requires that CaptureBuffers() has already been called. Optionally you may pass a screen region (xmin, ymin, xmax, ymax) to generate a selection from. The region must be a subregion of the region specified by SetArea(), otherwise it will be clipped to that region.

- `vtkSelection = obj.GenerateSelection (int r[4])` - Generates the vtkSelection from pixel buffers. Requires that CaptureBuffers() has already been called. Optionally you may pass a screen region (xmin, ymin, xmax, ymax) to generate a selection from. The region must be a subregion of the region specified by SetArea(), otherwise it will be clipped to that region.

- `vtkSelection = obj.GenerateSelection (int x1, int y1, int x2, int y2)` - Generates the vtkSelection from pixel buffers. Requires that CaptureBuffers() has already been called. Optionally you may pass a screen region (xmin, ymin, xmax, ymax) to generate a selection from. The region must be a subregion of the region specified by SetArea(), otherwise it will be clipped to that region.

39.53 vtkHierarchicalPolyDataMapper

39.53.1 Usage

Legacy class. Use vtkCompositePolyDataMapper instead.

To create an instance of class vtkHierarchicalPolyDataMapper, simply invoke its constructor as follows

```
obj = vtkHierarchicalPolyDataMapper
```

39.53.2 Methods

The class vtkHierarchicalPolyDataMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalPolyDataMapper class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkHierarchicalPolyDataMapper = obj.NewInstance ()`

- `vtkHierarchicalPolyDataMapper = obj.SafeDownCast (vtkObject o)`

39.54 vtkIdentColoredPainter

39.54.1 Usage

DEPRECATED. Refer to vtkHardwareSelectionPolyDataPainter instead. This painter will color each polygon in a color that encodes an integer. Doing so allows us to determine what polygon is behind each pixel on the screen.

To create an instance of class vtkIdentColoredPainter, simply invoke its constructor as follows

```
obj = vtkIdentColoredPainter
```

39.54.2 Methods

The class vtkIdentColoredPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIdentColoredPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIdentColoredPainter = obj.NewInstance ()`
- `vtkIdentColoredPainter = obj.SafeDownCast (vtkObject o)`
- `obj.ResetCurrentId ()`
- `obj.ColorByConstant (int constant)`
- `obj.ColorByIncreasingIdent (int plane)`
- `obj.ColorByActorId (vtkProp ActorId)`
- `obj.ColorByVertex ()`
- `vtkProp = obj.GetActorFromId (vtkIdType id)`

39.55 vtkImageActor

39.55.1 Usage

vtkImageActor is used to render an image in a 3D scene. The image is placed at the origin of the image, and its size is controlled by the image dimensions and image spacing. The orientation of the image is orthogonal to one of the x-y-z axes depending on which plane the image is defined in. vtkImageActor duplicates the functionality of combinations of other VTK classes in a convenient, single class.

To create an instance of class vtkImageActor, simply invoke its constructor as follows

```
obj = vtkImageActor
```

39.55.2 Methods

The class vtkImageActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageActor = obj.NewInstance ()`
- `vtkImageActor = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImageData )` - Set/Get the image data input for the image actor.
- `vtkImageData = obj.GetInput ()` - Set/Get the image data input for the image actor.
- `int = obj.GetInterpolate ()` - Turn on/off linear interpolation of the image when rendering.
- `obj.SetInterpolate (int )` - Turn on/off linear interpolation of the image when rendering.
- `obj.InterpolateOn ()` - Turn on/off linear interpolation of the image when rendering.
- `obj.InterpolateOff ()` - Turn on/off linear interpolation of the image when rendering.
- `obj.SetOpacity (double )` - Set/Get the object's opacity. 1.0 is totally opaque and 0.0 is completely transparent.

- `double = obj.GetOpacityMinValue ()` - Set/Get the object's opacity. 1.0 is totally opaque and 0.0 is completely transparent.
- `double = obj.GetOpacityMaxValue ()` - Set/Get the object's opacity. 1.0 is totally opaque and 0.0 is completely transparent.
- `double = obj.GetOpacity ()` - Set/Get the object's opacity. 1.0 is totally opaque and 0.0 is completely transparent.
- `obj.SetDisplayExtent (int extent[6])` - The image extent is generally set explicitly, but if not set it will be determined from the input image data.
- `obj.SetDisplayExtent (int minX, int maxX, int minY, int maxY, int minZ, int maxZ)` - The image extent is generally set explicitly, but if not set it will be determined from the input image data.
- `obj.GetDisplayExtent (int extent[6])` - The image extent is generally set explicitly, but if not set it will be determined from the input image data.
- `int = obj.GetDisplayExtent ()` - Get the bounds of this image actor. Either copy the bounds into a user provided array or return a pointer to an array. In either case the boudns is expressed as a 6-vector (xmin,xmax, ymin,ymax, zmin,zmax).
- `double = obj.GetBounds ()` - Get the bounds of this image actor. Either copy the bounds into a user provided array or return a pointer to an array. In either case the boudns is expressed as a 6-vector (xmin,xmax, ymin,ymax, zmin,zmax).
- `obj.GetBounds (double bounds[6])` - Get the bounds of this image actor. Either copy the bounds into a user provided array or return a pointer to an array. In either case the boudns is expressed as a 6-vector (xmin,xmax, ymin,ymax, zmin,zmax).
- `obj.GetDisplayBounds (double bounds[6])` - Get the bounds of the data that is displayed by this image actor. If the transformation matrix for this actor is the identity matrix, this will return the same value as GetBounds.
- `int = obj.GetSliceNumber ()` - Return the slice number (& min/max slice number) computed from the display extent.
- `int = obj.GetSliceNumberMax ()` - Return the slice number (& min/max slice number) computed from the display extent.
- `int = obj.GetSliceNumberMin ()` - Return the slice number (& min/max slice number) computed from the display extent.
- `obj.SetZSlice (int z)` - Set/Get the current slice number. The axis Z in ZSlice does not necessarily have any relation to the z axis of the data on disk. It is simply the axis orthogonal to the x,y, display plane. GetWholeZMax and Min are convenience methods for obtaining the number of slices that can be displayed. Again the number of slices is in reference to the display z axis, which is not necessarily the z axis on disk. (due to reformatting etc)
- `int = obj.GetZSlice ()` - Set/Get the current slice number. The axis Z in ZSlice does not necessarily have any relation to the z axis of the data on disk. It is simply the axis orthogonal to the x,y, display plane. GetWholeZMax and Min are convenience methods for obtaining the number of slices that can be displayed. Again the number of slices is in reference to the display z axis, which is not necessarily the z axis on disk. (due to reformatting etc)
- `int = obj.GetWholeZMin ()` - Set/Get the current slice number. The axis Z in ZSlice does not necessarily have any relation to the z axis of the data on disk. It is simply the axis orthogonal to the x,y, display plane. GetWholeZMax and Min are convenience methods for obtaining the number of slices that can be displayed. Again the number of slices is in reference to the display z axis, which is not necessarily the z axis on disk. (due to reformatting etc)

- `int = obj.GetWholeZMax ()` - Set/Get the current slice number. The axis Z in ZSlice does not necessarily have any relation to the z axis of the data on disk. It is simply the axis orthogonal to the x,y, display plane. GetWholeZMax and Min are convenience methods for obtaining the number of slices that can be displayed. Again the number of slices is in reference to the display z axis, which is not necessarily the z axis on disk. (due to reformatting etc)

39.56 vtkImageMapper

39.56.1 Usage

vtkImageMapper provides 2D image display support for vtk. It is a Mapper2D subclass that can be associated with an Actor2D and placed within a RenderWindow or ImageWindow.

To create an instance of class vtkImageMapper, simply invoke its constructor as follows

```
obj = vtkImageMapper
```

39.56.2 Methods

The class vtkImageMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageMapper = obj.NewInstance ()`
- `vtkImageMapper = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Override Modifiedtime as we have added a lookuptable
- `obj.SetColorWindow (double )` - Set/Get the window value for window/level
- `double = obj.GetColorWindow ()` - Set/Get the window value for window/level
- `obj.SetColorLevel (double )` - Set/Get the level value for window/level
- `double = obj.GetColorLevel ()` - Set/Get the level value for window/level
- `obj.SetZSlice (int )` - Set/Get the current slice number. The axis Z in ZSlice does not necessarily have any relation to the z axis of the data on disk. It is simply the axis orthogonal to the x,y, display plane. GetWholeZMax and Min are convenience methods for obtaining the number of slices that can be displayed. Again the number of slices is in reference to the display z axis, which is not necessarily the z axis on disk. (due to reformatting etc)
- `int = obj.GetZSlice ()` - Set/Get the current slice number. The axis Z in ZSlice does not necessarily have any relation to the z axis of the data on disk. It is simply the axis orthogonal to the x,y, display plane. GetWholeZMax and Min are convenience methods for obtaining the number of slices that can be displayed. Again the number of slices is in reference to the display z axis, which is not necessarily the z axis on disk. (due to reformatting etc)
- `int = obj.GetWholeZMin ()` - Set/Get the current slice number. The axis Z in ZSlice does not necessarily have any relation to the z axis of the data on disk. It is simply the axis orthogonal to the x,y, display plane. GetWholeZMax and Min are convenience methods for obtaining the number of slices that can be displayed. Again the number of slices is in reference to the display z axis, which is not necessarily the z axis on disk. (due to reformatting etc)

- `int = obj.GetWholeZMax ()` - Set/Get the current slice number. The axis Z in ZSlice does not necessarily have any relation to the z axis of the data on disk. It is simply the axis orthogonal to the x,y, display plane. GetWholeZMax and Min are convenience methods for obtaining the number of slices that can be displayed. Again the number of slices is in reference to the display z axis, which is not necessarily the z axis on disk. (due to reformatting etc)

- `obj.RenderStart (vtkViewport viewport, vtkActor2D actor)` - Draw the image to the screen.

- `obj.RenderData (vtkViewport , vtkImageData , vtkActor2D )` - Function called by Render to actually draw the image to to the screen

- `double = obj.GetColorShift ()` - Methods used internally for performing the Window/Level mapping.

- `double = obj.GetColorScale ()` - Methods used internally for performing the Window/Level mapping.

- `obj.SetInput (vtkImageData input)` - Set the Input of a filter.

- `vtkImageData = obj.GetInput ()` - Set the Input of a filter.

- `obj.SetRenderToRectangle (int )` - If RenderToRectangle is set (by default not), then the imagemapper will render the image into the rectangle supplied by the Actor2D's PositionCoordinate and Position2Coordinate

- `int = obj.GetRenderToRectangle ()` - If RenderToRectangle is set (by default not), then the imagemapper will render the image into the rectangle supplied by the Actor2D's PositionCoordinate and Position2Coordinate

- `obj.RenderToRectangleOn ()` - If RenderToRectangle is set (by default not), then the imagemapper will render the image into the rectangle supplied by the Actor2D's PositionCoordinate and Position2Coordinate

- `obj.RenderToRectangleOff ()` - If RenderToRectangle is set (by default not), then the imagemapper will render the image into the rectangle supplied by the Actor2D's PositionCoordinate and Position2Coordinate

- `obj.SetUseCustomExtents (int )` - Usually, the entire image is displayed, if UseCustomExtents is set (by default not), then the region supplied in the CustomDisplayExtents is used in preference. Note that the Custom extents are x,y only and the zslice is still applied

- `int = obj.GetUseCustomExtents ()` - Usually, the entire image is displayed, if UseCustomExtents is set (by default not), then the region supplied in the CustomDisplayExtents is used in preference. Note that the Custom extents are x,y only and the zslice is still applied

- `obj.UseCustomExtentsOn ()` - Usually, the entire image is displayed, if UseCustomExtents is set (by default not), then the region supplied in the CustomDisplayExtents is used in preference. Note that the Custom extents are x,y only and the zslice is still applied

- `obj.UseCustomExtentsOff ()` - Usually, the entire image is displayed, if UseCustomExtents is set (by default not), then the region supplied in the CustomDisplayExtents is used in preference. Note that the Custom extents are x,y only and the zslice is still applied

- `obj.SetCustomDisplayExtents (int [4])` - The image extents which should be displayed with UseCustomExtents Note that the Custom extents are x,y only and the zslice is still applied

- `int = obj. GetCustomDisplayExtents ()` - The image extents which should be displayed with UseCustomExtents Note that the Custom extents are x,y only and the zslice is still applied

39.57 vtkImageProcessingPass

39.57.1 Usage

Abstract class with some convenient methods frequently used in subclasses.
.SECTION Implementation
To create an instance of class vtkImageProcessingPass, simply invoke its constructor as follows

```
obj = vtkImageProcessingPass
```

39.57.2 Methods

The class vtkImageProcessingPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageProcessingPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageProcessingPass = obj.NewInstance ()`
- `vtkImageProcessingPass = obj.SafeDownCast (vtkObject o)`
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Release graphics resources and ask components to release their own resources.
- `vtkRenderPass = obj.GetDelegatePass ()` - Delegate for rendering the image to be processed. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkCameraPass or to a post-processing pass. Initial value is a NULL pointer.
- `obj.SetDelegatePass (vtkRenderPass delegatePass)` - Delegate for rendering the image to be processed. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkCameraPass or to a post-processing pass. Initial value is a NULL pointer.

39.58 vtkImageViewer

39.58.1 Usage

vtkImageViewer is a convenience class for displaying a 2d image. It packages up the functionality found in vtkRenderWindow, vtkRenderer, vtkActor2D and vtkImageMapper into a single easy to use class. Behind the scenes these four classes are actually used to to provide the required functionality. vtkImageViewer is simply a wrapper around them.

To create an instance of class vtkImageViewer, simply invoke its constructor as follows

```
obj = vtkImageViewer
```

39.58.2 Methods

The class vtkImageViewer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageViewer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkImageViewer = obj.NewInstance ()`
- `vtkImageViewer = obj.SafeDownCast (vtkObject o)`
- `string = obj.GetWindowName ()` - Get name of rendering window
- `obj.Render (void )` - Render the resulting image.
- `obj.SetInput (vtkImageData in)` - Set/Get the input to the viewer.
- `vtkImageData = obj.GetInput ()` - Set/Get the input to the viewer.
- `obj.SetInputConnection (vtkAlgorithmOutput input)` - Set/Get the input to the viewer.
- `int = obj.GetWholeZMin ()` - What is the possible Min/ Max z slices available.
- `int = obj.GetWholeZMax ()` - What is the possible Min/ Max z slices available.
- `int = obj.GetZSlice ()` - Set/Get the current Z Slice to display
- `obj.SetZSlice (int s)` - Set/Get the current Z Slice to display
- `double = obj.GetColorWindow ()` - Sets window/level for mapping pixels to colors.
- `double = obj.GetColorLevel ()` - Sets window/level for mapping pixels to colors.
- `obj.SetColorWindow (double s)` - Sets window/level for mapping pixels to colors.
- `obj.SetColorLevel (double s)` - Sets window/level for mapping pixels to colors.
- `int = obj.GetGrayScaleHint ()` - By default this is a color viewer. GrayScaleHintOn will improve the appearance of gray scale images on some systems.
- `obj.SetGrayScaleHint (int )` - By default this is a color viewer. GrayScaleHintOn will improve the appearance of gray scale images on some systems.
- `obj.GrayScaleHintOn ()` - By default this is a color viewer. GrayScaleHintOn will improve the appearance of gray scale images on some systems.
- `obj.GrayScaleHintOff ()` - By default this is a color viewer. GrayScaleHintOn will improve the appearance of gray scale images on some systems.
- `int = obj.GetPosition ()` - Set/Get the position in screen coordinates of the rendering window.
- `obj.SetPosition (int a, int b)` - Set/Get the position in screen coordinates of the rendering window.
- `obj.SetPosition (int a[2])` - Set/Get the position in screen coordinates of the rendering window.
- `int = obj.GetSize ()` - Set/Get the size of the window in screen coordinates in pixels.
- `obj.SetSize (int a, int b)` - Set/Get the size of the window in screen coordinates in pixels.
- `obj.SetSize (int a[2])` - Set/Get the size of the window in screen coordinates in pixels.
- `vtkRenderWindow = obj.GetRenderWindow ()` - Get the internal objects
- `vtkRenderer = obj.GetRenderer ()` - Get the internal objects
- `vtkImageMapper = obj.GetImageMapper ()` - Get the internal objects
- `vtkActor2D = obj.GetActor2D ()` - Get the internal objects
- `obj.SetupInteractor (vtkRenderWindowInteractor )` - Create and attach an interactor for this window

- `obj.SetOffScreenRendering (int )` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `int = obj.GetOffScreenRendering ()` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `obj.OffScreenRenderingOn ()` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `obj.OffScreenRenderingOff ()` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.

39.59 vtkImageViewer2

39.59.1 Usage

vtkImageViewer2 is a convenience class for displaying a 2D image. It packages up the functionality found in vtkRenderWindow, vtkRenderer, vtkImageActor and vtkImageMapToWindowLevelColors into a single easy to use class. This class also creates an image interactor style (vtkInteractorStyleImage) that allows zooming and panning of images, and supports interactive window/level operations on the image. Note that vtkImageViewer2 is simply a wrapper around these classes.

vtkImageViewer2 uses the 3D rendering and texture mapping engine to draw an image on a plane. This allows for rapid rendering, zooming, and panning. The image is placed in the 3D scene at a depth based on the z-coordinate of the particular image slice. Each call to SetSlice() changes the image data (slice) displayed AND changes the depth of the displayed slice in the 3D scene. This can be controlled by the AutoAdjustCameraClippingRange ivar of the InteractorStyle member.

It is possible to mix images and geometry, using the methods:

viewer-¿SetInput(myImage); viewer-¿GetRenderer()-¿AddActor(myActor);

This can be used to annotate an image with a PolyData of "edges" or or highlight sections of an image or display a 3D isosurface with a slice from the volume, etc. Any portions of your geometry that are in front of the displayed slice will be visible; any portions of your geometry that are behind the displayed slice will be obscured. A more general framework (with respect to viewing direction) for achieving this effect is provided by the vtkImagePlaneWidget .

Note that pressing 'r' will reset the window/level and pressing shift+'r' or control+'r' will reset the camera.

To create an instance of class vtkImageViewer2, simply invoke its constructor as follows

```
obj = vtkImageViewer2
```

39.59.2 Methods

The class vtkImageViewer2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageViewer2 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageViewer2 = obj.NewInstance ()`
- `vtkImageViewer2 = obj.SafeDownCast (vtkObject o)`

- `string = obj.GetWindowName ()` - Get the name of rendering window.
- `obj.Render (void )` - Render the resulting image.
- `obj.SetInput (vtkImageData in)` - Set/Get the input image to the viewer.
- `vtkImageData = obj.GetInput ()` - Set/Get the input image to the viewer.
- `obj.SetInputConnection (vtkAlgorithmOutput input)` - Set/Get the input image to the viewer.
- `int = obj.GetSliceOrientation ()` - Set/get the slice orientation
- `obj.SetSliceOrientation (int orientation)` - Set/get the slice orientation
- `obj.SetSliceOrientationToXY ()` - Set/get the slice orientation
- `obj.SetSliceOrientationToYZ ()` - Set/get the slice orientation
- `obj.SetSliceOrientationToXZ ()` - Set/get the slice orientation
- `int = obj.GetSlice ()` - Set/Get the current slice to display (depending on the orientation this can be in X, Y or Z).
- `obj.SetSlice (int s)` - Set/Get the current slice to display (depending on the orientation this can be in X, Y or Z).
- `obj.UpdateDisplayExtent ()` - Update the display extent manually so that the proper slice for the given orientation is displayed. It will also try to set a reasonable camera clipping range. This method is called automatically when the Input is changed, but most of the time the input of this class is likely to remain the same, i.e. connected to the output of a filter, or an image reader. When the input of this filter or reader itself is changed, an error message might be displayed since the current display extent is probably outside the new whole extent. Calling this method will ensure that the display extent is reset properly.
- `int = obj.GetSliceMin ()` - Return the minimum and maximum slice values (depending on the orientation this can be in X, Y or Z).
- `int = obj.GetSliceMax ()` - Return the minimum and maximum slice values (depending on the orientation this can be in X, Y or Z).
- `obj.GetSliceRange (int range[2])` - Return the minimum and maximum slice values (depending on the orientation this can be in X, Y or Z).
- `double = obj.GetColorWindow ()` - Set window and level for mapping pixels to colors.
- `double = obj.GetColorLevel ()` - Set window and level for mapping pixels to colors.
- `obj.SetColorWindow (double s)` - Set window and level for mapping pixels to colors.
- `obj.SetColorLevel (double s)` - Set window and level for mapping pixels to colors.
- `obj.SetPosition (int a, int b)` - Set/Get the position in screen coordinates of the rendering window.
- `obj.SetPosition (int a[2])` - Set/Get the size of the window in screen coordinates in pixels.
- `obj.SetSize (int a, int b)` - Set/Get the size of the window in screen coordinates in pixels.
- `obj.SetSize (int a[2])` - Get the internal render window, renderer, image actor, and image map instances.
- `vtkRenderWindow = obj.GetRenderWindow ()` - Get the internal render window, renderer, image actor, and image map instances.

- `vtkRenderer = obj.GetRenderer ()` - Get the internal render window, renderer, image actor, and image map instances.
- `vtkImageActor = obj.GetImageActor ()` - Get the internal render window, renderer, image actor, and image map instances.
- `vtkImageMapToWindowLevelColors = obj.GetWindowLevel ()` - Get the internal render window, renderer, image actor, and image map instances.
- `vtkInteractorStyleImage = obj.GetInteractorStyle ()` - Get the internal render window, renderer, image actor, and image map instances.
- `obj.SetRenderWindow (vtkRenderWindow arg)` - Set your own renderwindow and renderer
- `obj.SetRenderer (vtkRenderer arg)` - Set your own renderwindow and renderer
- `obj.SetupInteractor (vtkRenderWindowInteractor )` - Attach an interactor for the internal render window.
- `obj.SetOffScreenRendering (int )` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `int = obj.GetOffScreenRendering ()` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `obj.OffScreenRenderingOn ()` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `obj.OffScreenRenderingOff ()` - Create a window in memory instead of on the screen. This may not be supported for every type of window and on some windows you may need to invoke this prior to the first render.
- `int = obj.GetWholeZMin ()` - @deprecated Replaced by vtkImageViewer2::GetSliceMin() as of VTK 5.0.
- `int = obj.GetWholeZMax ()` - @deprecated Replaced by vtkImageViewer2::GetSliceMax() as of VTK 5.0.
- `int = obj.GetZSlice ()` - @deprecated Replaced by vtkImageViewer2::GetSlice() as of VTK 5.0.
- `obj.SetZSlice (int )` - @deprecated Replaced by vtkImageViewer2::SetSlice() as of VTK 5.0.

39.60 vtkImagingFactory

39.60.1 Usage

To create an instance of class vtkImagingFactory, simply invoke its constructor as follows

```
obj = vtkImagingFactory
```

39.60.2 Methods

The class vtkImagingFactory has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImagingFactory class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImagingFactory = obj.NewInstance ()`
- `vtkImagingFactory = obj.SafeDownCast (vtkObject o)`

39.61 vtkImporter

39.61.1 Usage

vtkImporter is an abstract class that specifies the protocol for importing actors, cameras, lights and properties into a vtkRenderWindow. The following takes place: 1) Create a RenderWindow and Renderer if none is provided. 2) Call ImportBegin, if ImportBegin returns False, return 3) Call ReadData, which calls: a) Import the Actors b) Import the cameras c) Import the lights d) Import the Properties 7) Call ImportEnd

Subclasses optionally implement the ImportActors, ImportCameras, ImportLights and ImportProperties or ReadData methods. An ImportBegin and ImportEnd can optionally be provided to perform Importer-specific initialization and termination. The Read method initiates the import process. If a RenderWindow is provided, its Renderer will contained the imported objects. If the RenderWindow has no Renderer, one is created. If no RenderWindow is provided, both a RenderWindow and Renderer will be created. Both the RenderWindow and Renderer can be accessed using Get methods.

To create an instance of class vtkImporter, simply invoke its constructor as follows

```
obj = vtkImporter
```

39.61.2 Methods

The class vtkImporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImporter = obj.NewInstance ()`
- `vtkImporter = obj.SafeDownCast (vtkObject o)`
- `vtkRenderer = obj.GetRenderer ()`
- `obj.SetRenderWindow (vtkRenderWindow )`
- `vtkRenderWindow = obj.GetRenderWindow ()`
- `obj.Read ()`
- `obj.Update ()`

39.62 vtkInteractorEventRecorder

39.62.1 Usage

vtkInteractorEventRecorder records all VTK events invoked from a vtkRenderWindowInteractor. The events are recorded to a file. vtkInteractorEventRecorder can also be used to play those events back and invoke them on an vtkRenderWindowInteractor. (Note: the events can also be played back from a file or string.)

The format of the event file is simple. It is: EventName X Y ctrl shift keycode repeatCount keySym The format also allows "#" comments.

To create an instance of class vtkInteractorEventRecorder, simply invoke its constructor as follows

```
obj = vtkInteractorEventRecorder
```

39.62.2 Methods

The class vtkInteractorEventRecorder has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorEventRecorder class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorEventRecorder = obj.NewInstance ()`
- `vtkInteractorEventRecorder = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )`
- `obj.SetInteractor (vtkRenderWindowInteractor iren)`
- `obj.SetFileName (string )` - Set/Get the name of a file events should be written to/from.
- `string = obj.GetFileName ()` - Set/Get the name of a file events should be written to/from.
- `obj.Record ()` - Invoke this method to begin recording events. The events will be recorded to the filename indicated.
- `obj.Play ()` - Invoke this method to begin playing events from the current position. The events will be played back from the filename indicated.
- `obj.Stop ()` - Invoke this method to stop recording/playing events.
- `obj.Rewind ()` - Rewind to the beginning of the file.
- `obj.SetReadFromInputString (int )` - Enable reading from an InputString as compared to the default behavior, which is to read from a file.
- `int = obj.GetReadFromInputString ()` - Enable reading from an InputString as compared to the default behavior, which is to read from a file.
- `obj.ReadFromInputStringOn ()` - Enable reading from an InputString as compared to the default behavior, which is to read from a file.
- `obj.ReadFromInputStringOff ()` - Enable reading from an InputString as compared to the default behavior, which is to read from a file.
- `obj.SetInputString (string )` - Set/Get the string to read from.
- `string = obj.GetInputString ()` - Set/Get the string to read from.

39.63 vtkInteractorObserver

39.63.1 Usage

vtkInteractorObserver is an abstract superclass for subclasses that observe events invoked by vtkRenderWindowInteractor. These subclasses are typically things like 3D widgets; objects that interact with actors in the scene, or interactively probe the scene for information.

vtkInteractorObserver defines the method SetInteractor() and enables and disables the processing of events by the vtkInteractorObserver. Use the methods EnabledOn() or SetEnabled(1) to turn on the interactor observer, and the methods EnabledOff() or SetEnabled(0) to turn off the interactor. Initial value is 0.

To support interactive manipulation of objects, this class (and subclasses) invoke the events StartInteractionEvent, InteractionEvent, and EndInteractionEvent. These events are invoked when the vtkInteractorObserver enters a state where rapid response is desired: mouse motion, etc. The events can be used, for example, to set the desired update frame rate (StartInteractionEvent), operate on data or update a pipeline (InteractionEvent), and set the desired frame rate back to normal values (EndInteractionEvent). Two other events, EnableEvent and DisableEvent, are invoked when the interactor observer is enabled or disabled.

To create an instance of class vtkInteractorObserver, simply invoke its constructor as follows

```
obj = vtkInteractorObserver
```

39.63.2 Methods

The class vtkInteractorObserver has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorObserver class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorObserver = obj.NewInstance ()`
- `vtkInteractorObserver = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods for turning the interactor observer on and off, and determining its state. All subclasses must provide the SetEnabled() method. Enabling a vtkInteractorObserver has the side effect of adding observers; disabling it removes the observers. Prior to enabling the vtkInteractorObserver you must set the render window interactor (via SetInteractor()). Initial value is 0.
- `int = obj.GetEnabled ()` - Methods for turning the interactor observer on and off, and determining its state. All subclasses must provide the SetEnabled() method. Enabling a vtkInteractorObserver has the side effect of adding observers; disabling it removes the observers. Prior to enabling the vtkInteractorObserver you must set the render window interactor (via SetInteractor()). Initial value is 0.
- `obj.EnabledOn ()` - Methods for turning the interactor observer on and off, and determining its state. All subclasses must provide the SetEnabled() method. Enabling a vtkInteractorObserver has the side effect of adding observers; disabling it removes the observers. Prior to enabling the vtkInteractorObserver you must set the render window interactor (via SetInteractor()). Initial value is 0.
- `obj.EnabledOff ()` - Methods for turning the interactor observer on and off, and determining its state. All subclasses must provide the SetEnabled() method. Enabling a vtkInteractorObserver has the side effect of adding observers; disabling it removes the observers. Prior to enabling the vtkInteractorObserver you must set the render window interactor (via SetInteractor()). Initial value is 0.

- `obj.On ()` - Methods for turning the interactor observer on and off, and determining its state. All subclasses must provide the SetEnabled() method. Enabling a vtkInteractorObserver has the side effect of adding observers; disabling it removes the observers. Prior to enabling the vtkInteractorObserver you must set the render window interactor (via SetInteractor()). Initial value is 0.

- `obj.Off ()` - This method is used to associate the widget with the render window interactor. Observers of the appropriate events invoked in the render window interactor are set up as a result of this method invocation. The SetInteractor() method must be invoked prior to enabling the vtkInteractorObserver.

- `obj.SetInteractor (vtkRenderWindowInteractor iren)` - This method is used to associate the widget with the render window interactor. Observers of the appropriate events invoked in the render window interactor are set up as a result of this method invocation. The SetInteractor() method must be invoked prior to enabling the vtkInteractorObserver.

- `vtkRenderWindowInteractor = obj.GetInteractor ()` - This method is used to associate the widget with the render window interactor. Observers of the appropriate events invoked in the render window interactor are set up as a result of this method invocation. The SetInteractor() method must be invoked prior to enabling the vtkInteractorObserver.

- `obj.SetPriority (float )` - Set/Get the priority at which events are processed. This is used when multiple interactor observers are used simultaneously. The default value is 0.0 (lowest priority.) Note that when multiple interactor observer have the same priority, then the last observer added will process the event first. (Note: once the SetInteractor() method has been called, changing the priority does not effect event processing. You will have to SetInteractor(NULL), change priority, and then SetInteractor(iren) to have the priority take effect.)

- `float = obj.GetPriorityMinValue ()` - Set/Get the priority at which events are processed. This is used when multiple interactor observers are used simultaneously. The default value is 0.0 (lowest priority.) Note that when multiple interactor observer have the same priority, then the last observer added will process the event first. (Note: once the SetInteractor() method has been called, changing the priority does not effect event processing. You will have to SetInteractor(NULL), change priority, and then SetInteractor(iren) to have the priority take effect.)

- `float = obj.GetPriorityMaxValue ()` - Set/Get the priority at which events are processed. This is used when multiple interactor observers are used simultaneously. The default value is 0.0 (lowest priority.) Note that when multiple interactor observer have the same priority, then the last observer added will process the event first. (Note: once the SetInteractor() method has been called, changing the priority does not effect event processing. You will have to SetInteractor(NULL), change priority, and then SetInteractor(iren) to have the priority take effect.)

- `float = obj.GetPriority ()` - Set/Get the priority at which events are processed. This is used when multiple interactor observers are used simultaneously. The default value is 0.0 (lowest priority.) Note that when multiple interactor observer have the same priority, then the last observer added will process the event first. (Note: once the SetInteractor() method has been called, changing the priority does not effect event processing. You will have to SetInteractor(NULL), change priority, and then SetInteractor(iren) to have the priority take effect.)

- `obj.SetKeyPressActivation (int )` - Enable/Disable of the use of a keypress to turn on and off the interactor observer. (By default, the keypress is 'i' for "interactor observer".) Set the KeyPressActivationValue to change which key activates the widget.)

- `int = obj.GetKeyPressActivation ()` - Enable/Disable of the use of a keypress to turn on and off the interactor observer. (By default, the keypress is 'i' for "interactor observer".) Set the KeyPressActivationValue to change which key activates the widget.)

- `obj.KeyPressActivationOn ()` - Enable/Disable of the use of a keypress to turn on and off the interactor observer. (By default, the keypress is 'i' for "interactor observer".) Set the KeyPressActivationValue to change which key activates the widget.)

- `obj.KeyPressActivationOff ()` - Enable/Disable of the use of a keypress to turn on and off the interactor observer. (By default, the keypress is 'i' for "interactor observer".) Set the KeyPressActivationValue to change which key activates the widget.)

- `obj.SetKeyPressActivationValue (char )` - Specify which key press value to use to activate the interactor observer (if key press activation is enabled). By default, the key press activation value is 'i'. Note: once the SetInteractor() method is invoked, changing the key press activation value will not affect the key press until SetInteractor(NULL)/SetInteractor(iren) is called.

- `char = obj.GetKeyPressActivationValue ()` - Specify which key press value to use to activate the interactor observer (if key press activation is enabled). By default, the key press activation value is 'i'. Note: once the SetInteractor() method is invoked, changing the key press activation value will not affect the key press until SetInteractor(NULL)/SetInteractor(iren) is called.

- `vtkRenderer = obj.GetDefaultRenderer ()` - Set/Get the default renderer to use when activating the interactor observer. Normally when the widget is activated (SetEnabled(1) or when keypress activation takes place), the renderer over which the mouse pointer is positioned is used. Alternatively, you can specify the renderer to bind the interactor to when the interactor observer is activated.

- `obj.SetDefaultRenderer (vtkRenderer )` - Set/Get the default renderer to use when activating the interactor observer. Normally when the widget is activated (SetEnabled(1) or when keypress activation takes place), the renderer over which the mouse pointer is positioned is used. Alternatively, you can specify the renderer to bind the interactor to when the interactor observer is activated.

- `vtkRenderer = obj.GetCurrentRenderer ()` - Set/Get the current renderer. Normally when the widget is activated (SetEnabled(1) or when keypress activation takes place), the renderer over which the mouse pointer is positioned is used and assigned to this Ivar. Alternatively, you might want to set the CurrentRenderer explicitly. WARNING: note that if the DefaultRenderer Ivar is set (see above), it will always override the parameter passed to SetCurrentRenderer, unless it is NULL. (i.e., SetCurrentRenderer(foo) = SetCurrentRenderer(DefaultRenderer).

- `obj.SetCurrentRenderer (vtkRenderer )` - Set/Get the current renderer. Normally when the widget is activated (SetEnabled(1) or when keypress activation takes place), the renderer over which the mouse pointer is positioned is used and assigned to this Ivar. Alternatively, you might want to set the CurrentRenderer explicitly. WARNING: note that if the DefaultRenderer Ivar is set (see above), it will always override the parameter passed to SetCurrentRenderer, unless it is NULL. (i.e., SetCurrentRenderer(foo) = SetCurrentRenderer(DefaultRenderer).

- `obj.OnChar ()` - Sets up the keypress-i event.

39.64 vtkInteractorStyle

39.64.1 Usage

vtkInteractorStyle is a base class implementing the majority of motion control routines and defines an event driven interface to support vtkRenderWindowInteractor. vtkRenderWindowInteractor implements platform dependent key/mouse routing and timer control, which forwards events in a neutral form to vtkInteractorStyle.

vtkInteractorStyle implements the "joystick" style of interaction. That is, holding down the mouse keys generates a stream of events that cause continuous actions (e.g., rotate, translate, pan, zoom). (The class vtkInteractorStyleTrackball implements a grab and move style.) The event bindings for this class include the following: - Keypress j / Keypress t: toggle between joystick (position sensitive) and trackball (motion sensitive) styles. In joystick style, motion occurs continuously as long as a mouse button is pressed. In trackball style, motion occurs when the mouse button is pressed and the mouse pointer moves. - Keypress c / Keypress a: toggle between camera and actor modes. In camera mode, mouse events affect the camera position and focal point. In actor mode, mouse events affect the actor that is under the mouse pointer. -

Button 1: rotate the camera around its focal point (if camera mode) or rotate the actor around its origin (if actor mode). The rotation is in the direction defined from the center of the renderer's viewport towards the mouse position. In joystick mode, the magnitude of the rotation is determined by the distance the mouse is from the center of the render window. - Button 2: pan the camera (if camera mode) or translate the actor (if actor mode). In joystick mode, the direction of pan or translation is from the center of the viewport towards the mouse position. In trackball mode, the direction of motion is the direction the mouse moves. (Note: with 2-button mice, pan is defined as Shift-Button 1.) - Button 3: zoom the camera (if camera mode) or scale the actor (if actor mode). Zoom in/increase scale if the mouse position is in the top half of the viewport; zoom out/decrease scale if the mouse position is in the bottom half. In joystick mode, the amount of zoom is controlled by the distance of the mouse pointer from the horizontal centerline of the window. - Keypress 3: toggle the render window into and out of stereo mode. By default, red-blue stereo pairs are created. Some systems support Crystal Eyes LCD stereo glasses; you have to invoke SetStereoTypeToCrystalEyes() on the rendering window. - Keypress e: exit the application. - Keypress f: fly to the picked point - Keypress p: perform a pick operation. The render window interactor has an internal instance of vtkCellPicker that it uses to pick. - Keypress r: reset the camera view along the current view direction. Centers the actors and moves the camera so that all actors are visible. - Keypress s: modify the representation of all actors so that they are surfaces. - Keypress u: invoke the user-defined function. Typically, this keypress will bring up an interactor that you can type commands in. Typing u calls UserCallBack() on the vtkRenderWindowInteractor, which invokes a vtkCommand::UserEvent. In other words, to define a user-defined callback, just add an observer to the vtkCommand::UserEvent on the vtkRenderWindowInteractor object. - Keypress w: modify the representation of all actors so that they are wireframe.

vtkInteractorStyle can be subclassed to provide new interaction styles and a facility to override any of the default mouse/key operations which currently handle trackball or joystick styles is provided. Note that this class will fire a variety of events that can be watched using an observer, such as LeftButtonPressEvent, LeftButtonReleaseEvent, MiddleButtonPressEvent, MiddleButtonReleaseEvent, RightButtonPressEvent, RightButtonReleaseEvent, EnterEvent, LeaveEvent, KeyPressEvent, KeyReleaseEvent, CharEvent, ExposeEvent, ConfigureEvent, TimerEvent, MouseMoveEvent,

To create an instance of class vtkInteractorStyle, simply invoke its constructor as follows

```
obj = vtkInteractorStyle
```

39.64.2 Methods

The class vtkInteractorStyle has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyle class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyle = obj.NewInstance ()`
- `vtkInteractorStyle = obj.SafeDownCast (vtkObject o)`
- `obj.SetInteractor (vtkRenderWindowInteractor interactor)` - Set/Get the Interactor wrapper being controlled by this object. (Satisfy superclass API.)
- `obj.SetEnabled (int )` - Turn on/off this interactor. Interactor styles operate a little bit differently than other types of interactor observers. When the SetInteractor() method is invoked, the automatically enable themselves. This is a legacy requirement, and convenient for the user.
- `obj.SetAutoAdjustCameraClippingRange (int )` - If AutoAdjustCameraClippingRange is on, then before each render the camera clipping range will be adjusted to "fit" the whole scene. Clipping will still occur if objects in the scene are behind the camera or come very close. If AutoAdjustCameraClippingRange is off, no adjustment will be made per render, but the camera clipping range will still be reset when the camera is reset.

- `int = obj.GetAutoAdjustCameraClippingRangeMinValue ()` - If AutoAdjustCameraClippingRange is on, then before each render the camera clipping range will be adjusted to "fit" the whole scene. Clipping will still occur if objects in the scene are behind the camera or come very close. If AutoAdjustCameraClippingRange is off, no adjustment will be made per render, but the camera clipping range will still be reset when the camera is reset.

- `int = obj.GetAutoAdjustCameraClippingRangeMaxValue ()` - If AutoAdjustCameraClippingRange is on, then before each render the camera clipping range will be adjusted to "fit" the whole scene. Clipping will still occur if objects in the scene are behind the camera or come very close. If AutoAdjustCameraClippingRange is off, no adjustment will be made per render, but the camera clipping range will still be reset when the camera is reset.

- `int = obj.GetAutoAdjustCameraClippingRange ()` - If AutoAdjustCameraClippingRange is on, then before each render the camera clipping range will be adjusted to "fit" the whole scene. Clipping will still occur if objects in the scene are behind the camera or come very close. If AutoAdjustCameraClippingRange is off, no adjustment will be made per render, but the camera clipping range will still be reset when the camera is reset.

- `obj.AutoAdjustCameraClippingRangeOn ()` - If AutoAdjustCameraClippingRange is on, then before each render the camera clipping range will be adjusted to "fit" the whole scene. Clipping will still occur if objects in the scene are behind the camera or come very close. If AutoAdjustCameraClippingRange is off, no adjustment will be made per render, but the camera clipping range will still be reset when the camera is reset.

- `obj.AutoAdjustCameraClippingRangeOff ()` - If AutoAdjustCameraClippingRange is on, then before each render the camera clipping range will be adjusted to "fit" the whole scene. Clipping will still occur if objects in the scene are behind the camera or come very close. If AutoAdjustCameraClippingRange is off, no adjustment will be made per render, but the camera clipping range will still be reset when the camera is reset.

- `obj.FindPokedRenderer (int , int )` - When an event occurs, we must determine which Renderer the event occurred within, since one RenderWindow may contain multiple renderers.

- `int = obj.GetState ()` - Some useful information for interaction

- `int = obj.GetUseTimers ()` - Set/Get timer hint

- `obj.SetUseTimers (int )` - Set/Get timer hint

- `obj.UseTimersOn ()` - Set/Get timer hint

- `obj.UseTimersOff ()` - Set/Get timer hint

- `obj.SetTimerDuration (long )` - If using timers, specify the default timer interval (in milliseconds). Care must be taken when adjusting the timer interval from the default value of 10 milliseconds–it may adversely affect the interactors.

- `GetTimerDurationMinValue = obj.()` - If using timers, specify the default timer interval (in milliseconds). Care must be taken when adjusting the timer interval from the default value of 10 milliseconds–it may adversely affect the interactors.

- `GetTimerDurationMaxValue = obj.()` - If using timers, specify the default timer interval (in milliseconds). Care must be taken when adjusting the timer interval from the default value of 10 milliseconds–it may adversely affect the interactors.

- `long = obj.GetTimerDuration ()` - If using timers, specify the default timer interval (in milliseconds). Care must be taken when adjusting the timer interval from the default value of 10 milliseconds–it may adversely affect the interactors.

- `obj.SetHandleObservers (int )` - Does ProcessEvents handle observers on this class or not
- `int = obj.GetHandleObservers ()` - Does ProcessEvents handle observers on this class or not
- `obj.HandleObserversOn ()` - Does ProcessEvents handle observers on this class or not
- `obj.HandleObserversOff ()` - Does ProcessEvents handle observers on this class or not
- `obj.OnMouseMove ()` - Generic event bindings can be overridden in subclasses
- `obj.OnLeftButtonDown ()` - Generic event bindings can be overridden in subclasses
- `obj.OnLeftButtonUp ()` - Generic event bindings can be overridden in subclasses
- `obj.OnMiddleButtonDown ()` - Generic event bindings can be overridden in subclasses
- `obj.OnMiddleButtonUp ()` - Generic event bindings can be overridden in subclasses
- `obj.OnRightButtonDown ()` - Generic event bindings can be overridden in subclasses
- `obj.OnRightButtonUp ()` - Generic event bindings can be overridden in subclasses
- `obj.OnMouseWheelForward ()` - Generic event bindings can be overridden in subclasses
- `obj.OnMouseWheelBackward ()` - Generic event bindings can be overridden in subclasses
- `obj.OnChar ()` - OnChar is triggered when an ASCII key is pressed. Some basic key presses are handled here ('q' for Quit, 'p' for Pick, etc)
- `obj.OnKeyDown ()`
- `obj.OnKeyUp ()`
- `obj.OnKeyPress ()`
- `obj.OnKeyRelease ()`
- `obj.OnExpose ()` - These are more esoteric events, but are useful in some cases.
- `obj.OnConfigure ()` - These are more esoteric events, but are useful in some cases.
- `obj.OnEnter ()` - These are more esoteric events, but are useful in some cases.
- `obj.OnLeave ()` - These are more esoteric events, but are useful in some cases.
- `obj.OnTimer ()` - OnTimer calls Rotate, Rotate etc which should be overridden by style subclasses.
- `obj.Rotate ()` - These methods for the different interactions in different modes are overridden in subclasses to perform the correct motion. Since they might be called from OnTimer, they do not have mouse coord parameters (use interactor's GetEventPosition and GetLastEventPosition)
- `obj.Spin ()` - These methods for the different interactions in different modes are overridden in subclasses to perform the correct motion. Since they might be called from OnTimer, they do not have mouse coord parameters (use interactor's GetEventPosition and GetLastEventPosition)
- `obj.Pan ()` - These methods for the different interactions in different modes are overridden in subclasses to perform the correct motion. Since they might be called from OnTimer, they do not have mouse coord parameters (use interactor's GetEventPosition and GetLastEventPosition)
- `obj.Dolly ()` - These methods for the different interactions in different modes are overridden in subclasses to perform the correct motion. Since they might be called from OnTimer, they do not have mouse coord parameters (use interactor's GetEventPosition and GetLastEventPosition)

- `obj.Zoom ()` - These methods for the different interactions in different modes are overridden in subclasses to perform the correct motion. Since they might be called from OnTimer, they do not have mouse coord parameters (use interactor's GetEventPosition and GetLastEventPosition)
- `obj.UniformScale ()` - These methods for the different interactions in different modes are overridden in subclasses to perform the correct motion. Since they might be called from OnTimer, they do not have mouse coord parameters (use interactor's GetEventPosition and GetLastEventPosition)
- `obj.StartState (int newstate)` - utility routines used by state changes
- `obj.StopState ()` - utility routines used by state changes
- `obj.StartAnimate ()` - Interaction mode entry points used internally.
- `obj.StopAnimate ()` - Interaction mode entry points used internally.
- `obj.StartRotate ()` - Interaction mode entry points used internally.
- `obj.EndRotate ()` - Interaction mode entry points used internally.
- `obj.StartZoom ()` - Interaction mode entry points used internally.
- `obj.EndZoom ()` - Interaction mode entry points used internally.
- `obj.StartPan ()` - Interaction mode entry points used internally.
- `obj.EndPan ()` - Interaction mode entry points used internally.
- `obj.StartSpin ()` - Interaction mode entry points used internally.
- `obj.EndSpin ()` - Interaction mode entry points used internally.
- `obj.StartDolly ()` - Interaction mode entry points used internally.
- `obj.EndDolly ()` - Interaction mode entry points used internally.
- `obj.StartUniformScale ()` - Interaction mode entry points used internally.
- `obj.EndUniformScale ()` - Interaction mode entry points used internally.
- `obj.StartTimer ()` - Interaction mode entry points used internally.
- `obj.EndTimer ()` - Interaction mode entry points used internally.
- `obj.HighlightProp (vtkProp prop)` - When picking successfully selects an actor, this method highlights the picked prop appropriately. Currently this is done by placing a bounding box around a picked vtkProp3D, and using the PickColor to highlight a vtkProp2D.
- `obj.HighlightActor2D (vtkActor2D actor2D)` - When picking successfully selects an actor, this method highlights the picked prop appropriately. Currently this is done by placing a bounding box around a picked vtkProp3D, and using the PickColor to highlight a vtkProp2D.
- `obj.HighlightProp3D (vtkProp3D prop3D)` - When picking successfully selects an actor, this method highlights the picked prop appropriately. Currently this is done by placing a bounding box around a picked vtkProp3D, and using the PickColor to highlight a vtkProp2D.
- `obj.SetPickColor (double , double , double )` - Set/Get the pick color (used by default to color vtkActor2D's). The color is expressed as red/green/blue values between (0.0,1.0).
- `obj.SetPickColor (double  a[3])` - Set/Get the pick color (used by default to color vtkActor2D's). The color is expressed as red/green/blue values between (0.0,1.0).

- `double = obj. GetPickColor ()` - Set/Get the pick color (used by default to color vtkActor2D's). The color is expressed as red/green/blue values between (0.0,1.0).
- `obj.SetMouseWheelMotionFactor (double )` - Set/Get the mouse wheel motion factor. Default to 1.0. Set it to a different value to emphasize or de-emphasize the action triggered by mouse wheel motion.
- `double = obj.GetMouseWheelMotionFactor ()` - Set/Get the mouse wheel motion factor. Default to 1.0. Set it to a different value to emphasize or de-emphasize the action triggered by mouse wheel motion.
- `vtkTDxInteractorStyle = obj.GetTDxStyle ()` - 3Dconnexion device interactor style. Initial value is a pointer to an object of class vtkTdxInteractorStyleCamera.
- `obj.SetTDxStyle (vtkTDxInteractorStyle tdxStyle)` - 3Dconnexion device interactor style. Initial value is a pointer to an object of class vtkTdxInteractorStyleCamera.

39.65 vtkInteractorStyleFlight

39.65.1 Usage

Left mouse button press produces forward motion. Right mouse button press produces reverse motion. Moving mouse during motion steers user in desired direction. Keyboard controls are: Left/Right/Up/Down Arrows for steering direction 'A' forward, 'Z' reverse motion Ctrl Key causes sidestep instead of steering in mouse and key modes Shift key is accelerator in mouse and key modes Ctrl and Shift together causes Roll in mouse and key modes

By default, one "step" of motion corresponds to 1/250th of the diagonal of bounding box of visible actors, '+' and '-' keys allow user to increase or decrease step size.

To create an instance of class vtkInteractorStyleFlight, simply invoke its constructor as follows

```
obj = vtkInteractorStyleFlight
```

39.65.2 Methods

The class vtkInteractorStyleFlight has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleFlight class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleFlight = obj.NewInstance ()`
- `vtkInteractorStyleFlight = obj.SafeDownCast (vtkObject o)`
- `obj.JumpTo (double campos[3], double focpos[3])` - Move the Eye/Camera to a specific location (no intermediate steps are taken
- `obj.SetMotionStepSize (double )` - Set the basic unit step size : by default 1/250 of bounding diagonal
- `double = obj.GetMotionStepSize ()` - Set the basic unit step size : by default 1/250 of bounding diagonal
- `obj.SetMotionAccelerationFactor (double )` - Set acceleration factor when shift key is applied : default 10

- `double = obj.GetMotionAccelerationFactor ()` - Set acceleration factor when shift key is applied : default 10
- `obj.SetAngleStepSize (double )` - Set the basic angular unit for turning : default 1 degree
- `double = obj.GetAngleStepSize ()` - Set the basic angular unit for turning : default 1 degree
- `obj.SetAngleAccelerationFactor (double )` - Set angular acceleration when shift key is applied : default 5
- `double = obj.GetAngleAccelerationFactor ()` - Set angular acceleration when shift key is applied : default 5
- `obj.SetDisableMotion (int )` - Disable motion (temporarily - for viewing etc)
- `int = obj.GetDisableMotion ()` - Disable motion (temporarily - for viewing etc)
- `obj.DisableMotionOn ()` - Disable motion (temporarily - for viewing etc)
- `obj.DisableMotionOff ()` - Disable motion (temporarily - for viewing etc)
- `obj.SetRestoreUpVector (int )` - When flying, apply a restorative force to the "Up" vector. This is activated when the current 'up' is close to the actual 'up' (as defined in DefaultUpVector). This prevents excessive twisting forces when viewing from arbitrary angles, but keep the horizon level when the user is flying over terrain.
- `int = obj.GetRestoreUpVector ()` - When flying, apply a restorative force to the "Up" vector. This is activated when the current 'up' is close to the actual 'up' (as defined in DefaultUpVector). This prevents excessive twisting forces when viewing from arbitrary angles, but keep the horizon level when the user is flying over terrain.
- `obj.RestoreUpVectorOn ()` - When flying, apply a restorative force to the "Up" vector. This is activated when the current 'up' is close to the actual 'up' (as defined in DefaultUpVector). This prevents excessive twisting forces when viewing from arbitrary angles, but keep the horizon level when the user is flying over terrain.
- `obj.RestoreUpVectorOff ()` - When flying, apply a restorative force to the "Up" vector. This is activated when the current 'up' is close to the actual 'up' (as defined in DefaultUpVector). This prevents excessive twisting forces when viewing from arbitrary angles, but keep the horizon level when the user is flying over terrain.
- `double = obj. GetDefaultUpVector ()`
- `obj.SetDefaultUpVector (double [3])`
- `obj.OnMouseMove ()` - Concrete implementation of Mouse event bindings for flight
- `obj.OnLeftButtonDown ()` - Concrete implementation of Mouse event bindings for flight
- `obj.OnLeftButtonUp ()` - Concrete implementation of Mouse event bindings for flight
- `obj.OnMiddleButtonDown ()` - Concrete implementation of Mouse event bindings for flight
- `obj.OnMiddleButtonUp ()` - Concrete implementation of Mouse event bindings for flight
- `obj.OnRightButtonDown ()` - Concrete implementation of Mouse event bindings for flight
- `obj.OnRightButtonUp ()` - Concrete implementation of Mouse event bindings for flight
- `obj.OnChar ()` - Concrete implementation of Keyboard event bindings for flight
- `obj.OnKeyDown ()` - Concrete implementation of Keyboard event bindings for flight

- `obj.OnKeyUp ()` - Concrete implementation of Keyboard event bindings for flight
- `obj.OnTimer ()` - Concrete implementation of Keyboard event bindings for flight
- `obj.ForwardFly ()` - Concrete implementation of Keyboard event bindings for flight
- `obj.ReverseFly ()` - Concrete implementation of Keyboard event bindings for flight
- `obj.StartForwardFly ()` - Concrete implementation of Keyboard event bindings for flight
- `obj.EndForwardFly ()` - Concrete implementation of Keyboard event bindings for flight
- `obj.StartReverseFly ()` - Concrete implementation of Keyboard event bindings for flight
- `obj.EndReverseFly ()` - Concrete implementation of Keyboard event bindings for flight

39.66 vtkInteractorStyleImage

39.66.1 Usage

vtkInteractorStyleImage allows the user to interactively manipulate (rotate, pan, zoomm etc.) the camera. vtkInteractorStyleImage is specially designed to work with images that are being rendered with vtkImage-Actor. Several events are overloaded from its superclass vtkInteractorStyle, hence the mouse bindings are different. (The bindings keep the camera's view plane normal perpendicular to the x-y plane.) In summary the mouse events are as follows: + Left Mouse button triggers window level events + CTRL Left Mouse spins the camera around its view plane normal + SHIFT Left Mouse pans the camera + CTRL SHIFT Left Mouse dollys (a positional zoom) the camera + Middle mouse button pans the camera + Right mouse button dollys the camera. + SHIFT Right Mouse triggers pick events

Note that the renderer's actors are not moved; instead the camera is moved.

To create an instance of class vtkInteractorStyleImage, simply invoke its constructor as follows

```
obj = vtkInteractorStyleImage
```

39.66.2 Methods

The class vtkInteractorStyleImage has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleImage class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleImage = obj.NewInstance ()`
- `vtkInteractorStyleImage = obj.SafeDownCast (vtkObject o)`
- `int = obj. GetWindowLevelStartPosition ()` - Some useful information for handling window level
- `int = obj. GetWindowLevelCurrentPosition ()` - Some useful information for handling window level
- `obj.OnMouseMove ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.

- `obj.OnLeftButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnChar ()` - Override the "fly-to" (f keypress) for images.
- `obj.WindowLevel ()`
- `obj.Pick ()`
- `obj.StartWindowLevel ()`
- `obj.EndWindowLevel ()`
- `obj.StartPick ()`
- `obj.EndPick ()`

39.67 vtkInteractorStyleJoystickActor

39.67.1 Usage

The class vtkInteractorStyleJoystickActor allows the user to interact with (rotate, zoom, etc.) separate objects in the scene independent of each other. The position of the mouse relative to the center of the object determines the speed of the object's motion. The mouse's velocity detemines the acceleration of the object's motion, so the object will continue moving even when the mouse is not moving. For a 3-button mouse, the left button is for rotation, the right button for zooming, the middle button for panning, and ctrl + left button for spinning. (With fewer mouse buttons, ctrl + shift + left button is for zooming, and shift + left button is for panning.)

To create an instance of class vtkInteractorStyleJoystickActor, simply invoke its constructor as follows

```
obj = vtkInteractorStyleJoystickActor
```

39.67.2 Methods

The class vtkInteractorStyleJoystickActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleJoystickActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleJoystickActor = obj.NewInstance ()`
- `vtkInteractorStyleJoystickActor = obj.SafeDownCast (vtkObject o)`
- `obj.OnMouseMove ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.

- `obj.OnLeftButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.Rotate ()`
- `obj.Spin ()`
- `obj.Pan ()`
- `obj.Dolly ()`
- `obj.UniformScale ()`

39.68 vtkInteractorStyleJoystickCamera

39.68.1 Usage

vtkInteractorStyleJoystickCamera allows the user to move (rotate, pan, etc.) the camera, the point of view for the scene. The position of the mouse relative to the center of the scene determines the speed at which the camera moves, and the speed of the mouse movement determines the acceleration of the camera, so the camera continues to move even if the mouse if not moving. For a 3-button mouse, the left button is for rotation, the right button for zooming, the middle button for panning, and ctrl + left button for spinning. (With fewer mouse buttons, ctrl + shift + left button is for zooming, and shift + left button is for panning.)

To create an instance of class vtkInteractorStyleJoystickCamera, simply invoke its constructor as follows

```
obj = vtkInteractorStyleJoystickCamera
```

39.68.2 Methods

The class vtkInteractorStyleJoystickCamera has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleJoystickCamera class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleJoystickCamera = obj.NewInstance ()`
- `vtkInteractorStyleJoystickCamera = obj.SafeDownCast (vtkObject o)`
- `obj.OnMouseMove ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.

- `obj.OnLeftButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMouseWheelForward ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMouseWheelBackward ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.Rotate ()`
- `obj.Spin ()`
- `obj.Pan ()`
- `obj.Dolly ()`

39.69 vtkInteractorStyleRubberBand2D

39.69.1 Usage

vtkInteractorStyleRubberBand2D manages interaction in a 2D view. Camera rotation is not allowed with this interactor style. The style also allows draws a rubber band using the left button. All camera changes invoke InteractionBeginEvent when the button is pressed, InteractionEvent when the mouse (or wheel) is moved, and InteractionEndEvent when the button is released. The bindings are as follows: Left mouse - Select (invokes a SelectionChangedEvent). Right mouse - Zoom. Middle mouse - Pan. Scroll wheel - Zoom.

To create an instance of class vtkInteractorStyleRubberBand2D, simply invoke its constructor as follows

```
obj = vtkInteractorStyleRubberBand2D
```

39.69.2 Methods

The class vtkInteractorStyleRubberBand2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleRubberBand2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleRubberBand2D = obj.NewInstance ()`
- `vtkInteractorStyleRubberBand2D = obj.SafeDownCast (vtkObject o)`
- `obj.OnLeftButtonDown ()`
- `obj.OnLeftButtonUp ()`

- `obj.OnMiddleButtonDown ()`
- `obj.OnMiddleButtonUp ()`
- `obj.OnRightButtonDown ()`
- `obj.OnRightButtonUp ()`
- `obj.OnMouseMove ()`
- `obj.OnMouseWheelForward ()`
- `obj.OnMouseWheelBackward ()`
- `obj.SetRenderOnMouseMove (bool )` - Whether to invoke a render when the mouse moves.
- `bool = obj.GetRenderOnMouseMove ()` - Whether to invoke a render when the mouse moves.
- `obj.RenderOnMouseMoveOn ()` - Whether to invoke a render when the mouse moves.
- `obj.RenderOnMouseMoveOff ()` - Whether to invoke a render when the mouse moves.
- `int = obj.GetInteraction ()` - Current interaction state

39.70 vtkInteractorStyleRubberBand3D

39.70.1 Usage

vtkInteractorStyleRubberBand3D manages interaction in a 3D view. The style also allows draws a rubber band using the left button. All camera changes invoke InteractionBeginEvent when the button is pressed, InteractionEvent when the mouse (or wheel) is moved, and InteractionEndEvent when the button is released. The bindings are as follows: Left mouse - Select (invokes a SelectionChangedEvent). Right mouse - Rotate. Shift + right mouse - Zoom. Middle mouse - Pan. Scroll wheel - Zoom.

To create an instance of class vtkInteractorStyleRubberBand3D, simply invoke its constructor as follows

```
obj = vtkInteractorStyleRubberBand3D
```

39.70.2 Methods

The class vtkInteractorStyleRubberBand3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleRubberBand3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleRubberBand3D = obj.NewInstance ()`
- `vtkInteractorStyleRubberBand3D = obj.SafeDownCast (vtkObject o)`
- `obj.OnLeftButtonDown ()`
- `obj.OnLeftButtonUp ()`
- `obj.OnMiddleButtonDown ()`
- `obj.OnMiddleButtonUp ()`
- `obj.OnRightButtonDown ()`

- `obj.OnRightButtonUp ()`
- `obj.OnMouseMove ()`
- `obj.OnMouseWheelForward ()`
- `obj.OnMouseWheelBackward ()`
- `obj.SetRenderOnMouseMove (bool )` - Whether to invoke a render when the mouse moves.
- `bool = obj.GetRenderOnMouseMove ()` - Whether to invoke a render when the mouse moves.
- `obj.RenderOnMouseMoveOn ()` - Whether to invoke a render when the mouse moves.
- `obj.RenderOnMouseMoveOff ()` - Whether to invoke a render when the mouse moves.
- `int = obj.GetInteraction ()` - Current interaction state

39.71 vtkInteractorStyleRubberBandPick

39.71.1 Usage

This interactor style allows the user to draw a rectangle in the render window by hitting 'r' and then using the left mouse button. When the mouse button is released, the attached picker operates on the pixel in the center of the selection rectangle. If the picker happens to be a vtkAreaPicker it will operate on the entire selection rectangle. When the 'p' key is hit the above pick operation occurs on a 1x1 rectangle. In other respects it behaves the same as its parent class.

To create an instance of class vtkInteractorStyleRubberBandPick, simply invoke its constructor as follows

```
obj = vtkInteractorStyleRubberBandPick
```

39.71.2 Methods

The class vtkInteractorStyleRubberBandPick has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleRubberBandPick class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleRubberBandPick = obj.NewInstance ()`
- `vtkInteractorStyleRubberBandPick = obj.SafeDownCast (vtkObject o)`
- `obj.StartSelect ()`
- `obj.OnMouseMove ()` - Event bindings
- `obj.OnLeftButtonDown ()` - Event bindings
- `obj.OnLeftButtonUp ()` - Event bindings
- `obj.OnChar ()` - Event bindings

39.72 vtkInteractorStyleRubberBandZoom

39.72.1 Usage

This interactor style allows the user to draw a rectangle in the render window using the left mouse button. When the mouse button is released, the current camera zooms by an amount determined from the shorter side of the drawn rectangle.

To create an instance of class vtkInteractorStyleRubberBandZoom, simply invoke its constructor as follows

```
obj = vtkInteractorStyleRubberBandZoom
```

39.72.2 Methods

The class vtkInteractorStyleRubberBandZoom has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleRubberBandZoom class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleRubberBandZoom = obj.NewInstance ()`
- `vtkInteractorStyleRubberBandZoom = obj.SafeDownCast (vtkObject o)`
- `obj.OnMouseMove ()` - Event bindings
- `obj.OnLeftButtonDown ()` - Event bindings
- `obj.OnLeftButtonUp ()` - Event bindings

39.73 vtkInteractorStyleSwitch

39.73.1 Usage

The class vtkInteractorStyleSwitch allows handles interactively switching between four interactor styles – joystick actor, joystick camera, trackball actor, and trackball camera. Type 'j' or 't' to select joystick or trackball, and type 'c' or 'a' to select camera or actor. The default interactor style is joystick camera.

To create an instance of class vtkInteractorStyleSwitch, simply invoke its constructor as follows

```
obj = vtkInteractorStyleSwitch
```

39.73.2 Methods

The class vtkInteractorStyleSwitch has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleSwitch class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleSwitch = obj.NewInstance ()`
- `vtkInteractorStyleSwitch = obj.SafeDownCast (vtkObject o)`
- `obj.SetInteractor (vtkRenderWindowInteractor iren)` - The sub styles need the interactor too.

- `obj.SetAutoAdjustCameraClippingRange (int value)` - We must override this method in order to pass the setting down to the underlying styles
- `vtkInteractorStyle = obj.GetCurrentStyle ()` - Set/Get current style
- `obj.SetCurrentStyleToJoystickActor ()` - Set/Get current style
- `obj.SetCurrentStyleToJoystickCamera ()` - Set/Get current style
- `obj.SetCurrentStyleToTrackballActor ()` - Set/Get current style
- `obj.SetCurrentStyleToTrackballCamera ()` - Set/Get current style
- `obj.OnChar ()` - Only care about the char event, which is used to switch between different styles.
- `obj.SetDefaultRenderer (vtkRenderer )` - Overridden from vtkInteractorObserver because the interactor styles used by this class must also be updated.
- `obj.SetCurrentRenderer (vtkRenderer )` - Overridden from vtkInteractorObserver because the interactor styles used by this class must also be updated.

39.74 vtkInteractorStyleTerrain

39.74.1 Usage

vtkInteractorStyleTerrain is used to manipulate a camera which is viewing a scene with a natural view up, e.g., terrain. The camera in such a scene is manipulated by specifying azimuth (angle around the view up vector) and elevation (the angle from the horizon).

The mouse binding for this class is as follows. Left mouse click followed rotates the camera around the focal point using both elevation and azimuth invocations on the camera. Left mouse motion in the horizontal direction results in azimuth motion; left mouse motion in the vertical direction results in elevation motion. Therefore, diagonal motion results in a combination of azimuth and elevation. (If the shift key is held during motion, then only one of elevation or azimuth is invoked, depending on the whether the mouse motion is primarily horizontal or vertical.) Middle mouse button pans the camera across the scene (again the shift key has a similar effect on limiting the motion to the vertical or horizontal direction. The right mouse is used to dolly (e.g., a type of zoom) towards or away from the focal point.

The class also supports some keypress events. The "r" key resets the camera. The "e" key invokes the exit callback and by default exits the program. The "f" key sets a new camera focal point and flys towards that point. The "u" key invokes the user event. The "3" key toggles between stereo and non-stero mode. The "l" key toggles on/off a latitude/longitude markers that can be used to estimate/control position.

To create an instance of class vtkInteractorStyleTerrain, simply invoke its constructor as follows

```
obj = vtkInteractorStyleTerrain
```

39.74.2 Methods

The class vtkInteractorStyleTerrain has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleTerrain class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleTerrain = obj.NewInstance ()`
- `vtkInteractorStyleTerrain = obj.SafeDownCast (vtkObject o)`

- `obj.OnMouseMove ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnChar ()` - Override the "fly-to" (f keypress) for images.
- `obj.Rotate ()`
- `obj.Pan ()`
- `obj.Dolly ()`
- `obj.SetLatLongLines (int )` - Turn on/off the latitude/longitude lines.
- `int = obj.GetLatLongLines ()` - Turn on/off the latitude/longitude lines.
- `obj.LatLongLinesOn ()` - Turn on/off the latitude/longitude lines.
- `obj.LatLongLinesOff ()` - Turn on/off the latitude/longitude lines.

39.75 vtkInteractorStyleTrackball

39.75.1 Usage

vtkInteractorStyleTrackball is an implementation of vtkInteractorStyle that defines the trackball style. It is now deprecated and as such a subclass of vtkInteractorStyleSwitch

To create an instance of class vtkInteractorStyleTrackball, simply invoke its constructor as follows

```
obj = vtkInteractorStyleTrackball
```

39.75.2 Methods

The class vtkInteractorStyleTrackball has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleTrackball class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleTrackball = obj.NewInstance ()`
- `vtkInteractorStyleTrackball = obj.SafeDownCast (vtkObject o)`

39.76 vtkInteractorStyleTrackballActor

39.76.1 Usage

vtkInteractorStyleTrackballActor allows the user to interact with (rotate, pan, etc.) objects in the scene indendent of each other. In trackball interaction, the magnitude of the mouse motion is proportional to the actor motion associated with a particular mouse binding. For example, small left-button motions cause small changes in the rotation of the actor around its center point.

The mouse bindings are as follows. For a 3-button mouse, the left button is for rotation, the right button for zooming, the middle button for panning, and ctrl + left button for spinning. (With fewer mouse buttons, ctrl + shift + left button is for zooming, and shift + left button is for panning.)

To create an instance of class vtkInteractorStyleTrackballActor, simply invoke its constructor as follows

```
obj = vtkInteractorStyleTrackballActor
```

39.76.2 Methods

The class vtkInteractorStyleTrackballActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleTrackballActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleTrackballActor = obj.NewInstance ()`
- `vtkInteractorStyleTrackballActor = obj.SafeDownCast (vtkObject o)`
- `obj.OnMouseMove ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.Rotate ()`
- `obj.Spin ()`
- `obj.Pan ()`
- `obj.Dolly ()`
- `obj.UniformScale ()`

39.77 vtkInteractorStyleTrackballCamera

39.77.1 Usage

vtkInteractorStyleTrackballCamera allows the user to interactively manipulate (rotate, pan, etc.) the camera, the viewpoint of the scene. In trackball interaction, the magnitude of the mouse motion is proportional to the camera motion associated with a particular mouse binding. For example, small left-button motions cause small changes in the rotation of the camera around its focal point. For a 3-button mouse, the left button is for rotation, the right button for zooming, the middle button for panning, and ctrl + left button for spinning. (With fewer mouse buttons, ctrl + shift + left button is for zooming, and shift + left button is for panning.)

To create an instance of class vtkInteractorStyleTrackballCamera, simply invoke its constructor as follows

```
obj = vtkInteractorStyleTrackballCamera
```

39.77.2 Methods

The class vtkInteractorStyleTrackballCamera has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleTrackballCamera class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleTrackballCamera = obj.NewInstance ()`
- `vtkInteractorStyleTrackballCamera = obj.SafeDownCast (vtkObject o)`
- `obj.OnMouseMove ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMouseWheelForward ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMouseWheelBackward ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.Rotate ()`
- `obj.Spin ()`

- `obj.Pan ()`
- `obj.Dolly ()`
- `obj.SetMotionFactor (double )` - Set the apparent sensitivity of the interactor style to mouse motion.
- `double = obj.GetMotionFactor ()` - Set the apparent sensitivity of the interactor style to mouse motion.

39.78 vtkInteractorStyleUnicam

39.78.1 Usage

UniCam is a camera interactor. Here, just the primary features of the UniCam technique are implemented. UniCam requires just one mouse button and supports context sensitive dollying, panning, and rotation. (In this implementation, it uses the right mouse button, leaving the middle and left available for other functions.) For more information, see the paper at:

ftp://ftp.cs.brown.edu/pub/papers/graphics/research/unicam.pdf

The following is a brief description of the UniCam Camera Controls. You can perform 3 operations on the camera: rotate, pan, and dolly the camera. All operations are reached through the right mouse button & mouse movements.

IMPORTANT: UniCam assumes there is an axis that makes sense as a "up" vector for the world. By default, this axis is defined to be the vector ¡0,0,1¿. You can set it explicitly for the data you are viewing with the 'SetWorldUpVector(..)' method in C++, or similarly in Tcl/Tk (or other interpreted languages).

1. ROTATE:

Position the cursor over the point you wish to rotate around and press and release the left mouse button. A 'focus dot' appears indicating the point that will be the center of rotation. To rotate, press and hold the left mouse button and drag the mouse.. release the button to complete the rotation.

Rotations can be done without placing a focus dot first by moving the mouse cursor to within 10left button followed by dragging the mouse. The last focus dot position will be re-used.

2. PAN:

Click and hold the left mouse button, and initially move the mouse left or right. The point under the initial pick will pick correlate w/ the mouse tip– (i.e., direct manipulation).

3. DOLLY (+ PAN):

Click and hold the left mouse button, and initially move the mouse up or down. Moving the mouse down will dolly towards the picked point, and moving the mouse up will dolly away from it. Dollying occurs relative to the picked point which simplifies the task of dollying towards a region of interest. Left and right mouse movements will pan the camera left and right.

To create an instance of class vtkInteractorStyleUnicam, simply invoke its constructor as follows

```
obj = vtkInteractorStyleUnicam
```

39.78.2 Methods

The class vtkInteractorStyleUnicam has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleUnicam class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleUnicam = obj.NewInstance ()`
- `vtkInteractorStyleUnicam = obj.SafeDownCast (vtkObject o)`

- `obj.SetWorldUpVector (double a[3])`
- `obj.SetWorldUpVector (double x, double y, double z)`
- `double = obj. GetWorldUpVector ()`
- `obj.OnMouseMove ()` - Concrete implementation of event bindings
- `obj.OnLeftButtonDown ()` - Concrete implementation of event bindings
- `obj.OnLeftButtonUp ()` - Concrete implementation of event bindings
- `obj.OnLeftButtonMove ()` - Concrete implementation of event bindings
- `obj.OnTimer ()` - OnTimer calls RotateCamera, RotateActor etc which should be overridden by style subclasses.

39.79 vtkInteractorStyleUser

39.79.1 Usage

The most common way to customize user interaction is to write a subclass of vtkInteractorStyle: vtkInteractorStyleUser allows you to customize the interaction to without subclassing vtkInteractorStyle. This is particularly useful for setting up custom interaction modes in scripting languages such as Tcl and Python. This class allows you to hook into the MouseMove, ButtonPress/Release, KeyPress/Release, etc. events. If you want to hook into just a single mouse button, but leave the interaction modes for the others unchanged, you must use e.g. SetMiddleButtonPressMethod() instead of the more general SetButtonPressMethod().

To create an instance of class vtkInteractorStyleUser, simply invoke its constructor as follows

```
obj = vtkInteractorStyleUser
```

39.79.2 Methods

The class vtkInteractorStyleUser has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleUser class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleUser = obj.NewInstance ()`
- `vtkInteractorStyleUser = obj.SafeDownCast (vtkObject o)`
- `int = obj. GetLastPos ()` - Get the most recent mouse position during mouse motion. In your user interaction method, you must use this to track the mouse movement. Do not use GetEventPosition(), which records the last position where a mouse button was pressed.
- `int = obj. GetOldPos ()` - Get the previous mouse position during mouse motion, or after a key press. This can be used to calculate the relative displacement of the mouse.
- `int = obj.GetShiftKey ()` - Test whether modifiers were held down when mouse button or key was pressed
- `int = obj.GetCtrlKey ()` - Test whether modifiers were held down when mouse button or key was pressed
- `int = obj.GetChar ()` - Get the character for a Char event.

- `string = obj.GetKeySym ()` - Get the KeySym (in the same format as Tk KeySyms) for a KeyPress or KeyRelease method.
- `int = obj.GetButton ()` - Get the mouse button that was last pressed inside the window (returns zero when the button is released).
- `obj.OnMouseMove ()` - Generic event bindings
- `obj.OnLeftButtonDown ()` - Generic event bindings
- `obj.OnLeftButtonUp ()` - Generic event bindings
- `obj.OnMiddleButtonDown ()` - Generic event bindings
- `obj.OnMiddleButtonUp ()` - Generic event bindings
- `obj.OnRightButtonDown ()` - Generic event bindings
- `obj.OnRightButtonUp ()` - Generic event bindings
- `obj.OnChar ()` - Keyboard functions
- `obj.OnKeyPress ()` - Keyboard functions
- `obj.OnKeyRelease ()` - Keyboard functions
- `obj.OnExpose ()` - These are more esoteric events, but are useful in some cases.
- `obj.OnConfigure ()` - These are more esoteric events, but are useful in some cases.
- `obj.OnEnter ()` - These are more esoteric events, but are useful in some cases.
- `obj.OnLeave ()` - These are more esoteric events, but are useful in some cases.
- `obj.OnTimer ()`

39.80 vtkIVExporter

39.80.1 Usage

vtkIVExporter is a concrete subclass of vtkExporter that writes OpenInventor 2.0 files.

To create an instance of class vtkIVExporter, simply invoke its constructor as follows

```
obj = vtkIVExporter
```

39.80.2 Methods

The class vtkIVExporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIVExporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIVExporter = obj.NewInstance ()`
- `vtkIVExporter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify the name of the OpenInventor file to write.
- `string = obj.GetFileName ()` - Specify the name of the OpenInventor file to write.

39.81 vtkLabeledDataMapper

39.81.1 Usage

vtkLabeledDataMapper is a mapper that renders text at dataset points. Various items can be labeled including point ids, scalars, vectors, normals, texture coordinates, tensors, and field data components.

The format with which the label is drawn is specified using a printf style format string. The font attributes of the text can be set through the vtkTextProperty associated to this mapper.

By default, all the components of multi-component data such as vectors, normals, texture coordinates, tensors, and multi-component scalars are labeled. However, you can specify a single component if you prefer. (Note: the label format specifies the format to use for a single component. The label is creating by looping over all components and using the label format to render each component.)

To create an instance of class vtkLabeledDataMapper, simply invoke its constructor as follows

```
obj = vtkLabeledDataMapper
```

39.81.2 Methods

The class vtkLabeledDataMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabeledDataMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabeledDataMapper = obj.NewInstance ()`
- `vtkLabeledDataMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetLabelFormat (string )` - Set/Get the format with which to print the labels. This should be a printf-style format string.

  By default, the mapper will try to print each component of the tuple using a sane format: doubles, format, set it here. You can do things like limit the number of significant digits, add prefixes/suffixes, basically anything that printf can do. If you only want to print one component of a vector, see the ivar LabeledComponent.

- `string = obj.GetLabelFormat ()` - Set/Get the format with which to print the labels. This should be a printf-style format string.

  By default, the mapper will try to print each component of the tuple using a sane format: doubles, format, set it here. You can do things like limit the number of significant digits, add prefixes/suffixes, basically anything that printf can do. If you only want to print one component of a vector, see the ivar LabeledComponent.

- `obj.SetLabeledComponent (int )` - Set/Get the component number to label if the data to print has more than one component. For example, all the components of scalars, vectors, normals, etc. are labeled by default (LabeledComponent=(-1)). However, if this ivar is nonnegative, then only the one component specified is labeled.

- `int = obj.GetLabeledComponent ()` - Set/Get the component number to label if the data to print has more than one component. For example, all the components of scalars, vectors, normals, etc. are labeled by default (LabeledComponent=(-1)). However, if this ivar is nonnegative, then only the one component specified is labeled.

- `obj.SetFieldDataArray (int arrayIndex)` - Set/Get the field data array to label. This instance variable is only applicable if field data is labeled. This will clear FieldDataName when set.

- `int = obj.GetFieldDataArray ()` - Set/Get the field data array to label. This instance variable is only applicable if field data is labeled. This will clear FieldDataName when set.
- `obj.SetFieldDataName (string arrayName)` - Set/Get the name of the field data array to label. This instance variable is only applicable if field data is labeled. This will override FieldDataArray when set.
- `string = obj.GetFieldDataName ()` - Set/Get the name of the field data array to label. This instance variable is only applicable if field data is labeled. This will override FieldDataArray when set.
- `obj.SetInput (vtkDataObject )` - Set the input dataset to the mapper. This mapper handles any type of data.
- `vtkDataSet = obj.GetInput ()` - Use GetInputDataObject() to get the input data object for composite datasets.
- `obj.SetLabelMode (int )` - Specify which data to plot: IDs, scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetLabeledComponent to control which components to plot. The default is VTK_LABEL_IDS.
- `int = obj.GetLabelMode ()` - Specify which data to plot: IDs, scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetLabeledComponent to control which components to plot. The default is VTK_LABEL_IDS.
- `obj.SetLabelModeToLabelIds ()` - Specify which data to plot: IDs, scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetLabeledComponent to control which components to plot. The default is VTK_LABEL_IDS.
- `obj.SetLabelModeToLabelScalars ()` - Specify which data to plot: IDs, scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetLabeledComponent to control which components to plot. The default is VTK_LABEL_IDS.
- `obj.SetLabelModeToLabelVectors ()` - Specify which data to plot: IDs, scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetLabeledComponent to control which components to plot. The default is VTK_LABEL_IDS.
- `obj.SetLabelModeToLabelNormals ()` - Specify which data to plot: IDs, scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetLabeledComponent to control which components to plot. The default is VTK_LABEL_IDS.
- `obj.SetLabelModeToLabelTCoords ()` - Specify which data to plot: IDs, scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetLabeledComponent to control which components to plot. The default is VTK_LABEL_IDS.
- `obj.SetLabelModeToLabelTensors ()` - Specify which data to plot: IDs, scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetLabeledComponent to control which components to plot. The default is VTK_LABEL_IDS.
- `obj.SetLabelModeToLabelFieldData ()` - Specify which data to plot: IDs, scalars, vectors, normals, texture coords, tensors, or field data. If the data has more than one component, use the method SetLabeledComponent to control which components to plot. The default is VTK_LABEL_IDS.
- `obj.SetLabelTextProperty (vtkTextProperty p)` - Set/Get the text property. If an integer argument is provided, you may provide different text properties for different label types. The type is determined by an optional type input array.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - Set/Get the text property. If an integer argument is provided, you may provide different text properties for different label types. The type is determined by an optional type input array.

- `obj.SetLabelTextProperty (vtkTextProperty p, int type)` - Set/Get the text property. If an integer argument is provided, you may provide different text properties for different label types. The type is determined by an optional type input array.

- `vtkTextProperty = obj.GetLabelTextProperty (int type)` - Set/Get the text property. If an integer argument is provided, you may provide different text properties for different label types. The type is determined by an optional type input array.

- `obj.RenderOpaqueGeometry (vtkViewport viewport, vtkActor2D actor)` - Draw the text to the screen at each input point.

- `obj.RenderOverlay (vtkViewport viewport, vtkActor2D actor)` - Draw the text to the screen at each input point.

- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor.

- `vtkTransform = obj.GetTransform ()` - The transform to apply to the labels before mapping to 2D.

- `obj.SetTransform (vtkTransform t)` - The transform to apply to the labels before mapping to 2D.

- `int = obj.GetCoordinateSystem ()` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.

- `obj.SetCoordinateSystem (int )` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.

- `int = obj.GetCoordinateSystemMinValue ()` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.

- `int = obj.GetCoordinateSystemMaxValue ()` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.

- `obj.CoordinateSystemWorld ()` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.

- `obj.CoordinateSystemDisplay ()` - Return the modified time for this object.

- `long = obj.GetMTime ()` - Return the modified time for this object.

39.82 vtkLabeledTreeMapDataMapper

39.82.1 Usage

vtkLabeledTreeMapDataMapper is a mapper that renders text on a tree map. A tree map is a vtkTree with an associated 4-tuple array used for storing the boundary rectangle for each vertex in the tree. The user must specify the array name used for storing the rectangles.

The mapper iterates through the tree and attempts and renders a label inside the vertex's rectangle as long as the following conditions hold: 1. The vertex level is within the range of levels specified for labeling. 2. The label can fully fit inside its box. 3. The label does not overlap an ancestor's label.

To create an instance of class vtkLabeledTreeMapDataMapper, simply invoke its constructor as follows

```
obj = vtkLabeledTreeMapDataMapper
```

39.82.2 Methods

The class vtkLabeledTreeMapDataMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabeledTreeMapDataMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabeledTreeMapDataMapper = obj.NewInstance ()`
- `vtkLabeledTreeMapDataMapper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderOpaqueGeometry (vtkViewport viewport, vtkActor2D actor)` - Draw the text to the screen at each input point.
- `obj.RenderOverlay (vtkViewport viewport, vtkActor2D actor)` - Draw the text to the screen at each input point.
- `vtkTree = obj.GetInputTree ()` - The input to this filter.
- `obj.SetRectanglesArrayName (string name)` - The name of the 4-tuple array used for
- `int = obj.GetClipTextMode ()` - Indicates if the label can be displayed clipped by the Window mode = 0 - ok to clip labels 1 - auto center labels w/r to the area of the vertex's clipped region
- `obj.SetClipTextMode (int )` - Indicates if the label can be displayed clipped by the Window mode = 0 - ok to clip labels 1 - auto center labels w/r to the area of the vertex's clipped region
- `int = obj.GetChildMotion ()` - Indicates if the label can be moved by its ancestors
- `obj.SetChildMotion (int )` - Indicates if the label can be moved by its ancestors
- `int = obj.GetDynamicLevel ()` - Indicates at which level labeling should be dynamic
- `obj.SetDynamicLevel (int )` - Indicates at which level labeling should be dynamic
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor.
- `obj.SetFontSizeRange (int maxSize, int minSize, int delta)` - The range of font sizes to use when rendering the labels.
- `obj.GetFontSizeRange (int range[3])` - The range of font sizes to use when rendering the labels.
- `obj.SetLevelRange (int startLevel, int endLevel)` - The range of levels to attempt to label. The level of a vertex is the length of the path to the root (the root has level 0).
- `obj.GetLevelRange (int range[2])` - The range of levels to attempt to label. The level of a vertex is the length of the path to the root (the root has level 0).

39.83 vtkLabelHierarchy

39.83.1 Usage

This class represents labels in a hierarchy used to denote rendering priority. A binary tree of labels is maintained that subdivides the bounds of the of the label anchors spatially. Which level of the tree a label occupies determines its priority; those at higher levels of the tree will be more likely to render than those at lower levels of the tree.

Pass vtkLabelHierarchy objects to a vtkLabelPlacementMapper filter for dynamic, non-overlapping, per-frame placement of labels.

Note that if we have a d-dimensional binary tree and we want a fixed number n of labels in each node (all nodes, not just leaves), we can compute the depth of tree required assuming a uniform distribution of points. Given a total of N points we know that $\frac{N}{|T|} = n$, where $|T|$ is the cardinality of the tree (i.e., the number of nodes it contains). Because we have a uniform distribution, the tree will be uniformly subdivided and thus $|T| = 1 + 2^d + \left(2^d\right)^2 + \cdots + \left(2^d\right)^k$, where d is the dimensionality of the input points (fixed at 3 for now). As k becomes large, $|T|pprox2\left(2^d\right)^k$. Using this approximation, we can solve for k:

$$k = \frac{\log \frac{N}{2n}}{\log 2^d}$$

Given a set of N input label anchors, we'll compute k and then bin the anchors into tree nodes at level k of the tree. After this, all the nodes will be in the leaves of the tree and those leaves will be at the k-th level; no anchors will be in levels $1, 2, \ldots, k-1$. To fix that, we'll choose to move some anchors upwards. The exact number to move upwards depends on TargetLabelCount. We'll move as many up as required to have TargetLabelCount at each node.

You should avoid situations where MaximumDepth does not allow for TargetLabelCount or fewer entries at each node. The MaximumDepth is a hard limit while TargetLabelCount is a suggested optimum. You will end up with many more than TargetLabelCount entries per node and things will be sloooow.

To create an instance of class vtkLabelHierarchy, simply invoke its constructor as follows

```
obj = vtkLabelHierarchy
```

39.83.2 Methods

The class vtkLabelHierarchy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabelHierarchy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabelHierarchy = obj.NewInstance ()`
- `vtkLabelHierarchy = obj.SafeDownCast (vtkObject o)`
- `obj.SetPoints (vtkPoints )` - Override SetPoints so we can reset the hierarchy when the points change.
- `obj.ComputeHierarchy ()` - Fill the hierarchy with the input labels.
- `obj.SetTargetLabelCount (int )` - The number of labels that is ideally present at any octree node. It is best if this is a multiple of 2^d.
- `int = obj.GetTargetLabelCount ()` - The number of labels that is ideally present at any octree node. It is best if this is a multiple of 2^d.

- `obj.SetMaximumDepth (int )` - The maximum depth of the octree.
- `int = obj.GetMaximumDepth ()` - The maximum depth of the octree.
- `obj.SetTextProperty (vtkTextProperty tprop)` - The default text property assigned to labels in this hierarchy.
- `vtkTextProperty = obj.GetTextProperty ()` - The default text property assigned to labels in this hierarchy.
- `obj.SetPriorities (vtkDataArray arr)` - Set/get the array specifying the importance (priority) of each label.
- `vtkDataArray = obj.GetPriorities ()` - Set/get the array specifying the importance (priority) of each label.
- `obj.SetLabels (vtkAbstractArray arr)` - Set/get the array specifying the text of each label.
- `vtkAbstractArray = obj.GetLabels ()` - Set/get the array specifying the text of each label.
- `obj.SetOrientations (vtkDataArray arr)` - Set/get the array specifying the orientation of each label.
- `vtkDataArray = obj.GetOrientations ()` - Set/get the array specifying the orientation of each label.
- `obj.SetIconIndices (vtkIntArray arr)` - Set/get the array specifying the icon index of each label.
- `vtkIntArray = obj.GetIconIndices ()` - Set/get the array specifying the icon index of each label.
- `obj.SetSizes (vtkDataArray arr)` - Set/get the array specifying the size of each label.
- `vtkDataArray = obj.GetSizes ()` - Set/get the array specifying the size of each label.
- `obj.SetBoundedSizes (vtkDataArray arr)` - Set/get the array specifying the maximum width and height in world coordinates of each label.
- `vtkDataArray = obj.GetBoundedSizes ()` - Set/get the array specifying the maximum width and height in world coordinates of each label.
- `obj.GetDiscreteNodeCoordinatesFromWorldPoint (int ijk[3], double pt[3], int level)` - Given a depth in the hierarchy (level) and a point pt in world space, compute ijk. This is used to find other octree nodes at the same level that are within the search radius for candidate labels to be placed. It is called with pt set to the camera eye point and pythagorean quadruples increasingly distant from the origin are added to ijk to identify octree nodes whose labels should be placed. @param[out] ijk - discrete coordinates of the octree node at level containing pt. @param[in] pt - input world point coordinates @param[in] level - input octree level to be considered
- `vtkIdType = obj.GetNumberOfCells ()` - Inherited members (from vtkDataSet)
- `vtkCell = obj.GetCell (vtkIdType )` - Inherited members (from vtkDataSet)
- `obj.GetCell (vtkIdType , vtkGenericCell )` - Inherited members (from vtkDataSet)
- `int = obj.GetCellType (vtkIdType )` - Inherited members (from vtkDataSet)
- `obj.GetCellPoints (vtkIdType , vtkIdList )` - Inherited members (from vtkDataSet)
- `obj.GetPointCells (vtkIdType , vtkIdList )` - Inherited members (from vtkDataSet)
- `int = obj.GetMaxCellSize ()` - Inherited members (from vtkDataSet)
- `vtkPoints = obj.GetCenterPts ()` - Provide access to original coordinates of sets of coincident points

- `vtkCoincidentPoints = obj.GetCoincidentPoints ()` - Provide access to the set of coincident points that have been perturbed by the hierarchy in order to render labels for each without overlap.

39.84 vtkLabelHierarchyAlgorithm

39.84.1 Usage

vtkLabelHierarchyAlgorithm is a convenience class to make writing algorithms easier. It is also designed to help transition old algorithms to the new pipeline architecture. There are some assumptions and defaults made by this class you should be aware of. This class defaults such that your filter will have one input port and one output port. If that is not the case simply change it with SetNumberOfInputPorts etc. See this class constructor for the default. This class also provides a FillInputPortInfo method that by default says that all inputs will be DataObjects. If that isn't the case then please override this method in your subclass. This class breaks out the downstream requests into separate functions such as RequestData and RequestInformation. You should implement RequestData(request, inputVec, outputVec) in subclasses.

To create an instance of class vtkLabelHierarchyAlgorithm, simply invoke its constructor as follows

```
obj = vtkLabelHierarchyAlgorithm
```

39.84.2 Methods

The class vtkLabelHierarchyAlgorithm has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabelHierarchyAlgorithm class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabelHierarchyAlgorithm = obj.NewInstance ()`
- `vtkLabelHierarchyAlgorithm = obj.SafeDownCast (vtkObject o)`
- `vtkLabelHierarchy = obj.GetOutput ()` - Get the output data object for a port on this algorithm.
- `vtkLabelHierarchy = obj.GetOutput (int )` - Get the output data object for a port on this algorithm.
- `obj.SetOutput (vtkDataObject d)` - Get the output data object for a port on this algorithm.
- `vtkDataObject = obj.GetInput ()`
- `vtkDataObject = obj.GetInput (int port)`
- `vtkLabelHierarchy = obj.GetLabelHierarchyInput (int port)`
- `obj.SetInput (vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.
- `obj.SetInput (int , vtkDataObject )` - Set an input of this algorithm. You should not override these methods because they are not the only way to connect a pipeline. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::SetInputConnection(). These methods transform the input index to the input port index, not an index of a connection within a single port.

- `obj.AddInput (vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.
- `obj.AddInput (int , vtkDataObject )` - Add an input of this algorithm. Note that these methods support old-style pipeline connections. When writing new code you should use the more general vtkAlgorithm::AddInputConnection(). See SetInput() for details.

39.85 vtkLabelHierarchyCompositeIterator

39.85.1 Usage

Iterates over child iterators in a round-robin order. Each iterator may have its own count, which is the number of times it is repeated until moving to the next iterator.

For example, if you initialize the iterator with

```
it->AddIterator(A, 1);
it->AddIterator(B, 3);
```

The order of iterators will be A,B,B,B,A,B,B,B,...

To create an instance of class vtkLabelHierarchyCompositeIterator, simply invoke its constructor as follows

```
obj = vtkLabelHierarchyCompositeIterator
```

39.85.2 Methods

The class vtkLabelHierarchyCompositeIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabelHierarchyCompositeIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabelHierarchyCompositeIterator = obj.NewInstance ()`
- `vtkLabelHierarchyCompositeIterator = obj.SafeDownCast (vtkObject o)`
- `obj.AddIterator (vtkLabelHierarchyIterator it)` - Adds a label iterator to this composite iterator. The second optional argument is the number of times to repeat the iterator before moving to the next one round-robin style. Default is 1.
- `obj.AddIterator (vtkLabelHierarchyIterator it, int count)` - Adds a label iterator to this composite iterator. The second optional argument is the number of times to repeat the iterator before moving to the next one round-robin style. Default is 1.
- `obj.ClearIterators ()` - Remove all iterators from this composite iterator.
- `obj.Begin (vtkIdTypeArray )` - Initializes the iterator. lastLabels is an array holding labels which should be traversed before any other labels in the hierarchy. This could include labels placed during a previous rendering or a label located under the mouse pointer. You may pass a null pointer.
- `obj.Next ()` - Advance the iterator.
- `bool = obj.IsAtEnd ()` - Returns true if the iterator is at the end.

- `vtkIdType = obj.GetLabelId ()` - Retrieves the current label id.
- `vtkLabelHierarchy = obj.GetHierarchy ()` - Retrieve the current label hierarchy.
- `obj.BoxNode ()` - Not implemented.
- `obj.BoxAllNodes (vtkPolyData )`

39.86 vtkLabelHierarchyIterator

39.86.1 Usage

Abstract superclass for iterators over vtkLabelHierarchy.

To create an instance of class vtkLabelHierarchyIterator, simply invoke its constructor as follows

```
obj = vtkLabelHierarchyIterator
```

39.86.2 Methods

The class vtkLabelHierarchyIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabelHierarchyIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabelHierarchyIterator = obj.NewInstance ()`
- `vtkLabelHierarchyIterator = obj.SafeDownCast (vtkObject o)`
- `obj.Begin (vtkIdTypeArray )` - Advance the iterator.
- `obj.Next ()` - Returns true if the iterator is at the end.
- `bool = obj.IsAtEnd ()` - Retrieves the current label location.
- `obj.GetPoint (double x[3])` - Retrieves the current label location.
- `obj.GetSize (double sz[2])` - Retrieves the current label size.
- `obj.GetBoundedSize (double sz[2])` - Retrieves the current label maximum width in world coordinates.
- `int = obj.GetType ()` - Retrieves the current label type.
- `double = obj.GetOrientation ()` - Retrieves the current label orientation.
- `vtkIdType = obj.GetLabelId ()` - Get the label hierarchy associated with the current label.
- `vtkLabelHierarchy = obj.GetHierarchy ()` - Get the label hierarchy associated with the current label.
- `obj.SetTraversedBounds (vtkPolyData )` - Sets a polydata to fill with geometry representing the bounding boxes of the traversed octree nodes.
- `obj.BoxNode ()` - Add a representation to TraversedBounds for the current octree node. This should be called by subclasses inside Next(). Does nothing if TraversedBounds is NULL.

- `obj.BoxAllNodes (vtkPolyData )` - Add a representation for all existing octree nodes to the specified polydata. This is equivalent to setting TraversedBounds, iterating over the entire hierarchy, and then resetting TraversedBounds to its original value.
- `obj.SetAllBounds (int )` - Set/get whether all nodes in the hierarchy should be added to the TraversedBounds polydata or only those traversed. When non-zero, all nodes will be added. By default, AllBounds is 0.
- `int = obj.GetAllBounds ()` - Set/get whether all nodes in the hierarchy should be added to the TraversedBounds polydata or only those traversed. When non-zero, all nodes will be added. By default, AllBounds is 0.

39.87 vtkLabelPlacementMapper

39.87.1 Usage

To use this mapper, first send your data through vtkPointSetToLabelHierarchy, which takes a set of points, associates special arrays to the points (label, priority, etc.), and produces a prioritized spatial tree of labels.

This mapper then takes that hierarchy (or hierarchies) as input, and every frame will decide which labels and/or icons to place in order of priority, and will render only those labels/icons. A label render strategy is used to render the labels, and can use e.g. FreeType or Qt for rendering.

To create an instance of class vtkLabelPlacementMapper, simply invoke its constructor as follows

```
obj = vtkLabelPlacementMapper
```

39.87.2 Methods

The class vtkLabelPlacementMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabelPlacementMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabelPlacementMapper = obj.NewInstance ()`
- `vtkLabelPlacementMapper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderOverlay (vtkViewport viewport, vtkActor2D actor)` - Draw non-overlapping labels to the screen.
- `obj.SetRenderStrategy (vtkLabelRenderStrategy s)` - Set the label rendering strategy.
- `vtkLabelRenderStrategy = obj.GetRenderStrategy ()` - Set the label rendering strategy.
- `obj.SetMaximumLabelFraction (double )` - The maximum fraction of the screen that the labels may cover. Label placement stops when this fraction is reached.
- `double = obj.GetMaximumLabelFractionMinValue ()` - The maximum fraction of the screen that the labels may cover. Label placement stops when this fraction is reached.
- `double = obj.GetMaximumLabelFractionMaxValue ()` - The maximum fraction of the screen that the labels may cover. Label placement stops when this fraction is reached.
- `double = obj.GetMaximumLabelFraction ()` - The maximum fraction of the screen that the labels may cover. Label placement stops when this fraction is reached.

- `obj.SetIteratorType (int )` - The type of iterator used when traversing the labels. May be vtkLabelHierarchy::FRUSTUM or vtkLabelHierarchy::FULL_SORT
- `int = obj.GetIteratorType ()` - The type of iterator used when traversing the labels. May be vtkLabelHierarchy::FRUSTUM or vtkLabelHierarchy::FULL_SORT
- `obj.SetUseUnicodeStrings (bool )` - Set whether, or not, to use unicode strings.
- `bool = obj.GetUseUnicodeStrings ()` - Set whether, or not, to use unicode strings.
- `obj.UseUnicodeStringsOn ()` - Set whether, or not, to use unicode strings.
- `obj.UseUnicodeStringsOff ()` - Set whether, or not, to use unicode strings.
- `bool = obj.GetPositionsAsNormals ()` - Use label anchor point coordinates as normal vectors and eliminate those pointing away from the camera. Valid only when points are on a sphere centered at the origin (such as a 3D geographic view). Off by default.
- `obj.SetPositionsAsNormals (bool )` - Use label anchor point coordinates as normal vectors and eliminate those pointing away from the camera. Valid only when points are on a sphere centered at the origin (such as a 3D geographic view). Off by default.
- `obj.PositionsAsNormalsOn ()` - Use label anchor point coordinates as normal vectors and eliminate those pointing away from the camera. Valid only when points are on a sphere centered at the origin (such as a 3D geographic view). Off by default.
- `obj.PositionsAsNormalsOff ()` - Use label anchor point coordinates as normal vectors and eliminate those pointing away from the camera. Valid only when points are on a sphere centered at the origin (such as a 3D geographic view). Off by default.
- `bool = obj.GetGeneratePerturbedLabelSpokes ()` - Enable drawing spokes (lines) to anchor point coordinates that were perturbed for being coincident with other anchor point coordinates.
- `obj.SetGeneratePerturbedLabelSpokes (bool )` - Enable drawing spokes (lines) to anchor point coordinates that were perturbed for being coincident with other anchor point coordinates.
- `obj.GeneratePerturbedLabelSpokesOn ()` - Enable drawing spokes (lines) to anchor point coordinates that were perturbed for being coincident with other anchor point coordinates.
- `obj.GeneratePerturbedLabelSpokesOff ()` - Enable drawing spokes (lines) to anchor point coordinates that were perturbed for being coincident with other anchor point coordinates.
- `bool = obj.GetUseDepthBuffer ()` - Use the depth buffer to test each label to see if it should not be displayed if it would be occluded by other objects in the scene. Off by default.
- `obj.SetUseDepthBuffer (bool )` - Use the depth buffer to test each label to see if it should not be displayed if it would be occluded by other objects in the scene. Off by default.
- `obj.UseDepthBufferOn ()` - Use the depth buffer to test each label to see if it should not be displayed if it would be occluded by other objects in the scene. Off by default.
- `obj.UseDepthBufferOff ()` - Use the depth buffer to test each label to see if it should not be displayed if it would be occluded by other objects in the scene. Off by default.
- `obj.SetPlaceAllLabels (bool )` - Tells the placer to place every label regardless of overlap. Off by default.
- `bool = obj.GetPlaceAllLabels ()` - Tells the placer to place every label regardless of overlap. Off by default.
- `obj.PlaceAllLabelsOn ()` - Tells the placer to place every label regardless of overlap. Off by default.

- `obj.PlaceAllLabelsOff ()` - Tells the placer to place every label regardless of overlap. Off by default.
- `obj.SetOutputTraversedBounds (bool )` - Whether to render traversed bounds. Off by default.
- `bool = obj.GetOutputTraversedBounds ()` - Whether to render traversed bounds. Off by default.
- `obj.OutputTraversedBoundsOn ()` - Whether to render traversed bounds. Off by default.
- `obj.OutputTraversedBoundsOff ()` - Whether to render traversed bounds. Off by default.
- `obj.SetShape (int )` - The shape of the label background, should be one of the values in the LabelShape enumeration.
- `int = obj.GetShapeMinValue ()` - The shape of the label background, should be one of the values in the LabelShape enumeration.
- `int = obj.GetShapeMaxValue ()` - The shape of the label background, should be one of the values in the LabelShape enumeration.
- `int = obj.GetShape ()` - The shape of the label background, should be one of the values in the LabelShape enumeration.
- `obj.SetShapeToNone ()` - The shape of the label background, should be one of the values in the LabelShape enumeration.
- `obj.SetShapeToRect ()` - The shape of the label background, should be one of the values in the LabelShape enumeration.
- `obj.SetShapeToRoundedRect ()` - The style of the label background shape, should be one of the values in the LabelStyle enumeration.
- `obj.SetStyle (int )` - The style of the label background shape, should be one of the values in the LabelStyle enumeration.
- `int = obj.GetStyleMinValue ()` - The style of the label background shape, should be one of the values in the LabelStyle enumeration.
- `int = obj.GetStyleMaxValue ()` - The style of the label background shape, should be one of the values in the LabelStyle enumeration.
- `int = obj.GetStyle ()` - The style of the label background shape, should be one of the values in the LabelStyle enumeration.
- `obj.SetStyleToFilled ()` - The style of the label background shape, should be one of the values in the LabelStyle enumeration.
- `obj.SetStyleToOutline ()` - The size of the margin on the label background shape. Default is 5.
- `obj.SetMargin (double )` - The size of the margin on the label background shape. Default is 5.
- `double = obj.GetMargin ()` - The size of the margin on the label background shape. Default is 5.
- `obj.SetBackgroundColor (double , double , double )` - The color of the background shape.
- `obj.SetBackgroundColor (double  a[3])` - The color of the background shape.
- `double = obj. GetBackgroundColor ()` - The color of the background shape.
- `obj.SetBackgroundOpacity (double )` - The opacity of the background shape.
- `double = obj.GetBackgroundOpacityMinValue ()` - The opacity of the background shape.
- `double = obj.GetBackgroundOpacityMaxValue ()` - The opacity of the background shape.
- `double = obj.GetBackgroundOpacity ()` - The opacity of the background shape.
- `vtkCoordinate = obj.GetAnchorTransform ()` - Get the transform for the anchor points.

39.88 vtkLabelPlacer

39.88.1 Usage

¡b¿This class is deprecated and will be removed from VTK in a future release. Use vtkLabelPlacementMapper instead.¡/b¿

This should probably be a mapper unto itself (given that the polydata output could be large and will realistically always be iterated over exactly once before being tossed for the next frame of the render).

In any event, it takes as input one (or more, eventually) vtkLabelHierarchies that represent prioritized lists of labels sorted by their placement in space. As output, it provides vtkPolyData containing only VTK_QUAD cells, each representing a single label from the input. Each quadrilateral has cell data indicating what label in the input it corresponds to (via an array named "LabelId").

To create an instance of class vtkLabelPlacer, simply invoke its constructor as follows

```
obj = vtkLabelPlacer
```

39.88.2 Methods

The class vtkLabelPlacer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabelPlacer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabelPlacer = obj.NewInstance ()`
- `vtkLabelPlacer = obj.SafeDownCast (vtkObject o)`
- `vtkRenderer = obj.GetRenderer ()`
- `obj.SetRenderer (vtkRenderer )`
- `vtkCoordinate = obj.GetAnchorTransform ()`
- `obj.SetGravity (int gravity)` - The placement of the label relative to the anchor point.
- `int = obj.GetGravity ()` - The placement of the label relative to the anchor point.
- `obj.SetMaximumLabelFraction (double )` - The maximum amount of screen space labels can take up before placement terminates.
- `double = obj.GetMaximumLabelFractionMinValue ()` - The maximum amount of screen space labels can take up before placement terminates.
- `double = obj.GetMaximumLabelFractionMaxValue ()` - The maximum amount of screen space labels can take up before placement terminates.
- `double = obj.GetMaximumLabelFraction ()` - The maximum amount of screen space labels can take up before placement terminates.
- `obj.SetIteratorType (int )` - The type of iterator used when traversing the labels. May be vtkLabelHierarchy::FRUSTUM or vtkLabelHierarchy::FULL_SORT.
- `int = obj.GetIteratorType ()` - The type of iterator used when traversing the labels. May be vtkLabelHierarchy::FRUSTUM or vtkLabelHierarchy::FULL_SORT.
- `obj.SetUseUnicodeStrings (bool )` - Set whether, or not, to use unicode strings.

- `bool = obj.GetUseUnicodeStrings ()` - Set whether, or not, to use unicode strings.
- `obj.UseUnicodeStringsOn ()` - Set whether, or not, to use unicode strings.
- `obj.UseUnicodeStringsOff ()` - Set whether, or not, to use unicode strings.
- `long = obj.GetMTime ()`
- `bool = obj.GetPositionsAsNormals ()` - Use label anchor point coordinates as normal vectors and eliminate those pointing away from the camera. Valid only when points are on a sphere centered at the origin (such as a 3D geographic view). Off by default.
- `obj.SetPositionsAsNormals (bool )` - Use label anchor point coordinates as normal vectors and eliminate those pointing away from the camera. Valid only when points are on a sphere centered at the origin (such as a 3D geographic view). Off by default.
- `obj.PositionsAsNormalsOn ()` - Use label anchor point coordinates as normal vectors and eliminate those pointing away from the camera. Valid only when points are on a sphere centered at the origin (such as a 3D geographic view). Off by default.
- `obj.PositionsAsNormalsOff ()` - Use label anchor point coordinates as normal vectors and eliminate those pointing away from the camera. Valid only when points are on a sphere centered at the origin (such as a 3D geographic view). Off by default.
- `bool = obj.GetGeneratePerturbedLabelSpokes ()` - Enable drawing spokes (lines) to anchor point coordinates that were perturbed for being coincident with other anchor point coordinates.
- `obj.SetGeneratePerturbedLabelSpokes (bool )` - Enable drawing spokes (lines) to anchor point coordinates that were perturbed for being coincident with other anchor point coordinates.
- `obj.GeneratePerturbedLabelSpokesOn ()` - Enable drawing spokes (lines) to anchor point coordinates that were perturbed for being coincident with other anchor point coordinates.
- `obj.GeneratePerturbedLabelSpokesOff ()` - Enable drawing spokes (lines) to anchor point coordinates that were perturbed for being coincident with other anchor point coordinates.
- `bool = obj.GetUseDepthBuffer ()` - Use the depth buffer to test each label to see if it should not be displayed if it would be occluded by other objects in the scene. Off by default.
- `obj.SetUseDepthBuffer (bool )` - Use the depth buffer to test each label to see if it should not be displayed if it would be occluded by other objects in the scene. Off by default.
- `obj.UseDepthBufferOn ()` - Use the depth buffer to test each label to see if it should not be displayed if it would be occluded by other objects in the scene. Off by default.
- `obj.UseDepthBufferOff ()` - Use the depth buffer to test each label to see if it should not be displayed if it would be occluded by other objects in the scene. Off by default.
- `bool = obj.GetOutputTraversedBounds ()` - In the second output, output the geometry of the traversed octree nodes.
- `obj.SetOutputTraversedBounds (bool )` - In the second output, output the geometry of the traversed octree nodes.
- `obj.OutputTraversedBoundsOn ()` - In the second output, output the geometry of the traversed octree nodes.
- `obj.OutputTraversedBoundsOff ()` - In the second output, output the geometry of the traversed octree nodes.
- `int = obj.GetOutputCoordinateSystem ()` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.

- `obj.SetOutputCoordinateSystem (int )` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.
- `int = obj.GetOutputCoordinateSystemMinValue ()` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.
- `int = obj.GetOutputCoordinateSystemMaxValue ()` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.
- `obj.OutputCoordinateSystemWorld ()` - Set/get the coordinate system used for output labels. The output datasets may have point coordinates reported in the world space or display space.
- `obj.OutputCoordinateSystemDisplay ()`

39.89 vtkLabelRenderStrategy

39.89.1 Usage

These methods should only be called within a mapper.

To create an instance of class vtkLabelRenderStrategy, simply invoke its constructor as follows

```
obj = vtkLabelRenderStrategy
```

39.89.2 Methods

The class vtkLabelRenderStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabelRenderStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabelRenderStrategy = obj.NewInstance ()`
- `vtkLabelRenderStrategy = obj.SafeDownCast (vtkObject o)`
- `bool = obj.SupportsRotation ()` - Whether the text rendering strategy supports bounded size. The superclass returns true. Subclasses should override this to return the appropriate value. Subclasses that return true from this method should implement the version of RenderLabel() that takes a maximum size (see RenderLabel()).
- `bool = obj.SupportsBoundedSize ()` - Set the renderer associated with this strategy.
- `obj.SetRenderer (vtkRenderer ren)` - Set the renderer associated with this strategy.
- `vtkRenderer = obj.GetRenderer ()` - Set the renderer associated with this strategy.
- `obj.SetDefaultTextProperty (vtkTextProperty tprop)` - Set the default text property for the strategy.
- `vtkTextProperty = obj.GetDefaultTextProperty ()` - Set the default text property for the strategy.
- `obj.StartFrame ()` - End a rendering frame.
- `obj.EndFrame ()` - Release any graphics resources that are being consumed by this strategy. The parameter window could be used to determine which graphic resources to release.
- `obj.ReleaseGraphicsResources (vtkWindow )`

39.90 vtkLabelSizeCalculator

39.90.1 Usage

This filter takes an input dataset, an array to process (which must be a string array), and a text property. It creates a new output array (named "LabelSize" by default) with 4 components per tuple that contain the width, height, horizontal offset, and descender height (in that order) of each string in the array.

Use the inherited SelectInputArrayToProcess to indicate a string array. In no input array is specified, the first of the following that is a string array is used: point scalars, cell scalars, field scalars.

The second input array to process is an array specifying the type of each label. Different label types may have different font properties. This array must be a vtkIntArray. Any type that does not map to a font property that was set will be set to the type 0's type property.

To create an instance of class vtkLabelSizeCalculator, simply invoke its constructor as follows

```
obj = vtkLabelSizeCalculator
```

39.90.2 Methods

The class vtkLabelSizeCalculator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLabelSizeCalculator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLabelSizeCalculator = obj.NewInstance ()`
- `vtkLabelSizeCalculator = obj.SafeDownCast (vtkObject o)`
- `obj.SetFontProperty (vtkTextProperty fontProp, int type)` - Get/Set the font used compute label sizes. This defaults to "Arial" at 12 points. If type is provided, it refers to the type of the text label provided in the optional label type array. The default type is type 0.
- `vtkTextProperty = obj.GetFontProperty (int type)` - Get/Set the font used compute label sizes. This defaults to "Arial" at 12 points. If type is provided, it refers to the type of the text label provided in the optional label type array. The default type is type 0.
- `obj.SetLabelSizeArrayName (string )` - The name of the output array containing text label sizes This defaults to "LabelSize"
- `string = obj.GetLabelSizeArrayName ()` - The name of the output array containing text label sizes This defaults to "LabelSize"

39.91 vtkLeaderActor2D

39.91.1 Usage

vtkLeaderActor2D creates a leader with an optional label and arrows. (A leader is typically used to indicate distance between points.) vtkLeaderActor2D is a type of vtkActor2D; that is, it is drawn on the overlay plane and is not occluded by 3D geometry. To use this class, you typically specify two points defining the start and end points of the line (x-y definition using vtkCoordinate class), whether to place arrows on one or both end points, and whether to label the leader. Also, this class has a special feature that allows curved leaders to be created by specifying a radius.

Use the vtkLeaderActor2D uses its superclass vtkActor2D instance variables Position and Position2 vtkCoordinates to place an instance of vtkLeaderActor2D (i.e., these two data members represent the start

and end points of the leader). Using these vtkCoordinates you can specify the position of the leader in a variety of coordinate systems.

To control the appearance of the actor, use the superclasses vtkActor2D::vtkProperty2D and the vtkTextProperty objects associated with this actor.

To create an instance of class vtkLeaderActor2D, simply invoke its constructor as follows

```
obj = vtkLeaderActor2D
```

39.91.2 Methods

The class vtkLeaderActor2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLeaderActor2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLeaderActor2D = obj.NewInstance ()`
- `vtkLeaderActor2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetRadius (double )` - Set/Get a radius which can be used to curve the leader. If a radius is specified whose absolute value is greater than one half the distance between the two points defined by the superclasses' Position and Position2 ivars, then the leader will be curved. A positive radius will produce a curve such that the center is to the right of the line from Position and Position2; a negative radius will produce a curve in the opposite sense. By default, the radius is set to zero and thus there is no curvature. Note that the radius is expresses as a multiple of the distance between (Position,Position2); this avoids issues relative to coordinate system transformations.
- `double = obj.GetRadius ()` - Set/Get a radius which can be used to curve the leader. If a radius is specified whose absolute value is greater than one half the distance between the two points defined by the superclasses' Position and Position2 ivars, then the leader will be curved. A positive radius will produce a curve such that the center is to the right of the line from Position and Position2; a negative radius will produce a curve in the opposite sense. By default, the radius is set to zero and thus there is no curvature. Note that the radius is expresses as a multiple of the distance between (Position,Position2); this avoids issues relative to coordinate system transformations.
- `obj.SetLabel (string )` - Set/Get the label for the leader. If the label is an empty string, then it will not be drawn.
- `string = obj.GetLabel ()` - Set/Get the label for the leader. If the label is an empty string, then it will not be drawn.
- `obj.SetLabelTextProperty (vtkTextProperty p)` - Set/Get the text property of the label.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - Set/Get the text property of the label.
- `obj.SetLabelFactor (double )` - Set/Get the factor that controls the overall size of the fonts used to label the leader.
- `double = obj.GetLabelFactorMinValue ()` - Set/Get the factor that controls the overall size of the fonts used to label the leader.
- `double = obj.GetLabelFactorMaxValue ()` - Set/Get the factor that controls the overall size of the fonts used to label the leader.
- `double = obj.GetLabelFactor ()` - Set/Get the factor that controls the overall size of the fonts used to label the leader.

- `obj.SetArrowPlacement (int )` - Control whether arrow heads are drawn on the leader. Arrows may be drawn on one end, both ends, or not at all.
- `int = obj.GetArrowPlacementMinValue ()` - Control whether arrow heads are drawn on the leader. Arrows may be drawn on one end, both ends, or not at all.
- `int = obj.GetArrowPlacementMaxValue ()` - Control whether arrow heads are drawn on the leader. Arrows may be drawn on one end, both ends, or not at all.
- `int = obj.GetArrowPlacement ()` - Control whether arrow heads are drawn on the leader. Arrows may be drawn on one end, both ends, or not at all.
- `obj.SetArrowPlacementToNone ()` - Control whether arrow heads are drawn on the leader. Arrows may be drawn on one end, both ends, or not at all.
- `obj.SetArrowPlacementToPoint1 ()` - Control whether arrow heads are drawn on the leader. Arrows may be drawn on one end, both ends, or not at all.
- `obj.SetArrowPlacementToPoint2 ()` - Control whether arrow heads are drawn on the leader. Arrows may be drawn on one end, both ends, or not at all.
- `obj.SetArrowPlacementToBoth ()` - Control the appearance of the arrow heads. A solid arrow head is a filled triangle; a open arrow looks like a "V"; and a hollow arrow looks like a non-filled triangle.
- `obj.SetArrowStyle (int )` - Control the appearance of the arrow heads. A solid arrow head is a filled triangle; a open arrow looks like a "V"; and a hollow arrow looks like a non-filled triangle.
- `int = obj.GetArrowStyleMinValue ()` - Control the appearance of the arrow heads. A solid arrow head is a filled triangle; a open arrow looks like a "V"; and a hollow arrow looks like a non-filled triangle.
- `int = obj.GetArrowStyleMaxValue ()` - Control the appearance of the arrow heads. A solid arrow head is a filled triangle; a open arrow looks like a "V"; and a hollow arrow looks like a non-filled triangle.
- `int = obj.GetArrowStyle ()` - Control the appearance of the arrow heads. A solid arrow head is a filled triangle; a open arrow looks like a "V"; and a hollow arrow looks like a non-filled triangle.
- `obj.SetArrowStyleToFilled ()` - Control the appearance of the arrow heads. A solid arrow head is a filled triangle; a open arrow looks like a "V"; and a hollow arrow looks like a non-filled triangle.
- `obj.SetArrowStyleToOpen ()` - Control the appearance of the arrow heads. A solid arrow head is a filled triangle; a open arrow looks like a "V"; and a hollow arrow looks like a non-filled triangle.
- `obj.SetArrowStyleToHollow ()` - Specify the arrow length and base width (in normalized viewport coordinates).
- `obj.SetArrowLength (double )` - Specify the arrow length and base width (in normalized viewport coordinates).
- `double = obj.GetArrowLengthMinValue ()` - Specify the arrow length and base width (in normalized viewport coordinates).
- `double = obj.GetArrowLengthMaxValue ()` - Specify the arrow length and base width (in normalized viewport coordinates).
- `double = obj.GetArrowLength ()` - Specify the arrow length and base width (in normalized viewport coordinates).
- `obj.SetArrowWidth (double )` - Specify the arrow length and base width (in normalized viewport coordinates).

- `double = obj.GetArrowWidthMinValue ()` - Specify the arrow length and base width (in normalized viewport coordinates).
- `double = obj.GetArrowWidthMaxValue ()` - Specify the arrow length and base width (in normalized viewport coordinates).
- `double = obj.GetArrowWidth ()` - Specify the arrow length and base width (in normalized viewport coordinates).
- `obj.SetMinimumArrowSize (double )` - Limit the minimum and maximum size of the arrows. These values are expressed in pixels and clamp the minimum/maximum possible size for the width/length of the arrow head. (When clamped, the ratio between length and width is preserved.)
- `double = obj.GetMinimumArrowSizeMinValue ()` - Limit the minimum and maximum size of the arrows. These values are expressed in pixels and clamp the minimum/maximum possible size for the width/length of the arrow head. (When clamped, the ratio between length and width is preserved.)
- `double = obj.GetMinimumArrowSizeMaxValue ()` - Limit the minimum and maximum size of the arrows. These values are expressed in pixels and clamp the minimum/maximum possible size for the width/length of the arrow head. (When clamped, the ratio between length and width is preserved.)
- `double = obj.GetMinimumArrowSize ()` - Limit the minimum and maximum size of the arrows. These values are expressed in pixels and clamp the minimum/maximum possible size for the width/length of the arrow head. (When clamped, the ratio between length and width is preserved.)
- `obj.SetMaximumArrowSize (double )` - Limit the minimum and maximum size of the arrows. These values are expressed in pixels and clamp the minimum/maximum possible size for the width/length of the arrow head. (When clamped, the ratio between length and width is preserved.)
- `double = obj.GetMaximumArrowSizeMinValue ()` - Limit the minimum and maximum size of the arrows. These values are expressed in pixels and clamp the minimum/maximum possible size for the width/length of the arrow head. (When clamped, the ratio between length and width is preserved.)
- `double = obj.GetMaximumArrowSizeMaxValue ()` - Limit the minimum and maximum size of the arrows. These values are expressed in pixels and clamp the minimum/maximum possible size for the width/length of the arrow head. (When clamped, the ratio between length and width is preserved.)
- `double = obj.GetMaximumArrowSize ()` - Limit the minimum and maximum size of the arrows. These values are expressed in pixels and clamp the minimum/maximum possible size for the width/length of the arrow head. (When clamped, the ratio between length and width is preserved.)
- `obj.SetAutoLabel (int )` - Enable auto-labelling. In this mode, the label is automatically updated based on distance (in world coordinates) between the two end points; or if a curved leader is being generated, the angle in degrees between the two points.
- `int = obj.GetAutoLabel ()` - Enable auto-labelling. In this mode, the label is automatically updated based on distance (in world coordinates) between the two end points; or if a curved leader is being generated, the angle in degrees between the two points.
- `obj.AutoLabelOn ()` - Enable auto-labelling. In this mode, the label is automatically updated based on distance (in world coordinates) between the two end points; or if a curved leader is being generated, the angle in degrees between the two points.
- `obj.AutoLabelOff ()` - Enable auto-labelling. In this mode, the label is automatically updated based on distance (in world coordinates) between the two end points; or if a curved leader is being generated, the angle in degrees between the two points.
- `obj.SetLabelFormat (string )` - Specify the format to use for auto-labelling.
- `string = obj.GetLabelFormat ()` - Specify the format to use for auto-labelling.

- `double = obj.GetLength ()` - Obtain the length of the leader if the leader is not curved, otherwise obtain the angle that the leader circumscribes.
- `double = obj.GetAngle ()` - Obtain the length of the leader if the leader is not curved, otherwise obtain the angle that the leader circumscribes.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods required by vtkProp and vtkActor2D superclasses.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods required by vtkProp and vtkActor2D superclasses.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Does this prop have some translucent polygonal geometry?
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ReleaseGraphicsResources (vtkWindow )`
- `obj.ShallowCopy (vtkProp prop)`

39.92 vtkLight

39.92.1 Usage

vtkLight is a virtual light for 3D rendering. It provides methods to locate and point the light, turn it on and off, and set its brightness and color. In addition to the basic infinite distance point light source attributes, you also can specify the light attenuation values and cone angle. These attributes are only used if the light is a positional light. The default is a directional light (e.g. infinite point light source).

Lights have a type that describes how the light should move with respect to the camera. A Headlight is always located at the current camera position and shines on the camera's focal point. A CameraLight also moves with the camera, but may not be coincident to it. CameraLights are defined in a normalized coordinate space where the camera is located at (0, 0, 1), the camera is looking at (0, 0, 0), and up is (0, 1, 0). Finally, a SceneLight is part of the scene itself and does not move with the camera. (Renderers are responsible for moving the light based on its type.)

Lights have a transformation matrix that describes the space in which they are positioned. A light's world space position and focal point are defined by their local position and focal point, transformed by their transformation matrix (if it exists).

To create an instance of class vtkLight, simply invoke its constructor as follows

```
obj = vtkLight
```

39.92.2 Methods

The class vtkLight has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLight class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLight = obj.NewInstance ()`
- `vtkLight = obj.SafeDownCast (vtkObject o)`

- `vtkLight = obj.ShallowClone ()` - Create a new light object with the same light parameters than the current object (any ivar from the superclasses (vtkObject and vtkObjectBase), like reference counting, timestamp and observers are not copied). This is a shallow clone (TransformMatrix is referenced)

- `obj.Render (vtkRenderer , int )` - Abstract interface to renderer. Each concrete subclass of vtkLight will load its data into the graphics system in response to this method invocation. The actual loading is performed by a vtkLightDevice subclass, which will get created automatically.

- `obj.SetAmbientColor (double , double , double )` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `obj.SetAmbientColor (double a[3])` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `double = obj. GetAmbientColor ()` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `obj.SetDiffuseColor (double , double , double )` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `obj.SetDiffuseColor (double a[3])` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `double = obj. GetDiffuseColor ()` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `obj.SetSpecularColor (double , double , double )` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `obj.SetSpecularColor (double a[3])` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `double = obj. GetSpecularColor ()` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `obj.SetColor (double , double , double )` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `obj.SetColor (double a[3])` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `obj.GetColor (double rgb[3])` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `double = obj.GetColor ()` - Set/Get the color of the light. It is possible to set the ambient, diffuse and specular colors separately. The SetColor() method sets the diffuse and specular colors to the same color (this is a feature to preserve backward compatbility.)

- `obj.SetPosition (double , double , double )` - Set/Get the position of the light. Note: The position of the light is defined in the coordinate space indicated by its transformation matrix (if it exists). Thus, to get the light's world space position, use vtkGetTransformedPosition() instead of vtkGetPosition().
- `obj.SetPosition (double a[3])` - Set/Get the position of the light. Note: The position of the light is defined in the coordinate space indicated by its transformation matrix (if it exists). Thus, to get the light's world space position, use vtkGetTransformedPosition() instead of vtkGetPosition().
- `double = obj. GetPosition ()` - Set/Get the position of the light. Note: The position of the light is defined in the coordinate space indicated by its transformation matrix (if it exists). Thus, to get the light's world space position, use vtkGetTransformedPosition() instead of vtkGetPosition().
- `obj.SetPosition (float a)` - Set/Get the position of the light. Note: The position of the light is defined in the coordinate space indicated by its transformation matrix (if it exists). Thus, to get the light's world space position, use vtkGetTransformedPosition() instead of vtkGetPosition().
- `obj.SetFocalPoint (double , double , double )` - Set/Get the point at which the light is shining. Note: The focal point of the light is defined in the coordinate space indicated by its transformation matrix (if it exists). Thus, to get the light's world space focal point, use vtkGetTransformedFocalPoint() instead of vtkGetFocalPoint().
- `obj.SetFocalPoint (double a[3])` - Set/Get the point at which the light is shining. Note: The focal point of the light is defined in the coordinate space indicated by its transformation matrix (if it exists). Thus, to get the light's world space focal point, use vtkGetTransformedFocalPoint() instead of vtkGetFocalPoint().
- `double = obj. GetFocalPoint ()` - Set/Get the point at which the light is shining. Note: The focal point of the light is defined in the coordinate space indicated by its transformation matrix (if it exists). Thus, to get the light's world space focal point, use vtkGetTransformedFocalPoint() instead of vtkGetFocalPoint().
- `obj.SetFocalPoint (float a)` - Set/Get the point at which the light is shining. Note: The focal point of the light is defined in the coordinate space indicated by its transformation matrix (if it exists). Thus, to get the light's world space focal point, use vtkGetTransformedFocalPoint() instead of vtkGetFocalPoint().
- `obj.SetIntensity (double )` - Set/Get the brightness of the light (from one to zero).
- `double = obj.GetIntensity ()` - Set/Get the brightness of the light (from one to zero).
- `obj.SetSwitch (int )` - Turn the light on or off.
- `int = obj.GetSwitch ()` - Turn the light on or off.
- `obj.SwitchOn ()` - Turn the light on or off.
- `obj.SwitchOff ()` - Turn the light on or off.
- `obj.SetPositional (int )` - Turn positional lighting on or off.
- `int = obj.GetPositional ()` - Turn positional lighting on or off.
- `obj.PositionalOn ()` - Turn positional lighting on or off.
- `obj.PositionalOff ()` - Turn positional lighting on or off.
- `obj.SetExponent (double )` - Set/Get the exponent of the cosine used in positional lighting.
- `double = obj.GetExponentMinValue ()` - Set/Get the exponent of the cosine used in positional lighting.

- `double = obj.GetExponentMaxValue ()` - Set/Get the exponent of the cosine used in positional lighting.
- `double = obj.GetExponent ()` - Set/Get the exponent of the cosine used in positional lighting.
- `obj.SetConeAngle (double )` - Set/Get the lighting cone angle of a positional light in degrees. This is the angle between the axis of the cone and a ray along the edge of the cone. A value of 180 indicates that you want no spot lighting effects just a positional light.
- `double = obj.GetConeAngle ()` - Set/Get the lighting cone angle of a positional light in degrees. This is the angle between the axis of the cone and a ray along the edge of the cone. A value of 180 indicates that you want no spot lighting effects just a positional light.
- `obj.SetAttenuationValues (double , double , double )` - Set/Get the quadratic attenuation constants. They are specified as constant, linear, and quadratic, in that order.
- `obj.SetAttenuationValues (double a[3])` - Set/Get the quadratic attenuation constants. They are specified as constant, linear, and quadratic, in that order.
- `double = obj. GetAttenuationValues ()` - Set/Get the quadratic attenuation constants. They are specified as constant, linear, and quadratic, in that order.
- `obj.SetTransformMatrix (vtkMatrix4x4 )` - Set/Get the light's transformation matrix. If a matrix is set for a light, the light's parameters (position and focal point) are transformed by the matrix before being rendered.
- `vtkMatrix4x4 = obj.GetTransformMatrix ()` - Set/Get the light's transformation matrix. If a matrix is set for a light, the light's parameters (position and focal point) are transformed by the matrix before being rendered.
- `obj.GetTransformedPosition (double a[3])` - Get the position of the light, modified by the transformation matrix (if it exists).
- `double = obj.GetTransformedPosition ()` - Get the position of the light, modified by the transformation matrix (if it exists).
- `obj.GetTransformedFocalPoint (double a[3])` - Get the focal point of the light, modified by the transformation matrix (if it exists).
- `double = obj.GetTransformedFocalPoint ()` - Get the focal point of the light, modified by the transformation matrix (if it exists).
- `obj.SetDirectionAngle (double elevation, double azimuth)` - Set the position and focal point of a light based on elevation and azimuth. The light is moved so it is shining from the given angle. Angles are given in degrees. If the light is a positional light, it is made directional instead.
- `obj.SetDirectionAngle (double ang[2])` - Set the position and focal point of a light based on elevation and azimuth. The light is moved so it is shining from the given angle. Angles are given in degrees. If the light is a positional light, it is made directional instead.
- `obj.DeepCopy (vtkLight light)` - Perform deep copy of this light.
- `obj.SetLightType (int )` - Set/Get the type of the light. A SceneLight is a light located in the world coordinate space. A light is initially created as a scene light.

  A Headlight is always located at the camera and is pointed at the camera's focal point. The renderer is free to modify the position and focal point of the camera at any time.

  A CameraLight is also attached to the camera, but is not necessarily located at the camera's position. CameraLights are defined in a coordinate space where the camera is located at (0, 0, 1), looking towards (0, 0, 0) at a distance of 1, with up being (0, 1, 0).

  Note: Use SetLightTypeToSceneLight, rather than SetLightType(3), since the former clears the light's transform matrix.

- `int = obj.GetLightType ()` - Set/Get the type of the light. A SceneLight is a light located in the world coordinate space. A light is initially created as a scene light.

  A Headlight is always located at the camera and is pointed at the camera's focal point. The renderer is free to modify the position and focal point of the camera at any time.

  A CameraLight is also attached to the camera, but is not necessarily located at the camera's position. CameraLights are defined in a coordinate space where the camera is located at (0, 0, 1), looking towards (0, 0, 0) at a distance of 1, with up being (0, 1, 0).

  Note: Use SetLightTypeToSceneLight, rather than SetLightType(3), since the former clears the light's transform matrix.

- `obj.SetLightTypeToHeadlight ()` - Set/Get the type of the light. A SceneLight is a light located in the world coordinate space. A light is initially created as a scene light.

  A Headlight is always located at the camera and is pointed at the camera's focal point. The renderer is free to modify the position and focal point of the camera at any time.

  A CameraLight is also attached to the camera, but is not necessarily located at the camera's position. CameraLights are defined in a coordinate space where the camera is located at (0, 0, 1), looking towards (0, 0, 0) at a distance of 1, with up being (0, 1, 0).

  Note: Use SetLightTypeToSceneLight, rather than SetLightType(3), since the former clears the light's transform matrix.

- `obj.SetLightTypeToSceneLight ()` - Set/Get the type of the light. A SceneLight is a light located in the world coordinate space. A light is initially created as a scene light.

  A Headlight is always located at the camera and is pointed at the camera's focal point. The renderer is free to modify the position and focal point of the camera at any time.

  A CameraLight is also attached to the camera, but is not necessarily located at the camera's position. CameraLights are defined in a coordinate space where the camera is located at (0, 0, 1), looking towards (0, 0, 0) at a distance of 1, with up being (0, 1, 0).

  Note: Use SetLightTypeToSceneLight, rather than SetLightType(3), since the former clears the light's transform matrix.

- `obj.SetLightTypeToCameraLight ()` - Query the type of the light.

- `int = obj.LightTypeIsHeadlight ()` - Query the type of the light.

- `int = obj.LightTypeIsSceneLight ()` - Query the type of the light.

- `int = obj.LightTypeIsCameraLight ()` - Query the type of the light.

39.93 vtkLightActor

39.93.1 Usage

vtkLightActor is a composite actor used to represent a spotlight. The cone angle is equal to the spotlight angle, the cone apex is at the position of the light, the direction of the light goes from the cone apex to the center of the base of the cone. The square frustum position is the light position, the frustum focal point is in the direction of the light direction. The frustum vertical view angle (aperture) (this is also the horizontal view angle as the frustum is square) is equal to twice the cone angle. The clipping range of the frustum is arbitrary set by the user (initially at 0.5,11.0).

To create an instance of class vtkLightActor, simply invoke its constructor as follows

```
obj = vtkLightActor
```

39.93.2 Methods

The class vtkLightActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLightActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLightActor = obj.NewInstance ()`
- `vtkLightActor = obj.SafeDownCast (vtkObject o)`
- `obj.SetLight (vtkLight light)` - The spotlight to represent. Initial value is NULL.
- `vtkLight = obj.GetLight ()` - The spotlight to represent. Initial value is NULL.
- `obj.SetClippingRange (double dNear, double dFar)` - Set/Get the location of the near and far clipping planes along the direction of projection. Both of these values must be positive. Initial values are (0.5,11.0)
- `obj.SetClippingRange (double a[2])` - Set/Get the location of the near and far clipping planes along the direction of projection. Both of these values must be positive. Initial values are (0.5,11.0)
- `double = obj. GetClippingRange ()` - Set/Get the location of the near and far clipping planes along the direction of projection. Both of these values must be positive. Initial values are (0.5,11.0)
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry? No.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `long = obj.GetMTime ()` - Get the actors mtime plus consider its properties and texture if set.

39.94 vtkLightCollection

39.94.1 Usage

vtkLightCollection represents and provides methods to manipulate a list of lights (i.e., vtkLight and subclasses). The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkLightCollection, simply invoke its constructor as follows

```
obj = vtkLightCollection
```

39.94.2 Methods

The class vtkLightCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLightCollection class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkLightCollection = obj.NewInstance ()`
- `vtkLightCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkLight a)` - Add a light to the list.
- `vtkLight = obj.GetNextItem ()` - Get the next light in the list. NULL is returned when the collection is exhausted.

39.95 vtkLightKit

39.95.1 Usage

vtkLightKit is designed to make general purpose lighting of vtk scenes simple, flexible, and attractive (or at least not horribly ugly without significant effort). Use a LightKit when you want more control over your lighting than you can get with the default vtk light, which is a headlight located at the camera. (HeadLights are very simple to use, but they don't show the shape of objects very well, don't give a good sense of "up" and "down", and don't evenly light the object.)

A LightKit consists of three lights, a key light, a fill light, and a headlight. The main light is the key light. It is usually positioned so that it appears like an overhead light (like the sun, or a ceiling light). It is generally positioned to shine down on the scene from about a 45 degree angle vertically and at least a little offset side to side. The key light usually at least about twice as bright as the total of all other lights in the scene to provide good modeling of object features.

The other lights in the kit (the fill light, headlight, and a pair of back lights) are weaker sources that provide extra illumination to fill in the spots that the key light misses. The fill light is usually positioned across from or opposite from the key light (though still on the same side of the object as the camera) in order to simulate diffuse reflections from other objects in the scene. The headlight, always located at the position of the camera, reduces the contrast between areas lit by the key and fill light. The two back lights, one on the left of the object as seen from the observer and one on the right, fill on the high-contrast areas behind the object. To enforce the relationship between the different lights, the intensity of the fill, back and headlights are set as a ratio to the key light brightness. Thus, the brightness of all the lights in the scene can be changed by changing the key light intensity.

All lights are directional lights (infinitely far away with no falloff). Lights move with the camera.

For simplicity, the position of lights in the LightKit can only be specified using angles: the elevation (latitude) and azimuth (longitude) of each light with respect to the camera, expressed in degrees. (Lights always shine on the camera's lookat point.) For example, a light at (elevation=0, azimuth=0) is located at the camera (a headlight). A light at (elevation=90, azimuth=0) is above the lookat point, shining down. Negative azimuth values move the lights clockwise as seen above, positive values counter-clockwise. So, a light at (elevation=45, azimuth=-20) is above and in front of the object and shining slightly from the left side.

vtkLightKit limits the colors that can be assigned to any light to those of incandescent sources such as light bulbs and sunlight. It defines a special color spectrum called "warmth" from which light colors can be chosen, where 0 is cold blue, 0.5 is neutral white, and 1 is deep sunset red. Colors close to 0.5 are "cool whites" and "warm whites," respectively.

Since colors far from white on the warmth scale appear less bright, key-to-fill and key-to-headlight ratios are skewed by key, fill, and headlight colors. If the flag MaintainLuminance is set, vtkLightKit will attempt to compensate for these perceptual differences by increasing the brightness of more saturated colors.

A LightKit is not explicitly part of the vtk pipeline. Rather, it is a composite object that controls the behavior of lights using a unified user interface. Every time a parameter of vtkLightKit is adjusted, the properties of its lights are modified.

.SECTION Credits vtkLightKit was originally written and contributed to vtk by Michael Halle (mhalle@bwh.harvard.edu) at the Surgical Planning Lab, Brigham and Women's Hospital.

To create an instance of class vtkLightKit, simply invoke its constructor as follows

```
obj = vtkLightKit
```

39.95.2 Methods

The class vtkLightKit has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLightKit class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLightKit = obj.NewInstance ()`
- `vtkLightKit = obj.SafeDownCast (vtkObject o)`
- `obj.SetKeyLightIntensity (double )` - Set/Get the intensity of the key light. The key light is the brightest light in the scene. The intensities of the other two lights are ratios of the key light's intensity.
- `double = obj.GetKeyLightIntensity ()` - Set/Get the intensity of the key light. The key light is the brightest light in the scene. The intensities of the other two lights are ratios of the key light's intensity.
- `obj.SetKeyToFillRatio (double )` - Set/Get the key-to-fill ratio. This ratio controls how bright the fill light is compared to the key light: larger values correspond to a dimmer fill light. The purpose of the fill light is to light parts of the object not lit by the key light, while still maintaining constrast. This type of lighting may correspond to indirect illumination from the key light, bounced off a wall, floor, or other object. The fill light should never be brighter than the key light: a good range for the key-to-fill ratio is between 2 and 10.
- `double = obj.GetKeyToFillRatioMinValue ()` - Set/Get the key-to-fill ratio. This ratio controls how bright the fill light is compared to the key light: larger values correspond to a dimmer fill light. The purpose of the fill light is to light parts of the object not lit by the key light, while still maintaining constrast. This type of lighting may correspond to indirect illumination from the key light, bounced off a wall, floor, or other object. The fill light should never be brighter than the key light: a good range for the key-to-fill ratio is between 2 and 10.
- `double = obj.GetKeyToFillRatioMaxValue ()` - Set/Get the key-to-fill ratio. This ratio controls how bright the fill light is compared to the key light: larger values correspond to a dimmer fill light. The purpose of the fill light is to light parts of the object not lit by the key light, while still maintaining constrast. This type of lighting may correspond to indirect illumination from the key light, bounced off a wall, floor, or other object. The fill light should never be brighter than the key light: a good range for the key-to-fill ratio is between 2 and 10.
- `double = obj.GetKeyToFillRatio ()` - Set/Get the key-to-fill ratio. This ratio controls how bright the fill light is compared to the key light: larger values correspond to a dimmer fill light. The purpose of the fill light is to light parts of the object not lit by the key light, while still maintaining constrast. This type of lighting may correspond to indirect illumination from the key light, bounced off a wall, floor, or other object. The fill light should never be brighter than the key light: a good range for the key-to-fill ratio is between 2 and 10.
- `obj.SetKeyToHeadRatio (double )` - Set/Get the key-to-headlight ratio. Similar to the key-to-fill ratio, this ratio controls how bright the headlight light is compared to the key light: larger values correspond to a dimmer headlight light. The headlight is special kind of fill light, lighting only the parts of the object that the camera can see. As such, a headlight tends to reduce the contrast of a scene. It can be used to fill in "shadows" of the object missed by the key and fill lights. The headlight should always be significantly dimmer than the key light: ratios of 2 to 15 are typical.

- `double = obj.GetKeyToHeadRatioMinValue ()` - Set/Get the key-to-headlight ratio. Similar to the key-to-fill ratio, this ratio controls how bright the headlight light is compared to the key light: larger values correspond to a dimmer headlight light. The headlight is special kind of fill light, lighting only the parts of the object that the camera can see. As such, a headlight tends to reduce the contrast of a scene. It can be used to fill in "shadows" of the object missed by the key and fill lights. The headlight should always be significantly dimmer than the key light: ratios of 2 to 15 are typical.

- `double = obj.GetKeyToHeadRatioMaxValue ()` - Set/Get the key-to-headlight ratio. Similar to the key-to-fill ratio, this ratio controls how bright the headlight light is compared to the key light: larger values correspond to a dimmer headlight light. The headlight is special kind of fill light, lighting only the parts of the object that the camera can see. As such, a headlight tends to reduce the contrast of a scene. It can be used to fill in "shadows" of the object missed by the key and fill lights. The headlight should always be significantly dimmer than the key light: ratios of 2 to 15 are typical.

- `double = obj.GetKeyToHeadRatio ()` - Set/Get the key-to-headlight ratio. Similar to the key-to-fill ratio, this ratio controls how bright the headlight light is compared to the key light: larger values correspond to a dimmer headlight light. The headlight is special kind of fill light, lighting only the parts of the object that the camera can see. As such, a headlight tends to reduce the contrast of a scene. It can be used to fill in "shadows" of the object missed by the key and fill lights. The headlight should always be significantly dimmer than the key light: ratios of 2 to 15 are typical.

- `obj.SetKeyToBackRatio (double )` - Set/Get the key-to-back light ratio. This ratio controls how bright the back lights are compared to the key light: larger values correspond to dimmer back lights. The back lights fill in the remaining high-contrast regions behind the object. Values between 2 and 10 are good.

- `double = obj.GetKeyToBackRatioMinValue ()` - Set/Get the key-to-back light ratio. This ratio controls how bright the back lights are compared to the key light: larger values correspond to dimmer back lights. The back lights fill in the remaining high-contrast regions behind the object. Values between 2 and 10 are good.

- `double = obj.GetKeyToBackRatioMaxValue ()` - Set/Get the key-to-back light ratio. This ratio controls how bright the back lights are compared to the key light: larger values correspond to dimmer back lights. The back lights fill in the remaining high-contrast regions behind the object. Values between 2 and 10 are good.

- `double = obj.GetKeyToBackRatio ()` - Set/Get the key-to-back light ratio. This ratio controls how bright the back lights are compared to the key light: larger values correspond to dimmer back lights. The back lights fill in the remaining high-contrast regions behind the object. Values between 2 and 10 are good.

- `obj.SetKeyLightWarmth (double )` - Set the warmth of each the lights. Warmth is a parameter that varies from 0 to 1, where 0 is "cold" (looks icy or lit by a very blue sky), 1 is "warm" (the red of a very red sunset, or the embers of a campfire), and 0.5 is a neutral white. The warmth scale is non-linear. Warmth values close to 0.5 are subtly "warmer" or "cooler," much like a warmer tungsten incandescent bulb, a cooler halogen, or daylight (cooler still). Moving further away from 0.5, colors become more quickly varying towards blues and reds. With regards to aesthetics, extremes of warmth should be used sparingly.

- `double = obj.GetKeyLightWarmth ()` - Set the warmth of each the lights. Warmth is a parameter that varies from 0 to 1, where 0 is "cold" (looks icy or lit by a very blue sky), 1 is "warm" (the red of a very red sunset, or the embers of a campfire), and 0.5 is a neutral white. The warmth scale is non-linear. Warmth values close to 0.5 are subtly "warmer" or "cooler," much like a warmer tungsten incandescent bulb, a cooler halogen, or daylight (cooler still). Moving further away from 0.5, colors become more quickly varying towards blues and reds. With regards to aesthetics, extremes of warmth should be used sparingly.

- `obj.SetFillLightWarmth (double )`
- `double = obj.GetFillLightWarmth ()`
- `obj.SetHeadLightWarmth (double )`
- `double = obj.GetHeadLightWarmth ()`
- `obj.SetBackLightWarmth (double )`
- `double = obj.GetBackLightWarmth ()`
- `double = obj. GetKeyLightColor ()` - Returns the floating-point RGB values of each of the light's color.
- `double = obj. GetFillLightColor ()` - Returns the floating-point RGB values of each of the light's color.
- `double = obj. GetHeadLightColor ()` - Returns the floating-point RGB values of each of the light's color.
- `double = obj. GetBackLightColor ()` - Returns the floating-point RGB values of each of the light's color.
- `obj.SetHeadlightWarmth (double v)` - To maintain a deprecation API:
- `double = obj.GetHeadlightWarmth ()` - To maintain a deprecation API:
- `obj.GetHeadlightColor (double color)` - To maintain a deprecation API:
- `obj.MaintainLuminanceOn ()` - If MaintainLuminance is set, the LightKit will attempt to maintain the apparent intensity of lights based on their perceptual brightnesses. By default, MaintainLuminance is off.
- `obj.MaintainLuminanceOff ()` - If MaintainLuminance is set, the LightKit will attempt to maintain the apparent intensity of lights based on their perceptual brightnesses. By default, MaintainLuminance is off.
- `int = obj.GetMaintainLuminance ()` - If MaintainLuminance is set, the LightKit will attempt to maintain the apparent intensity of lights based on their perceptual brightnesses. By default, MaintainLuminance is off.
- `obj.SetMaintainLuminance (int )` - If MaintainLuminance is set, the LightKit will attempt to maintain the apparent intensity of lights based on their perceptual brightnesses. By default, MaintainLuminance is off.
- `obj.SetKeyLightAngle (double elevation, double azimuth)` - Get/Set the position of the key, fill, and back lights using angular methods. Elevation corresponds to latitude, azimuth to longitude. It is recommended that the key light always be on the viewer's side of the object and above the object, while the fill light generally lights the part of the object not lit by the fill light. The headlight, which is always located at the viewer, can then be used to reduce the contrast in the image. There are a pair of back lights. They are located at the same elevation and at opposing azimuths (ie, one to the left, and one to the right). They are generally set at the equator (elevation = 0), and at approximately 120 degrees (lighting from each side and behind).
- `obj.SetKeyLightAngle (double angle[2])` - Get/Set the position of the key, fill, and back lights using angular methods. Elevation corresponds to latitude, azimuth to longitude. It is recommended that the key light always be on the viewer's side of the object and above the object, while the fill light generally lights the part of the object not lit by the fill light. The headlight, which is always located at the viewer, can then be used to reduce the contrast in the image. There are a pair of back lights. They

are located at the same elevation and at opposing azimuths (ie, one to the left, and one to the right). They are generally set at the equator (elevation = 0), and at approximately 120 degrees (lighting from each side and behind).

- `obj.SetKeyLightElevation (double x)`
- `obj.SetKeyLightAzimuth (double x)`
- `double = obj. GetKeyLightAngle ()`
- `double = obj.GetKeyLightElevation ()`
- `double = obj.GetKeyLightAzimuth ()`
- `obj.SetFillLightAngle (double elevation, double azimuth)`
- `obj.SetFillLightAngle (double angle[2])`
- `obj.SetFillLightElevation (double x)`
- `obj.SetFillLightAzimuth (double x)`
- `double = obj. GetFillLightAngle ()`
- `double = obj.GetFillLightElevation ()`
- `double = obj.GetFillLightAzimuth ()`
- `obj.SetBackLightAngle (double elevation, double azimuth)`
- `obj.SetBackLightAngle (double angle[2])`
- `obj.SetBackLightElevation (double x)`
- `obj.SetBackLightAzimuth (double x)`
- `double = obj. GetBackLightAngle ()`
- `double = obj.GetBackLightElevation ()`
- `double = obj.GetBackLightAzimuth ()`
- `obj.AddLightsToRenderer (vtkRenderer renderer)` - Add lights to, or remove lights from, a renderer. Lights may be added to more than one renderer, if desired.
- `obj.RemoveLightsFromRenderer (vtkRenderer renderer)` - Add lights to, or remove lights from, a renderer. Lights may be added to more than one renderer, if desired.
- `obj.DeepCopy (vtkLightKit kit)`
- `obj.Modified ()`
- `obj.Update ()`

39.96 vtkLightsPass

39.96.1 Usage

Render the lights.

This pass expects an initialized camera. It disables all the lights, apply transformations for lights following the camera, and turn on the enables lights.

To create an instance of class vtkLightsPass, simply invoke its constructor as follows

```
obj = vtkLightsPass
```

39.96.2 Methods

The class vtkLightsPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLightsPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLightsPass = obj.NewInstance ()`
- `vtkLightsPass = obj.SafeDownCast (vtkObject o)`

39.97 vtkLineIntegralConvolution2D

39.97.1 Usage

This class resorts to GLSL to implement GPU-based Line Integral Convolution (LIC) for visualizing a 2D vector field that may be obtained by projecting an original 3D vector field onto a surface (such that the resulting 2D vector at each grid point on the surface is tangential to the local normal, as done in vtkSurfaceLICPainter).

As an image-based technique, 2D LIC works by (1) integrating a bidirectional streamline from the center of each pixel (of the LIC output image), (2) locating the pixels along / hit by this streamline as the correlated pixels of the starting pixel (seed point / pixel), (3) indexing a (usually white) noise texture (another input to LIC, in addition to the 2D vector field, usually with the same size as that of the 2D vetor field) to determine the values (colors) of these pixels (the starting and the correlated pixels), typically through bilinear interpolation, and (4) performing convolution (weighted averaging) on these values, by adopting a low-pass filter (such as box, ramp, and Hanning kernels), to obtain the result value (color) that is then assigned to the seed pixel.

The GLSL-based GPU implementation herein maps the aforementioned pipeline to fragment shaders and a box kernel is employed. Both the white noise and the vector field are provided to the GPU as texture objects (supported by the multi-texturing capability). In addition, there are four texture objects (color buffers) allocated to constitute two pairs that work in a ping-pong fashion, with one as the read buffers and the other as the write / render targets. Maintained by a frame buffer object (GL_EXT_framebuffer_object), each pair employs one buffer to store the current (dynamically updated) position (by means of the texture coordinate that keeps being warped by the underlying vector) of the (virtual) particle initially released from each fragment while using the bother buffer to store the current (dynamically updated too) accumulated texture value that each seed fragment (before the 'mesh' is warped) collects. Given NumberOfSteps integration steps in each direction, there are a total of (2 * NumberOfSteps + 1) fragments (including the seed fragment) are convolved and each contributes 1 / (2 * NumberOfSteps + 1) of the associated texture value to fulfill the box filter.

One pass of LIC (basic LIC) tends to produce low-contrast / blurred images and vtkLineIntegralConvolution2D provides an option for creating enhanced LIC images. Enhanced LIC improves image quality by increasing inter-streamline contrast while suppressing artifacts. It performs two passes of LIC, with a 3x3 Laplacian high-pass filter in between that processes the output of pass #1 LIC and forwards the result as the input 'noise' to pass #2 LIC. Enhanced LIC automatically degenerates to basic LIC during user interaction.

vtkLineIntegralConvolution2D applies masking to zero-vector fragments so that un-filtered white noise areas are made totally transparent by class vtkSurfaceLICPainter to show the underlying geometry surface.

.SECTION Required OpenGL Extensins GL_ARB_texture_non_power_of_two GL_VERSION_2_0 GL_ARB_texture_float GL_ARB_draw_buffers GL_EXT_framebuffer_object

To create an instance of class vtkLineIntegralConvolution2D, simply invoke its constructor as follows

```
obj = vtkLineIntegralConvolution2D
```

39.97.2 Methods

The class vtkLineIntegralConvolution2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLineIntegralConvolution2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLineIntegralConvolution2D = obj.NewInstance ()`
- `vtkLineIntegralConvolution2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnhancedLIC (int )` - Enable/Disable enhanced LIC that improves image quality by increasing inter-streamline contrast while suppressing artifacts. Enhanced LIC performs two passes of LIC, with a 3x3 Laplacian high-pass filter in between that processes the output of pass #1 LIC and forwards the result as the input 'noise' to pass #2 LIC. This flag is automatically turned off during user interaction.
- `int = obj.GetEnhancedLIC ()` - Enable/Disable enhanced LIC that improves image quality by increasing inter-streamline contrast while suppressing artifacts. Enhanced LIC performs two passes of LIC, with a 3x3 Laplacian high-pass filter in between that processes the output of pass #1 LIC and forwards the result as the input 'noise' to pass #2 LIC. This flag is automatically turned off during user interaction.
- `obj.EnhancedLICOn ()` - Enable/Disable enhanced LIC that improves image quality by increasing inter-streamline contrast while suppressing artifacts. Enhanced LIC performs two passes of LIC, with a 3x3 Laplacian high-pass filter in between that processes the output of pass #1 LIC and forwards the result as the input 'noise' to pass #2 LIC. This flag is automatically turned off during user interaction.
- `obj.EnhancedLICOff ()` - Enable/Disable enhanced LIC that improves image quality by increasing inter-streamline contrast while suppressing artifacts. Enhanced LIC performs two passes of LIC, with a 3x3 Laplacian high-pass filter in between that processes the output of pass #1 LIC and forwards the result as the input 'noise' to pass #2 LIC. This flag is automatically turned off during user interaction.
- `obj.SetLICForSurface (int )`
- `int = obj.GetLICForSurface ()`
- `obj.LICForSurfaceOn ()`
- `obj.LICForSurfaceOff ()`
- `obj.SetNumberOfSteps (int )` - Number of streamline integration steps (initial value is 1). In term of visual quality, the greater (within some range) the better.
- `int = obj.GetNumberOfSteps ()` - Number of streamline integration steps (initial value is 1). In term of visual quality, the greater (within some range) the better.
- `obj.SetLICStepSize (double )` - Get/Set the streamline integration step size (0.01 by default). This is the length of each step in normalized image space i.e. in range [0, 1]. In term of visual quality, the smaller the better. The type for the interface is double as VTK interface is, but GPU only supports float. Thus it will be converted to float in the execution of the algorithm.
- `double = obj.GetLICStepSizeMinValue ()` - Get/Set the streamline integration step size (0.01 by default). This is the length of each step in normalized image space i.e. in range [0, 1]. In term of visual quality, the smaller the better. The type for the interface is double as VTK interface is, but GPU only supports float. Thus it will be converted to float in the execution of the algorithm.

- `double = obj.GetLICStepSizeMaxValue ()` - Get/Set the streamline integration step size (0.01 by default). This is the length of each step in normalized image space i.e. in range [0, 1]. In term of visual quality, the smaller the better. The type for the interface is double as VTK interface is, but GPU only supports float. Thus it will be converted to float in the execution of the algorithm.

- `double = obj.GetLICStepSize ()` - Get/Set the streamline integration step size (0.01 by default). This is the length of each step in normalized image space i.e. in range [0, 1]. In term of visual quality, the smaller the better. The type for the interface is double as VTK interface is, but GPU only supports float. Thus it will be converted to float in the execution of the algorithm.

- `obj.SetNoise (vtkTextureObject noise)` - Set/Get the input white noise texture (initial value is NULL).

- `vtkTextureObject = obj.GetNoise ()` - Set/Get the input white noise texture (initial value is NULL).

- `obj.SetVectorField (vtkTextureObject vectorField)` - Set/Get the vector field (initial value is NULL).

- `vtkTextureObject = obj.GetVectorField ()` - Set/Get the vector field (initial value is NULL).

- `obj.SetComponentIds (int , int )` - If VectorField has ¿= 3 components, we must choose which 2 components form the (X, Y) components for the vector field. Must be in the range [0, 3].

- `obj.SetComponentIds (int a[2])` - If VectorField has ¿= 3 components, we must choose which 2 components form the (X, Y) components for the vector field. Must be in the range [0, 3].

- `int = obj. GetComponentIds ()` - If VectorField has ¿= 3 components, we must choose which 2 components form the (X, Y) components for the vector field. Must be in the range [0, 3].

- `obj.SetGridSpacings (double , double )` - Set/Get the spacing in each dimension of the plane on which the vector field is defined. This class performs LIC in the normalized image space and hence generally it needs to transform the input vector field (given in physical space) to the normalized image space. The Spacing is needed to determine the tranform. Default is (1.0, 1.0). It is possible to disable vector transformation by setting TransformVectors to 0.

- `obj.SetGridSpacings (double a[2])` - Set/Get the spacing in each dimension of the plane on which the vector field is defined. This class performs LIC in the normalized image space and hence generally it needs to transform the input vector field (given in physical space) to the normalized image space. The Spacing is needed to determine the tranform. Default is (1.0, 1.0). It is possible to disable vector transformation by setting TransformVectors to 0.

- `double = obj. GetGridSpacings ()` - Set/Get the spacing in each dimension of the plane on which the vector field is defined. This class performs LIC in the normalized image space and hence generally it needs to transform the input vector field (given in physical space) to the normalized image space. The Spacing is needed to determine the tranform. Default is (1.0, 1.0). It is possible to disable vector transformation by setting TransformVectors to 0.

- `obj.SetTransformVectors (int )` - This class performs LIC in the normalized image space. Hence, by default it transforms the input vectors to the normalized image space (using the GridSpacings and input vector field dimensions). Set this to 0 to disable tranformation if the vectors are already tranformed.

- `int = obj.GetTransformVectorsMinValue ()` - This class performs LIC in the normalized image space. Hence, by default it transforms the input vectors to the normalized image space (using the GridSpacings and input vector field dimensions). Set this to 0 to disable tranformation if the vectors are already tranformed.

- `int = obj.GetTransformVectorsMaxValue ()` - This class performs LIC in the normalized image space. Hence, by default it transforms the input vectors to the normalized image space (using the GridSpacings and input vector field dimensions). Set this to 0 to disable tranformation if the vectors are already tranformed.

- `obj.TransformVectorsOn ()` - This class performs LIC in the normalized image space. Hence, by default it transforms the input vectors to the normalized image space (using the GridSpacings and input vector field dimensions). Set this to 0 to disable tranformation if the vectors are already tranformed.

- `obj.TransformVectorsOff ()` - This class performs LIC in the normalized image space. Hence, by default it transforms the input vectors to the normalized image space (using the GridSpacings and input vector field dimensions). Set this to 0 to disable tranformation if the vectors are already tranformed.

- `int = obj.GetTransformVectors ()` - This class performs LIC in the normalized image space. Hence, by default it transforms the input vectors to the normalized image space (using the GridSpacings and input vector field dimensions). Set this to 0 to disable tranformation if the vectors are already tranformed.

- `obj.SetMagnification (int )` - The the magnification factor (default is 1.0).

- `int = obj.GetMagnificationMinValue ()` - The the magnification factor (default is 1.0).

- `int = obj.GetMagnificationMaxValue ()` - The the magnification factor (default is 1.0).

- `int = obj.GetMagnification ()` - The the magnification factor (default is 1.0).

- `obj.SetVectorShiftScale (double shift, double scale)` - Returns if the context supports the required extensions.

- `int = obj.Execute ()` - Perform the LIC and obtain the LIC texture. Return 1 if no error.

- `int = obj.Execute (int extent[4])` - Same as Execute() except that the LIC operation is performed only on a window (given by the ęxtent) in the input VectorField. The ęxtent is relative to the input VectorField. The output LIC image will be of the size specified by extent.

- `int = obj.Execute (int extent[4])` - Same as Execute() except that the LIC operation is performed only on a window (given by the ęxtent) in the input VectorField. The ęxtent is relative to the input VectorField. The output LIC image will be of the size specified by extent.

- `obj.SetLIC (vtkTextureObject lic)` - LIC texture (initial value is NULL) set by Execute().

- `vtkTextureObject = obj.GetLIC ()` - LIC texture (initial value is NULL) set by Execute().

39.98 vtkLinesPainter

39.98.1 Usage

This painter tries to paint lines efficiently. Request to Render any other primitive are ignored and not passed to the delegate painter, if any. This painter cannot handle cell colors/normals. If they are present the request is passed on to the Delegate painter. If this class is able to render the primitive, the render request is not propagated to the delegate painter.

To create an instance of class vtkLinesPainter, simply invoke its constructor as follows

```
obj = vtkLinesPainter
```

39.98.2 Methods

The class vtkLinesPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLinesPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLinesPainter = obj.NewInstance ()`
- `vtkLinesPainter = obj.SafeDownCast (vtkObject o)`

39.99 vtkLODActor

39.99.1 Usage

vtkLODActor is an actor that stores multiple levels of detail (LOD) and can automatically switch between them. It selects which level of detail to use based on how much time it has been allocated to render. Currently a very simple method of TotalTime/NumberOfActors is used. (In the future this should be modified to dynamically allocate the rendering time between different actors based on their needs.)

There are three levels of detail by default. The top level is just the normal data. The lowest level of detail is a simple bounding box outline of the actor. The middle level of detail is a point cloud of a fixed number of points that have been randomly sampled from the mapper's input data. Point attributes are copied over to the point cloud. These two lower levels of detail are accomplished by creating instances of a vtkOutlineFilter (low-res) and vtkMaskPoints (medium-res). Additional levels of detail can be add using the AddLODMapper() method.

To control the frame rate, you typically set the vtkRenderWindowInteractor DesiredUpdateRate and StillUpdateRate. This then will cause vtkLODActor to adjust its LOD to fulfill the requested update rate.

For greater control on levels of detail, see also vtkLODProp3D. That class allows arbitrary definition of each LOD.

To create an instance of class vtkLODActor, simply invoke its constructor as follows

```
obj = vtkLODActor
```

39.99.2 Methods

The class vtkLODActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLODActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLODActor = obj.NewInstance ()`
- `vtkLODActor = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer , vtkMapper )` - This causes the actor to be rendered. It, in turn, will render the actor's property and then mapper.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - This method is used internally by the rendering process. We overide the superclass method to properly set the estimated render time.

- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.AddLODMapper (vtkMapper mapper)` - Add another level of detail. They do not have to be in any order of complexity.
- `obj.SetLowResFilter (vtkPolyDataAlgorithm )` - You may plug in your own filters to decimate/subsample the input. The default is to use a vtkOutlineFilter (low-res) and vtkMaskPoints (medium-res).
- `obj.SetMediumResFilter (vtkPolyDataAlgorithm )` - You may plug in your own filters to decimate/subsample the input. The default is to use a vtkOutlineFilter (low-res) and vtkMaskPoints (medium-res).
- `vtkPolyDataAlgorithm = obj.GetLowResFilter ()` - You may plug in your own filters to decimate/subsample the input. The default is to use a vtkOutlineFilter (low-res) and vtkMaskPoints (medium-res).
- `vtkPolyDataAlgorithm = obj.GetMediumResFilter ()` - You may plug in your own filters to decimate/subsample the input. The default is to use a vtkOutlineFilter (low-res) and vtkMaskPoints (medium-res).
- `int = obj.GetNumberOfCloudPoints ()` - Set/Get the number of random points for the point cloud.
- `obj.SetNumberOfCloudPoints (int )` - Set/Get the number of random points for the point cloud.
- `vtkMapperCollection = obj.GetLODMappers ()` - All the mappers for different LODs are stored here. The order is not important.
- `obj.Modified ()` - When this objects gets modified, this method also modifies the object.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of an LOD actor. Overloads the virtual vtkProp method.

39.100 vtkLODProp3D

39.100.1 Usage

vtkLODProp3D is a class to support level of detail rendering for Prop3D. Any number of mapper/property/texture items can be added to this object. Render time will be measured, and will be used to select a LOD based on the AllocatedRenderTime of this Prop3D. Depending on the type of the mapper/property, a vtkActor or a vtkVolume will be created behind the scenes.

To create an instance of class vtkLODProp3D, simply invoke its constructor as follows

```
obj = vtkLODProp3D
```

39.100.2 Methods

The class vtkLODProp3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLODProp3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLODProp3D = obj.NewInstance ()`

- `vtkLODProp3D = obj.SafeDownCast (vtkObject o)`
- `double = obj.GetBounds ()` - Standard vtkProp method to get 3D bounds of a 3D prop
- `obj.GetBounds (double bounds[6])` - Standard vtkProp method to get 3D bounds of a 3D prop
- `int = obj.AddLOD (vtkMapper m, vtkProperty p, vtkProperty back, vtkTexture t, double time)` - Add a level of detail with a given mapper, property, backface property, texture, and guess of rendering time. The property and texture fields can be set to NULL (the other methods are included for script access where null variables are not allowed). The time field can be set to 0.0 indicating that no initial guess for rendering time is being supplied. The returned integer value is an ID that can be used later to delete this LOD, or set it as the selected LOD.
- `int = obj.AddLOD (vtkMapper m, vtkProperty p, vtkTexture t, double time)` - Add a level of detail with a given mapper, property, backface property, texture, and guess of rendering time. The property and texture fields can be set to NULL (the other methods are included for script access where null variables are not allowed). The time field can be set to 0.0 indicating that no initial guess for rendering time is being supplied. The returned integer value is an ID that can be used later to delete this LOD, or set it as the selected LOD.
- `int = obj.AddLOD (vtkMapper m, vtkProperty p, vtkProperty back, double time)` - Add a level of detail with a given mapper, property, backface property, texture, and guess of rendering time. The property and texture fields can be set to NULL (the other methods are included for script access where null variables are not allowed). The time field can be set to 0.0 indicating that no initial guess for rendering time is being supplied. The returned integer value is an ID that can be used later to delete this LOD, or set it as the selected LOD.
- `int = obj.AddLOD (vtkMapper m, vtkProperty p, double time)` - Add a level of detail with a given mapper, property, backface property, texture, and guess of rendering time. The property and texture fields can be set to NULL (the other methods are included for script access where null variables are not allowed). The time field can be set to 0.0 indicating that no initial guess for rendering time is being supplied. The returned integer value is an ID that can be used later to delete this LOD, or set it as the selected LOD.
- `int = obj.AddLOD (vtkMapper m, vtkTexture t, double time)` - Add a level of detail with a given mapper, property, backface property, texture, and guess of rendering time. The property and texture fields can be set to NULL (the other methods are included for script access where null variables are not allowed). The time field can be set to 0.0 indicating that no initial guess for rendering time is being supplied. The returned integer value is an ID that can be used later to delete this LOD, or set it as the selected LOD.
- `int = obj.AddLOD (vtkMapper m, double time)` - Add a level of detail with a given mapper, property, backface property, texture, and guess of rendering time. The property and texture fields can be set to NULL (the other methods are included for script access where null variables are not allowed). The time field can be set to 0.0 indicating that no initial guess for rendering time is being supplied. The returned integer value is an ID that can be used later to delete this LOD, or set it as the selected LOD.
- `int = obj.AddLOD (vtkAbstractVolumeMapper m, vtkVolumeProperty p, double time)` - Add a level of detail with a given mapper, property, backface property, texture, and guess of rendering time. The property and texture fields can be set to NULL (the other methods are included for script access where null variables are not allowed). The time field can be set to 0.0 indicating that no initial guess for rendering time is being supplied. The returned integer value is an ID that can be used later to delete this LOD, or set it as the selected LOD.
- `int = obj.AddLOD (vtkAbstractVolumeMapper m, double time)` - Add a level of detail with a given mapper, property, backface property, texture, and guess of rendering time. The property and texture fields can be set to NULL (the other methods are included for script access where null variables

are not allowed). The time field can be set to 0.0 indicating that no initial guess for rendering time is being supplied. The returned integer value is an ID that can be used later to delete this LOD, or set it as the selected LOD.

- `int = obj.GetNumberOfLODs ()` - Get the current number of LODs.
- `int = obj.GetCurrentIndex ()` - Get the current index, used to determine the ID of the next LOD that is added. Useful for guessing what IDs have been used (with NumberOfLODs, without depending on the constructor initialization to 1000.
- `obj.RemoveLOD (int id)` - Delete a level of detail given an ID. This is the ID returned by the AddLOD method
- `obj.SetLODProperty (int id, vtkProperty p)` - Methods to set / get the property of an LOD. Since the LOD could be a volume or an actor, you have to pass in the pointer to the property to get it. The returned property will be NULL if the id is not valid, or the property is of the wrong type for the corresponding Prop3D.
- `obj.SetLODProperty (int id, vtkVolumeProperty p)` - Methods to set / get the property of an LOD. Since the LOD could be a volume or an actor, you have to pass in the pointer to the property to get it. The returned property will be NULL if the id is not valid, or the property is of the wrong type for the corresponding Prop3D.
- `obj.SetLODMapper (int id, vtkMapper m)` - Methods to set / get the mapper of an LOD. Since the LOD could be a volume or an actor, you have to pass in the pointer to the mapper to get it. The returned mapper will be NULL if the id is not valid, or the mapper is of the wrong type for the corresponding Prop3D.
- `obj.SetLODMapper (int id, vtkAbstractVolumeMapper m)` - Methods to set / get the mapper of an LOD. Since the LOD could be a volume or an actor, you have to pass in the pointer to the mapper to get it. The returned mapper will be NULL if the id is not valid, or the mapper is of the wrong type for the corresponding Prop3D.
- `vtkAbstractMapper3D = obj.GetLODMapper (int id)` - Get the LODMapper as an vtkAbstractMapper3D. It is the user's respondibility to safe down cast this to a vtkMapper or vtkVolumeMapper as appropriate.
- `obj.SetLODBackfaceProperty (int id, vtkProperty t)` - Methods to set / get the backface property of an LOD. This method is only valid for LOD ids that are Actors (not Volumes)
- `obj.SetLODTexture (int id, vtkTexture t)` - Methods to set / get the texture of an LOD. This method is only valid for LOD ids that are Actors (not Volumes)
- `obj.EnableLOD (int id)` - Enable / disable a particular LOD. If it is disabled, it will not be used during automatic selection, but can be selected as the LOD if automatic LOD selection is off.
- `obj.DisableLOD (int id)` - Enable / disable a particular LOD. If it is disabled, it will not be used during automatic selection, but can be selected as the LOD if automatic LOD selection is off.
- `int = obj.IsLODEnabled (int id)` - Enable / disable a particular LOD. If it is disabled, it will not be used during automatic selection, but can be selected as the LOD if automatic LOD selection is off.
- `obj.SetLODLevel (int id, double level)` - Set the level of a particular LOD. When a LOD is selected for rendering because it has the largest render time that fits within the allocated time, all LOD are then checked to see if any one can render faster but has a lower (more resolution/better) level. This quantity is a double to ensure that a level can be inserted between 2 and 3.
- `double = obj.GetLODLevel (int id)` - Set the level of a particular LOD. When a LOD is selected for rendering because it has the largest render time that fits within the allocated time, all LOD are then checked to see if any one can render faster but has a lower (more resolution/better) level. This quantity is a double to ensure that a level can be inserted between 2 and 3.

- `double = obj.GetLODIndexLevel (int index)` - Set the level of a particular LOD. When a LOD is selected for rendering because it has the largest render time that fits within the allocated time, all LOD are then checked to see if any one can render faster but has a lower (more resolution/better) level. This quantity is a double to ensure that a level can be inserted between 2 and 3.
- `double = obj.GetLODEstimatedRenderTime (int id)` - Access method that can be used to find out the estimated render time (the thing used to select an LOD) for a given LOD ID or index. Value is returned in seconds.
- `double = obj.GetLODIndexEstimatedRenderTime (int index)` - Access method that can be used to find out the estimated render time (the thing used to select an LOD) for a given LOD ID or index. Value is returned in seconds.
- `obj.SetAutomaticLODSelection (int )` - Turn on / off automatic selection of LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `int = obj.GetAutomaticLODSelectionMinValue ()` - Turn on / off automatic selection of LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `int = obj.GetAutomaticLODSelectionMaxValue ()` - Turn on / off automatic selection of LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `int = obj.GetAutomaticLODSelection ()` - Turn on / off automatic selection of LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `obj.AutomaticLODSelectionOn ()` - Turn on / off automatic selection of LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `obj.AutomaticLODSelectionOff ()` - Turn on / off automatic selection of LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `obj.SetSelectedLODID (int )` - Set the id of the LOD that is to be drawn when automatic LOD selection is turned off.
- `int = obj.GetSelectedLODID ()` - Set the id of the LOD that is to be drawn when automatic LOD selection is turned off.
- `int = obj.GetLastRenderedLODID ()` - Get the ID of the previously (during the last render) selected LOD index
- `int = obj.GetPickLODID (void )` - Get the ID of the appropriate pick LOD index
- `obj.GetActors (vtkPropCollection )` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.
- `obj.GetVolumes (vtkPropCollection )` - For some exporters and other other operations we must be able to collect all the actors or volumes. These methods are used in that process.
- `obj.SetSelectedPickLODID (int id)` - Set the id of the LOD that is to be used for picking when automatic LOD pick selection is turned off.
- `int = obj.GetSelectedPickLODID ()` - Set the id of the LOD that is to be used for picking when automatic LOD pick selection is turned off.
- `obj.SetAutomaticPickLODSelection (int )` - Turn on / off automatic selection of picking LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.

- `int = obj.GetAutomaticPickLODSelectionMinValue ()` - Turn on / off automatic selection of picking LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `int = obj.GetAutomaticPickLODSelectionMaxValue ()` - Turn on / off automatic selection of picking LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `int = obj.GetAutomaticPickLODSelection ()` - Turn on / off automatic selection of picking LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `obj.AutomaticPickLODSelectionOn ()` - Turn on / off automatic selection of picking LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `obj.AutomaticPickLODSelectionOff ()` - Turn on / off automatic selection of picking LOD. This is on by default. If it is off, then the SelectedLODID is rendered regardless of rendering time or desired update rate.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this vtkLODProp3D.

39.101 vtkMapArrayValues

39.101.1 Usage

vtkMapArrayValues allows you to associate certain values of an attribute array (on either a vertex, edge, point, or cell) with different values in a newly created attribute array.

vtkMapArrayValues manages an internal STL map of vtkVariants that can be added to or cleared. When this filter executes, each "key" is searched for in the input array and the indices of the output array at which there were matches the set to the mapped "value".

You can control whether the input array values are passed to the output before the mapping occurs (using PassArray) or, if not, what value to set the unmapped indices to (using FillValue).

One application of this filter is to help address the dirty data problem. For example, using vtkMapArrayValues you could associate the vertex values "Foo, John", "Foo, John.", and "John Foo" with a single entity.

To create an instance of class vtkMapArrayValues, simply invoke its constructor as follows

```
obj = vtkMapArrayValues
```

39.101.2 Methods

The class vtkMapArrayValues has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMapArrayValues class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMapArrayValues = obj.NewInstance ()`
- `vtkMapArrayValues = obj.SafeDownCast (vtkObject o)`
- `obj.SetFieldType (int )` - Set/Get where the data is located that is being mapped. See FieldType enumeration for possible values. Default is POINT_DATA.

- `int = obj.GetFieldType ()` - Set/Get where the data is located that is being mapped. See FieldType enumeration for possible values. Default is POINT_DATA.
- `obj.SetPassArray (int )` - Set/Get whether to copy the data from the input array to the output array before the mapping occurs. If turned off, FillValue is used to initialize any unmapped array indices. Default is off.
- `int = obj.GetPassArray ()` - Set/Get whether to copy the data from the input array to the output array before the mapping occurs. If turned off, FillValue is used to initialize any unmapped array indices. Default is off.
- `obj.PassArrayOn ()` - Set/Get whether to copy the data from the input array to the output array before the mapping occurs. If turned off, FillValue is used to initialize any unmapped array indices. Default is off.
- `obj.PassArrayOff ()` - Set/Get whether to copy the data from the input array to the output array before the mapping occurs. If turned off, FillValue is used to initialize any unmapped array indices. Default is off.
- `obj.SetFillValue (double )` - Set/Get whether to copy the data from the input array to the output array before the mapping occurs. If turned off, FillValue is used to initialize any unmapped array indices. Default is -1.
- `double = obj.GetFillValue ()` - Set/Get whether to copy the data from the input array to the output array before the mapping occurs. If turned off, FillValue is used to initialize any unmapped array indices. Default is -1.
- `obj.SetInputArrayName (string )` - Set/Get the name of the input array. This must be set prior to execution.
- `string = obj.GetInputArrayName ()` - Set/Get the name of the input array. This must be set prior to execution.
- `obj.SetOutputArrayName (string )` - Set/Get the name of the output array. Default is "ArrayMap".
- `string = obj.GetOutputArrayName ()` - Set/Get the name of the output array. Default is "ArrayMap".
- `int = obj.GetOutputArrayType ()` - Set/Get the type of the output array. See vtkSetGet.h for possible values. Default is VTK_INT.
- `obj.SetOutputArrayType (int )` - Set/Get the type of the output array. See vtkSetGet.h for possible values. Default is VTK_INT.
- `obj.AddToMap (int from, int to)` - Add to the internal STL map. "from" should be a value in the input array and "to" should be the new value it gets assigned in the output array.
- `obj.AddToMap (int from, string to)` - Add to the internal STL map. "from" should be a value in the input array and "to" should be the new value it gets assigned in the output array.
- `obj.AddToMap (string from, int to)` - Add to the internal STL map. "from" should be a value in the input array and "to" should be the new value it gets assigned in the output array.
- `obj.AddToMap (string from, string to)` - Add to the internal STL map. "from" should be a value in the input array and "to" should be the new value it gets assigned in the output array.
- `obj.ClearMap ()` - Clear the internal map.
- `int = obj.GetMapSize ()` - Get the size of the internal map.

39.102 vtkMapper

39.102.1 Usage

vtkMapper is an abstract class to specify interface between data and graphics primitives. Subclasses of vtkMapper map data through a lookuptable and control the creation of rendering primitives that interface to the graphics library. The mapping can be controlled by supplying a lookup table and specifying a scalar range to map data through.

There are several important control mechanisms affecting the behavior of this object. The ScalarVisibility flag controls whether scalar data (if any) controls the color of the associated actor(s) that refer to the mapper. The ScalarMode ivar is used to determine whether scalar point data or cell data is used to color the object. By default, point data scalars are used unless there are none, in which cell scalars are used. Or you can explicitly control whether to use point or cell scalar data. Finally, the mapping of scalars through the lookup table varies depending on the setting of the ColorMode flag. See the documentation for the appropriate methods for an explanation.

Another important feature of this class is whether to use immediate mode rendering (ImmediateModeRenderingOn) or display list rendering (ImmediateModeRenderingOff). If display lists are used, a data structure is constructed (generally in the rendering library) which can then be rapidly traversed and rendered by the rendering library. The disadvantage of display lists is that they require additionally memory which may affect the performance of the system.

Another important feature of the mapper is the ability to shift the z-buffer to resolve coincident topology. For example, if you'd like to draw a mesh with some edges a different color, and the edges lie on the mesh, this feature can be useful to get nice looking lines. (See the ResolveCoincidentTopology-related methods.)

To create an instance of class vtkMapper, simply invoke its constructor as follows

```
obj = vtkMapper
```

39.102.2 Methods

The class vtkMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkMapper = obj.NewInstance ()`
- `vtkMapper = obj.SafeDownCast (vtkObject o)`
- `obj.ShallowCopy (vtkAbstractMapper m)` - Make a shallow copy of this mapper.
- `long = obj.GetMTime ()` - Overload standard modified time function. If lookup table is modified, then this object is modified as well.
- `obj.Render (vtkRenderer ren, vtkActor a)` - Method initiates the mapping process. Generally sent by the actor as each frame is rendered.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this mapper. The parameter window could be used to determine which graphic resources to release.
- `obj.SetLookupTable (vtkScalarsToColors lut)` - Specify a lookup table for the mapper to use.
- `vtkScalarsToColors = obj.GetLookupTable ()` - Specify a lookup table for the mapper to use.
- `obj.CreateDefaultLookupTable ()` - Create default lookup table. Generally used to create one when none is available with the scalar data.

- `obj.SetScalarVisibility (int )` - Turn on/off flag to control whether scalar data is used to color objects.
- `int = obj.GetScalarVisibility ()` - Turn on/off flag to control whether scalar data is used to color objects.
- `obj.ScalarVisibilityOn ()` - Turn on/off flag to control whether scalar data is used to color objects.
- `obj.ScalarVisibilityOff ()` - Turn on/off flag to control whether scalar data is used to color objects.
- `obj.SetStatic (int )` - Turn on/off flag to control whether the mapper's data is static. Static data means that the mapper does not propagate updates down the pipeline, greatly decreasing the time it takes to update many mappers. This should only be used if the data never changes.
- `int = obj.GetStatic ()` - Turn on/off flag to control whether the mapper's data is static. Static data means that the mapper does not propagate updates down the pipeline, greatly decreasing the time it takes to update many mappers. This should only be used if the data never changes.
- `obj.StaticOn ()` - Turn on/off flag to control whether the mapper's data is static. Static data means that the mapper does not propagate updates down the pipeline, greatly decreasing the time it takes to update many mappers. This should only be used if the data never changes.
- `obj.StaticOff ()` - Turn on/off flag to control whether the mapper's data is static. Static data means that the mapper does not propagate updates down the pipeline, greatly decreasing the time it takes to update many mappers. This should only be used if the data never changes.
- `obj.SetColorMode (int )` - Control how the scalar data is mapped to colors. By default (ColorModeToDefault), unsigned char scalars are treated as colors, and NOT mapped through the lookup table, while everything else is. Setting ColorModeToMapScalars means that all scalar data will be mapped through the lookup table. (Note that for multi-component scalars, the particular component to use for mapping can be specified using the SelectColorArray() method.)
- `int = obj.GetColorMode ()` - Control how the scalar data is mapped to colors. By default (ColorModeToDefault), unsigned char scalars are treated as colors, and NOT mapped through the lookup table, while everything else is. Setting ColorModeToMapScalars means that all scalar data will be mapped through the lookup table. (Note that for multi-component scalars, the particular component to use for mapping can be specified using the SelectColorArray() method.)
- `obj.SetColorModeToDefault ()` - Control how the scalar data is mapped to colors. By default (ColorModeToDefault), unsigned char scalars are treated as colors, and NOT mapped through the lookup table, while everything else is. Setting ColorModeToMapScalars means that all scalar data will be mapped through the lookup table. (Note that for multi-component scalars, the particular component to use for mapping can be specified using the SelectColorArray() method.)
- `obj.SetColorModeToMapScalars ()` - Control how the scalar data is mapped to colors. By default (ColorModeToDefault), unsigned char scalars are treated as colors, and NOT mapped through the lookup table, while everything else is. Setting ColorModeToMapScalars means that all scalar data will be mapped through the lookup table. (Note that for multi-component scalars, the particular component to use for mapping can be specified using the SelectColorArray() method.)
- `string = obj.GetColorModeAsString ()` - Return the method of coloring scalar data.
- `obj.SetInterpolateScalarsBeforeMapping (int )` - By default, vertex color is used to map colors to a surface. Colors are interpolated after being mapped. This option avoids color interpolation by using a one dimensional texture map for the colors.
- `int = obj.GetInterpolateScalarsBeforeMapping ()` - By default, vertex color is used to map colors to a surface. Colors are interpolated after being mapped. This option avoids color interpolation by using a one dimensional texture map for the colors.

- `obj.InterpolateScalarsBeforeMappingOn ()` - By default, vertex color is used to map colors to a surface. Colors are interpolated after being mapped. This option avoids color interpolation by using a one dimensional texture map for the colors.
- `obj.InterpolateScalarsBeforeMappingOff ()` - By default, vertex color is used to map colors to a surface. Colors are interpolated after being mapped. This option avoids color interpolation by using a one dimensional texture map for the colors.
- `obj.SetUseLookupTableScalarRange (int )` - Control whether the mapper sets the lookuptable range based on its own ScalarRange, or whether it will use the LookupTable ScalarRange regardless of it's own setting. By default the Mapper is allowed to set the LookupTable range, but users who are sharing LookupTables between mappers/actors will probably wish to force the mapper to use the LookupTable unchanged.
- `int = obj.GetUseLookupTableScalarRange ()` - Control whether the mapper sets the lookuptable range based on its own ScalarRange, or whether it will use the LookupTable ScalarRange regardless of it's own setting. By default the Mapper is allowed to set the LookupTable range, but users who are sharing LookupTables between mappers/actors will probably wish to force the mapper to use the LookupTable unchanged.
- `obj.UseLookupTableScalarRangeOn ()` - Control whether the mapper sets the lookuptable range based on its own ScalarRange, or whether it will use the LookupTable ScalarRange regardless of it's own setting. By default the Mapper is allowed to set the LookupTable range, but users who are sharing LookupTables between mappers/actors will probably wish to force the mapper to use the LookupTable unchanged.
- `obj.UseLookupTableScalarRangeOff ()` - Control whether the mapper sets the lookuptable range based on its own ScalarRange, or whether it will use the LookupTable ScalarRange regardless of it's own setting. By default the Mapper is allowed to set the LookupTable range, but users who are sharing LookupTables between mappers/actors will probably wish to force the mapper to use the LookupTable unchanged.
- `obj.SetScalarRange (double , double )` - Specify range in terms of scalar minimum and maximum (smin,smax). These values are used to map scalars into lookup table. Has no effect when UseLookupTableScalarRange is true.
- `obj.SetScalarRange (double a[2])` - Specify range in terms of scalar minimum and maximum (smin,smax). These values are used to map scalars into lookup table. Has no effect when UseLookupTableScalarRange is true.
- `double = obj. GetScalarRange ()` - Specify range in terms of scalar minimum and maximum (smin,smax). These values are used to map scalars into lookup table. Has no effect when UseLookupTableScalarRange is true.
- `obj.SetImmediateModeRendering (int )` - Turn on/off flag to control whether data is rendered using immediate mode or note. Immediate mode rendering tends to be slower but it can handle larger datasets. The default value is immediate mode off. If you are having problems rendering a large dataset you might want to consider using immediate more rendering.
- `int = obj.GetImmediateModeRendering ()` - Turn on/off flag to control whether data is rendered using immediate mode or note. Immediate mode rendering tends to be slower but it can handle larger datasets. The default value is immediate mode off. If you are having problems rendering a large dataset you might want to consider using immediate more rendering.
- `obj.ImmediateModeRenderingOn ()` - Turn on/off flag to control whether data is rendered using immediate mode or note. Immediate mode rendering tends to be slower but it can handle larger datasets. The default value is immediate mode off. If you are having problems rendering a large dataset you might want to consider using immediate more rendering.

- `obj.ImmediateModeRenderingOff ()` - Turn on/off flag to control whether data is rendered using immediate mode or note. Immediate mode rendering tends to be slower but it can handle larger datasets. The default value is immediate mode off. If you are having problems rendering a large dataset you might want to consider using immediate more rendering.

- `obj.SetScalarMode (int )` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectColorArray before you call GetColors. When ScalarMode is set to use Field Data (ScalarModeToFieldData), you must call SelectColorArray to choose the field data array to be used to color cells. In this mode, if the poly data has triangle strips, the field data is treated as the celldata for each mini-cell formed by a triangle in the strip rather than the entire strip.

- `int = obj.GetScalarMode ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectColorArray before you call GetColors. When ScalarMode is set to use Field Data (ScalarModeToFieldData), you must call SelectColorArray to choose the field data array to be used to color cells. In this mode, if the poly data has triangle strips, the field data is treated as the celldata for each mini-cell formed by a triangle in the strip rather than the entire strip.

- `obj.SetScalarModeToDefault ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectColorArray before you call GetColors. When ScalarMode is set to use Field Data (ScalarModeToFieldData), you must call SelectColorArray to choose the field data array to be used to color cells. In this mode, if the poly data has triangle strips, the field data is treated as the celldata for each mini-cell formed by a triangle in the strip rather than the entire strip.

- `obj.SetScalarModeToUsePointData ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectColorArray before you call GetColors. When ScalarMode is set to use Field Data (ScalarModeToFieldData), you must call SelectColorArray to choose the field data array to be used to color cells. In this mode, if the poly data has triangle strips, the field data is treated as the celldata for each mini-cell formed by a triangle in the strip rather than the entire strip.

- `obj.SetScalarModeToUseCellData ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call

SelectColorArray before you call GetColors. When ScalarMode is set to use Field Data (ScalarModeToFieldData), you must call SelectColorArray to choose the field data array to be used to color cells. In this mode, if the poly data has triangle strips, the field data is treated as the celldata for each mini-cell formed by a triangle in the strip rather than the entire strip.

- `obj.SetScalarModeToUsePointFieldData ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectColorArray before you call GetColors. When ScalarMode is set to use Field Data (ScalarModeToFieldData), you must call SelectColorArray to choose the field data array to be used to color cells. In this mode, if the poly data has triangle strips, the field data is treated as the celldata for each mini-cell formed by a triangle in the strip rather than the entire strip.

- `obj.SetScalarModeToUseCellFieldData ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call SelectColorArray before you call GetColors. When ScalarMode is set to use Field Data (ScalarModeToFieldData), you must call SelectColorArray to choose the field data array to be used to color cells. In this mode, if the poly data has triangle strips, the field data is treated as the celldata for each mini-cell formed by a triangle in the strip rather than the entire strip.

- `obj.SetScalarModeToUseFieldData ()` - When ScalarMode is set to UsePointFieldData or UseCellFieldData, you can specify which array to use for coloring using these methods. The lookup table will decide how to convert vectors to colors.

- `obj.SelectColorArray (int arrayNum)` - When ScalarMode is set to UsePointFieldData or UseCellFieldData, you can specify which array to use for coloring using these methods. The lookup table will decide how to convert vectors to colors.

- `obj.SelectColorArray (string arrayName)` - When ScalarMode is set to UsePointFieldData or UseCellFieldData, you can specify which array to use for coloring using these methods. The lookup table will decide how to convert vectors to colors.

- `obj.ColorByArrayComponent (int arrayNum, int component)` - Legacy: These methods used to be used to specify the array component. It is better to do this in the lookup table.

- `obj.ColorByArrayComponent (string arrayName, int component)` - Legacy: These methods used to be used to specify the array component. It is better to do this in the lookup table.

- `string = obj.GetArrayName ()` - Get the array name or number and component to color by.

- `int = obj.GetArrayId ()` - Get the array name or number and component to color by.

- `int = obj.GetArrayAccessMode ()` - Get the array name or number and component to color by.

- `int = obj.GetArrayComponent ()` - Return the method for obtaining scalar data.

- `string = obj.GetScalarModeAsString ()` - Return the method for obtaining scalar data.

- `double = obj.GetBounds ()` - Return bounding box (array of six doubles) of data expressed as (xmin,xmax, ymin,ymax, zmin,zmax).

- `obj.GetBounds (double bounds[6])` - Return bounding box (array of six doubles) of data expressed as (xmin,xmax, ymin,ymax, zmin,zmax).
- `obj.SetRenderTime (double time)` - This instance variable is used by vtkLODActor to determine which mapper to use. It is an estimate of the time necessary to render. Setting the render time does not modify the mapper.
- `double = obj.GetRenderTime ()` - This instance variable is used by vtkLODActor to determine which mapper to use. It is an estimate of the time necessary to render. Setting the render time does not modify the mapper.
- `vtkDataSet = obj.GetInputAsDataSet ()` - Map the scalars (if there are any scalars and ScalarVisibility is on) through the lookup table, returning an unsigned char RGBA array. This is typically done as part of the rendering process. The alpha parameter allows the blending of the scalars with an additional alpha (typically which comes from a vtkActor, etc.)
- `vtkUnsignedCharArray = obj.MapScalars (double alpha)` - Map the scalars (if there are any scalars and ScalarVisibility is on) through the lookup table, returning an unsigned char RGBA array. This is typically done as part of the rendering process. The alpha parameter allows the blending of the scalars with an additional alpha (typically which comes from a vtkActor, etc.)
- `obj.SetScalarMaterialMode (int )` - Set/Get the light-model color mode.
- `int = obj.GetScalarMaterialMode ()` - Set/Get the light-model color mode.
- `obj.SetScalarMaterialModeToDefault ()` - Set/Get the light-model color mode.
- `obj.SetScalarMaterialModeToAmbient ()` - Set/Get the light-model color mode.
- `obj.SetScalarMaterialModeToDiffuse ()` - Set/Get the light-model color mode.
- `obj.SetScalarMaterialModeToAmbientAndDiffuse ()` - Set/Get the light-model color mode.
- `string = obj.GetScalarMaterialModeAsString ()` - Return the light-model color mode.
- `bool = obj.GetSupportsSelection ()` - WARNING: INTERNAL METHOD - NOT INTENDED FOR GENERAL USE DO NOT USE THIS METHOD OUTSIDE OF THE RENDERING PROCESS Used by vtkHardwareSelector to determine if the prop supports hardware selection.

39.103 vtkMapperCollection

39.103.1 Usage

vtkMapperCollection represents and provides methods to manipulate a list of mappers (i.e., vtkMapper and subclasses). The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkMapperCollection, simply invoke its constructor as follows

```
obj = vtkMapperCollection
```

39.103.2 Methods

The class vtkMapperCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkMapperCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkMapperCollection = obj.NewInstance ()`
- `vtkMapperCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkMapper a)` - Add an mapper to the list.
- `vtkMapper = obj.GetNextItem ()` - Get the next mapper in the list.
- `vtkMapper = obj.GetLastItem ()` - Get the last mapper in the list.

39.104 vtkOBJExporter

39.104.1 Usage

vtkOBJExporter is a concrete subclass of vtkExporter that writes wavefront .OBJ files in ASCII form. It also writes out a mtl file that contains the material properties. The filenames are derived by appending the .obj and .mtl suffix onto the user specified FilePrefix.

To create an instance of class vtkOBJExporter, simply invoke its constructor as follows

```
obj = vtkOBJExporter
```

39.104.2 Methods

The class vtkOBJExporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOBJExporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOBJExporter = obj.NewInstance ()`
- `vtkOBJExporter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFilePrefix (string )` - Specify the prefix of the files to write out. The resulting filenames will have .obj and .mtl appended to them.
- `string = obj.GetFilePrefix ()` - Specify the prefix of the files to write out. The resulting filenames will have .obj and .mtl appended to them.

39.105 vtkObserverMediator

39.105.1 Usage

The vtkObserverMediator is a helper class that manages requests for cursor changes from multiple interactor observers (e.g. widgets). It keeps a list of widgets (and their priorities) and their current requests for cursor shape. It then satisfies requests based on widget priority and the relative importance of the request (e.g., a lower priority widget requesting a particular cursor shape will overrule a higher priority widget requesting a default shape).

To create an instance of class vtkObserverMediator, simply invoke its constructor as follows

```
obj = vtkObserverMediator
```

39.105.2 Methods

The class vtkObserverMediator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkObserverMediator class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkObserverMediator = obj.NewInstance ()` - Standard macros.
- `vtkObserverMediator = obj.SafeDownCast (vtkObject o)` - Standard macros.
- `obj.SetInteractor (vtkRenderWindowInteractor iren)` - Specify the instance of vtkRenderWindow whose cursor shape is to be managed.
- `vtkRenderWindowInteractor = obj.GetInteractor ()` - Specify the instance of vtkRenderWindow whose cursor shape is to be managed.
- `int = obj.RequestCursorShape (vtkInteractorObserver , int cursorShape)` - Method used to request a cursor shape. Note that the shape is specified using one of the integral values determined in vtkRenderWindow.h. The method returns a non-zero value if the shape was successfully changed.
- `obj.RemoveAllCursorShapeRequests (vtkInteractorObserver )` - Remove all requests for cursor shape from a given interactor.

39.106 vtkOOGLExporter

39.106.1 Usage

vtkOOGLExporter is a concrete subclass of vtkExporter that writes Geomview OOGL files.

To create an instance of class vtkOOGLExporter, simply invoke its constructor as follows

```
obj = vtkOOGLExporter
```

39.106.2 Methods

The class vtkOOGLExporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOOGLExporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOOGLExporter = obj.NewInstance ()`
- `vtkOOGLExporter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify the name of the Geomview file to write.
- `string = obj.GetFileName ()` - Specify the name of the Geomview file to write.

39.107 vtkOpaquePass

39.107.1 Usage

vtkOpaquePass renders the opaque geometry of all the props that have the keys contained in vtkRenderState.

This pass expects an initialized depth buffer and color buffer. Initialized buffers means they have been cleared with farest z-value and background color/gradient/transparent color.

To create an instance of class vtkOpaquePass, simply invoke its constructor as follows

```
obj = vtkOpaquePass
```

39.107.2 Methods

The class vtkOpaquePass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpaquePass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpaquePass = obj.NewInstance ()`
- `vtkOpaquePass = obj.SafeDownCast (vtkObject o)`

39.108 vtkOpenGLActor

39.108.1 Usage

vtkOpenGLActor is a concrete implementation of the abstract class vtkActor. vtkOpenGLActor interfaces to the OpenGL rendering library.

To create an instance of class vtkOpenGLActor, simply invoke its constructor as follows

```
obj = vtkOpenGLActor
```

39.108.2 Methods

The class vtkOpenGLActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLActor = obj.NewInstance ()`
- `vtkOpenGLActor = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer ren, vtkMapper mapper)` - Actual actor render method.

39.109 vtkOpenGLCamera

39.109.1 Usage

vtkOpenGLCamera is a concrete implementation of the abstract class vtkCamera. vtkOpenGLCamera interfaces to the OpenGL rendering library.

To create an instance of class vtkOpenGLCamera, simply invoke its constructor as follows

```
obj = vtkOpenGLCamera
```

39.109.2 Methods

The class vtkOpenGLCamera has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLCamera class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLCamera = obj.NewInstance ()`
- `vtkOpenGLCamera = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer ren)` - Implement base class method.
- `obj.UpdateViewport (vtkRenderer ren)`

39.110 vtkOpenGLClipPlanesPainter

39.110.1 Usage

This painter is an openGL specific painter which handles clipplanes. This painter must typically be placed before the painter that do the primitive rendering.

To create an instance of class vtkOpenGLClipPlanesPainter, simply invoke its constructor as follows

```
obj = vtkOpenGLClipPlanesPainter
```

39.110.2 Methods

The class vtkOpenGLClipPlanesPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLClipPlanesPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLClipPlanesPainter = obj.NewInstance ()`
- `vtkOpenGLClipPlanesPainter = obj.SafeDownCast (vtkObject o)`

39.111 vtkOpenGLCoincidentTopologyResolutionPainter

39.111.1 Usage

Implementation for vtkCoincidentTopologyResolutionPainter using OpenGL.

To create an instance of class vtkOpenGLCoincidentTopologyResolutionPainter, simply invoke its constructor as follows

```
obj = vtkOpenGLCoincidentTopologyResolutionPainter
```

39.111.2 Methods

The class vtkOpenGLCoincidentTopologyResolutionPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLCoincidentTopologyResolutionPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLCoincidentTopologyResolutionPainter = obj.NewInstance ()`
- `vtkOpenGLCoincidentTopologyResolutionPainter = obj.SafeDownCast (vtkObject o)`

39.112 vtkOpenGLDisplayListPainter

39.112.1 Usage

vtkOpenGLDisplayListPainter creates an OpenGL display list for rendering. This painter creates a different display list for every render request with a different set of typeflags. If any of the data or inputs change, then all display lists are discarded.

To create an instance of class vtkOpenGLDisplayListPainter, simply invoke its constructor as follows

```
obj = vtkOpenGLDisplayListPainter
```

39.112.2 Methods

The class vtkOpenGLDisplayListPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLDisplayListPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLDisplayListPainter = obj.NewInstance ()`
- `vtkOpenGLDisplayListPainter = obj.SafeDownCast (vtkObject o)`
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this mapper. The parameter window could be used to determine which graphic resources to release. In this case, releases the display lists.

39.113 vtkOpenGLExtensionManager

39.113.1 Usage

vtkOpenGLExtensionManager acts as an interface to OpenGL extensions. It provides methods to query OpenGL extensions on the current or a given render window and to load extension function pointers. Currently does not support GLU extensions since the GLU library is not linked to VTK.

Before using vtkOpenGLExtensionManager, an OpenGL context must be created. This is generally done with a vtkRenderWindow. Note that simply creating the vtkRenderWindow is not sufficient. Usually you have to call Render before the actual OpenGL context is created. You can specify the RenderWindow with the SetRenderWindow method.

```
vtkOpenGLExtensionManager *extensions = vtkOpenGLExtensionManager::New();
extensions->SetRenderWindow(renwin);
```

If no vtkRenderWindow is specified, the current OpenGL context (if any) is used.

Generally speaking, when using OpenGL extensions, you will need an vtkOpenGLExtensionManager and the prototypes defined in vtkgl.h.

```
\#include ''vtkOpenGLExtensionManager.h''
\#include ''vtkgl.h''
```

The vtkgl.h include file contains all the constants and function pointers required for using OpenGL extensions in a portable and namespace safe way. vtkgl.h is built from parsed glext.h, glxext.h, and wglext.h files. Snapshots of these files are distributed with VTK, but you can also set CMake options to use other files.

To use an OpenGL extension, you first need to make an instance of vtkOpenGLExtensionManager and give it a vtkRenderWindow. You can then query the vtkOpenGLExtensionManager to see if the extension is supported with the ExtensionSupported method. Valid names for extensions are given in the OpenGL extension registry at http://www.opengl.org/registry/ . You can also grep vtkgl.h (which will be in the binary build directory if VTK is not installed) for appropriate names. There are also special extensions GL_VERSION_X_X (where X_X is replaced with a major and minor version, respectively) which contain all the constants and functions for OpenGL versions for which the gl.h header file is of an older version than the driver.

```
if (   !extensions->ExtensionSupported(''GL\_VERSION\_1\_2'')
    || !extensions->ExtensionSupported(''GL\_ARB\_multitexture'') ) {
  {
  vtkErrorMacro(''Required extensions not supported!'');
  }
```

Once you have verified that the extensions you want exist, before you use them you have to load them with the LoadExtension method.

```
extensions->LoadExtension(''GL\_VERSION\_1\_2'');
extensions->LoadExtension(''GL\_ARB\_multitexture'');
```

Alternatively, you can use the LoadSupportedExtension method, which checks whether the requested extension is supported and, if so, loads it. The LoadSupportedExtension method will not raise any errors or warnings if it fails, so it is important for callers to pay attention to the return value.

```
if (   extensions->LoadSupportedExtension(''GL\_VERSION\_1\_2'')
    \&\& extensions->LoadSupportedExtension(''GL\_ARB\_multitexture'') ) {
  {
```

```
  vtkgl::ActiveTexture(vtkgl::TEXTURE0\_ARB);
  }
else
  {
  vtkErrorMacro(''Required extensions could not be loaded!'');
  }
```

Once you have queried and loaded all of the extensions you need, you can delete the vtkOpenGLExtensionManager. To use a constant of an extension, simply replace the "GL_" prefix with "vtkgl::". Likewise, replace the "gl" prefix of functions with "vtkgl::". In rare cases, an extension will add a type. In this case, add vtkgl:: to the type (i.e. vtkgl::GLchar).

```
extensions->Delete();
...
vtkgl::ActiveTexture(vtkgl::TEXTURE0\_ARB);
```

For wgl extensions, replace the "WGL_" and "wgl" prefixes with "vtkwgl::". For glX extensions, replace the "GLX_" and "glX" prefixes with "vtkglX::".

To create an instance of class vtkOpenGLExtensionManager, simply invoke its constructor as follows

```
obj = vtkOpenGLExtensionManager
```

39.113.2 Methods

The class vtkOpenGLExtensionManager has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLExtensionManager class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLExtensionManager = obj.NewInstance ()`
- `vtkOpenGLExtensionManager = obj.SafeDownCast (vtkObject o)`
- `vtkRenderWindow = obj.GetRenderWindow ()` - Set/Get the render window to query extensions on. If set to null, justs queries the current render window.
- `obj.SetRenderWindow (vtkRenderWindow renwin)` - Set/Get the render window to query extensions on. If set to null, justs queries the current render window.
- `obj.Update ()` - Updates the extensions string.
- `string = obj.GetExtensionsString ()` - Returns a string listing all available extensions. Call Update first to validate this string.
- `int = obj.ExtensionSupported (string name)` - Returns true if the extension is supported, false otherwise.
- `obj.LoadExtension (string name)` - Loads all the functions associated with the given extension into the appropriate static members of vtkgl. This method emits a warning if the requested extension is not supported. It emits an error if the extension does not load successfully.

- `int = obj.LoadSupportedExtension (string name)` - Returns true if the extension is supported and loaded successfully, false otherwise. This method will "fail silently/gracefully" if the extension is not supported or does not load properly. It emits neither warnings nor errors. It is up to the caller to determine if the extension loaded properly by paying attention to the return value.

- `obj.LoadCorePromotedExtension (string name)` - Loads all the functions associated with the given core-promoted extension into the appropriate static members of vtkgl associated with the OpenGL version that promoted the extension as a core feature. This method emits a warning if the requested extension is not supported. It emits an error if the extension does not load successfully.

  For instance, extension GL_ARB_multitexture was promoted as a core feature into OpenGL 1.3. An implementation that uses this feature has to (IN THIS ORDER), check if OpenGL 1.3 is supported with ExtensionSupported("GL_VERSION_1_3"), if true, load the extension with LoadExtension("GL_VERSION_1_3"). If false, test for the extension with ExtensionSupported("GL_ARB_multitexture"),if true load the extension with this method LoadCorePromotedExtension("GL_ARB_multitexture"). If any of those loading stage succeeded, use vtgl::ActiveTexture() in any case, NOT vtgl::ActiveTextureARB(). This method avoids the use of if statements everywhere in implementations using core-promoted extensions. Without this method, the implementation code should look like:

```
int opengl\_1\_3=extensions->ExtensionSupported(''GL\_VERSION\_1\_3'');
if(opengl\_1\_3)
{
  extensions->LoadExtension(''GL\_VERSION\_1\_3'');
}
else
{
 if(extensions->ExtensionSupported(''GL\_ARB\_multitexture''))
 {
  extensions->LoadCorePromotedExtension(''GL\_ARB\_multitexture'');
 }
 else
 {
  vtkErrorMacro(''Required multitexture feature is not supported!'');
 }
}
...
if(opengl\_1\_3)
{
 vtkgl::ActiveTexture(vtkgl::TEXTURE0)
}
else
{
 vtkgl::ActiveTextureARB(vtkgl::TEXTURE0\_ARB)
}
```

  Thanks to this method, the code looks like:

```
int opengl\_1\_3=extensions->ExtensionSupported(''GL\_VERSION\_1\_3'');
if(opengl\_1\_3)
{
  extensions->LoadExtension(''GL\_VERSION\_1\_3'');
}
else
{
```

```
 if(extensions->ExtensionSupported(''GL\_ARB\_multitexture''))
 {
  extensions->LoadCorePromotedExtension(''GL\_ARB\_multitexture'');
 }
 else
 {
  vtkErrorMacro(''Required multitexture feature is not supported!'');
 }
}
...
vtkgl::ActiveTexture(vtkgl::TEXTURE0);
```

39.114 vtkOpenGLFreeTypeTextMapper

39.114.1 Usage

vtkOpenGLFreeTypeTextMapper provides 2D text annotation support for VTK using the FreeType and FTGL libraries. Normally the user should use vtktextMapper which in turn will use this class.

To create an instance of class vtkOpenGLFreeTypeTextMapper, simply invoke its constructor as follows

```
obj = vtkOpenGLFreeTypeTextMapper
```

39.114.2 Methods

The class vtkOpenGLFreeTypeTextMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLFreeTypeTextMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLFreeTypeTextMapper = obj.NewInstance ()`
- `vtkOpenGLFreeTypeTextMapper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderOverlay (vtkViewport viewport, vtkActor2D actor)` - Actally draw the text.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.GetSize (vtkViewport viewport, int size[2])` - What is the size of the rectangle required to draw this mapper ?

39.115 vtkOpenGLHardwareSupport

39.115.1 Usage

vtkOpenGLHardwareSupport is an implementation of methods used to query OpenGL and the hardware of what kind of graphics support is available. When VTK supports more than one Graphics API an abstract super class vtkHardwareSupport should be implemented for this class to derive from.

To create an instance of class vtkOpenGLHardwareSupport, simply invoke its constructor as follows

```
obj = vtkOpenGLHardwareSupport
```

39.115.2 Methods

The class vtkOpenGLHardwareSupport has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLHardwareSupport class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLHardwareSupport = obj.NewInstance ()`
- `vtkOpenGLHardwareSupport = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfFixedTextureUnits ()` - Return the number of fixed-function texture units.
- `int = obj.GetNumberOfTextureUnits ()` - Return the total number of texture image units accessible by a shader program.
- `bool = obj.GetSupportsMultiTexturing ()` - Test if MultiTexturing is supported.
- `vtkOpenGLExtensionManager = obj.GetExtensionManager ()` - Set/Get a reference to a vtkRenderWindow which is Required for most methods of this class to work.
- `obj.SetExtensionManager (vtkOpenGLExtensionManager extensionManager)` - Set/Get a reference to a vtkRenderWindow which is Required for most methods of this class to work.

39.116 vtkOpenGLImageActor

39.116.1 Usage

vtkOpenGLImageActor is a concrete implementation of the abstract class vtkImageActor. vtkOpenGLImageActor interfaces to the OpenGL rendering library.

To create an instance of class vtkOpenGLImageActor, simply invoke its constructor as follows

```
obj = vtkOpenGLImageActor
```

39.116.2 Methods

The class vtkOpenGLImageActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLImageActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLImageActor = obj.NewInstance ()`
- `vtkOpenGLImageActor = obj.SafeDownCast (vtkObject o)`
- `obj.Load (vtkRenderer ren)` - Implement base class method.
- `obj.Render (vtkRenderer ren)` - Implement base class method.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this texture. The parameter window could be used to determine which graphic resources to release. Using the same texture object in multiple render windows is NOT currently supported.

39.117 vtkOpenGLImageMapper

39.117.1 Usage

vtkOpenGLImageMapper is a concrete subclass of vtkImageMapper that renders images under OpenGL

To create an instance of class vtkOpenGLImageMapper, simply invoke its constructor as follows

```
obj = vtkOpenGLImageMapper
```

39.117.2 Methods

The class vtkOpenGLImageMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLImageMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLImageMapper = obj.NewInstance ()`
- `vtkOpenGLImageMapper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderOverlay (vtkViewport viewport, vtkActor2D actor)` - Called by the Render function in vtkImageMapper. Actually draws the image to the screen.
- `obj.RenderData (vtkViewport viewport, vtkImageData data, vtkActor2D actor)` - Called by the Render function in vtkImageMapper. Actually draws the image to the screen.

39.118 vtkOpenGLLight

39.118.1 Usage

vtkOpenGLLight is a concrete implementation of the abstract class vtkLight. vtkOpenGLLight interfaces to the OpenGL rendering library.

To create an instance of class vtkOpenGLLight, simply invoke its constructor as follows

```
obj = vtkOpenGLLight
```

39.118.2 Methods

The class vtkOpenGLLight has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLLight class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLLight = obj.NewInstance ()`
- `vtkOpenGLLight = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer ren, int light\_index)` - Implement base class method.

39.119 vtkOpenGLLightingPainter

39.119.1 Usage

This painter manages lighting. Ligting is disabled when rendering points/lines and no normals are present or rendering Polygons/TStrips and representation is points and no normals are present.

To create an instance of class vtkOpenGLLightingPainter, simply invoke its constructor as follows

```
obj = vtkOpenGLLightingPainter
```

39.119.2 Methods

The class vtkOpenGLLightingPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLLightingPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLLightingPainter = obj.NewInstance ()`
- `vtkOpenGLLightingPainter = obj.SafeDownCast (vtkObject o)`
- `double = obj.GetTimeToDraw ()` - This painter overrides GetTimeToDraw() to never pass the request to the delegate. This is done since this class may propagate a single render request multiple times to the delegate. In that case the time accumulation responsibility is borne by the painter causing the multiple rendering requests i.e. this painter itself.

39.120 vtkOpenGLPainterDeviceAdapter

39.120.1 Usage

An adapter between vtkPainter and the OpenGL rendering system. Only a handful of attributes with special meaning are supported. The OpenGL attribute used for each attribute is given below.

```
vtkDataSetAttributes::NORMALS          glNormal
vtkDataSetAttributes:::SCALARS         glColor
vtkDataSetAttributes::TCOORDS          glTexCoord
vtkDataSetAttributes::NUM\_ATTRIBUTES   glVertex
```

To create an instance of class vtkOpenGLPainterDeviceAdapter, simply invoke its constructor as follows

```
obj = vtkOpenGLPainterDeviceAdapter
```

39.120.2 Methods

The class vtkOpenGLPainterDeviceAdapter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLPainterDeviceAdapter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLPainterDeviceAdapter = obj.NewInstance ()`

- `vtkOpenGLPainterDeviceAdapter = obj.SafeDownCast (vtkObject o)`
- `obj.BeginPrimitive (int mode)` - Converts mode from VTK_* to GL_* and calls glBegin.
- `obj.EndPrimitive ()` - Calls glEnd.
- `int = obj.IsAttributesSupported (int attribute)` - Returns if the given attribute type is supported by the device. Returns 1 is supported, 0 otherwise.
- `obj.EnableAttributeArray (int index)` - Calls glEnableClientState or glDisableClientState.
- `obj.DisableAttributeArray (int index)` - Calls glEnableClientState or glDisableClientState.
- `obj.DrawArrays (int mode, vtkIdType first, vtkIdType count)` - Calls glDrawArrays. Mode is converted from VTK_* to GL_*.
- `int = obj.Compatible (vtkRenderer renderer)` - Returns true if renderer is a vtkOpenGLRenderer.
- `obj.MakeLighting (int mode)` - Turns lighting on and off.
- `int = obj.QueryLighting ()` - Returns current lighting setting.
- `obj.MakeMultisampling (int mode)` - Turns antialiasing on and off.
- `int = obj.QueryMultisampling ()` - Returns current antialiasing setting.
- `obj.MakeBlending (int mode)` - Turns blending on and off.
- `int = obj.QueryBlending ()` - Returns current blending setting.
- `obj.MakeVertexEmphasis (bool mode)` - Turns emphasis of vertices on or off for vertex selection. When emphasized verts are drawn nearer to the camera and are drawn larger than normal to make selection of them more reliable.
- `obj.MakeVertexEmphasisWithStencilCheck (int mode)` - @deprecated
- `obj.Stencil (int on)` - Control use of the stencil buffer (for vertex selection).
- `obj.WriteStencil (vtkIdType value)` - Control use of the stencil buffer (for vertex selection).
- `obj.TestStencil (vtkIdType value)` - Control use of the stencil buffer (for vertex selection).

39.121 vtkOpenGLPolyDataMapper

39.121.1 Usage

vtkOpenGLPolyDataMapper is a subclass of vtkPolyDataMapper. vtkOpenGLPolyDataMapper is a geometric PolyDataMapper for the OpenGL rendering library.

To create an instance of class vtkOpenGLPolyDataMapper, simply invoke its constructor as follows

```
obj = vtkOpenGLPolyDataMapper
```

39.121.2 Methods

The class vtkOpenGLPolyDataMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLPolyDataMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLPolyDataMapper = obj.NewInstance ()`
- `vtkOpenGLPolyDataMapper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderPiece (vtkRenderer ren, vtkActor a)` - Implement superclass render method.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this mapper. The parameter window could be used to determine which graphic resources to release.
- `int = obj.Draw (vtkRenderer ren, vtkActor a)` - Draw method for OpenGL.

39.122 vtkOpenGLPolyDataMapper2D

39.122.1 Usage

vtkOpenGLPolyDataMapper2D provides 2D PolyData annotation support for vtk under OpenGL. Normally the user should use vtkPolyDataMapper2D which in turn will use this class.

To create an instance of class vtkOpenGLPolyDataMapper2D, simply invoke its constructor as follows

```
obj = vtkOpenGLPolyDataMapper2D
```

39.122.2 Methods

The class vtkOpenGLPolyDataMapper2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLPolyDataMapper2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLPolyDataMapper2D = obj.NewInstance ()`
- `vtkOpenGLPolyDataMapper2D = obj.SafeDownCast (vtkObject o)`
- `obj.RenderOverlay (vtkViewport viewport, vtkActor2D actor)` - Actually draw the poly data.

39.123 vtkOpenGLProperty

39.123.1 Usage

vtkOpenGLProperty is a concrete implementation of the abstract class vtkProperty. vtkOpenGLProperty interfaces to the OpenGL rendering library.

To create an instance of class vtkOpenGLProperty, simply invoke its constructor as follows

```
obj = vtkOpenGLProperty
```

39.123.2 Methods

The class vtkOpenGLProperty has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLProperty class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLProperty = obj.NewInstance ()`
- `vtkOpenGLProperty = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkActor a, vtkRenderer ren)` - Implement base class method.
- `obj.BackfaceRender (vtkActor a, vtkRenderer ren)` - Implement base class method.
- `obj.AddShaderVariable (string name, int numVars, int x)` - Provide values to initialize shader variables. Useful to initialize shader variables that change over time (animation, GUI widgets inputs, etc.) - name - hardware name of the uniform variable - numVars - number of variables being set - x - values
- `obj.AddShaderVariable (string name, int numVars, float x)` - Provide values to initialize shader variables. Useful to initialize shader variables that change over time (animation, GUI widgets inputs, etc.) - name - hardware name of the uniform variable - numVars - number of variables being set - x - values
- `obj.AddShaderVariable (string name, int numVars, double x)` - Provide values to initialize shader variables. Useful to initialize shader variables that change over time (animation, GUI widgets inputs, etc.) - name - hardware name of the uniform variable - numVars - number of variables being set - x - values

39.124 vtkOpenGLRenderer

39.124.1 Usage

vtkOpenGLRenderer is a concrete implementation of the abstract class vtkRenderer. vtkOpenGLRenderer interfaces to the OpenGL graphics library.

To create an instance of class vtkOpenGLRenderer, simply invoke its constructor as follows

```
obj = vtkOpenGLRenderer
```

39.124.2 Methods

The class vtkOpenGLRenderer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLRenderer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLRenderer = obj.NewInstance ()`
- `vtkOpenGLRenderer = obj.SafeDownCast (vtkObject o)`
- `obj.DeviceRender (void )` - Concrete open gl render method.

- `obj.DeviceRenderTranslucentPolygonalGeometry ()` - Render translucent polygonal geometry. Default implementation just call UpdateTranslucentPolygonalGeometry(). Subclasses of vtkRenderer that can deal with depth peeling must override this method.
- `obj.ClearLights (void )` - Internal method temporarily removes lights before reloading them into graphics pipeline.
- `obj.Clear (void )`
- `int = obj.UpdateLights (void )` - Ask lights to load themselves into graphics pipeline.
- `int = obj.GetDepthPeelingHigherLayer ()` - Is rendering at translucent geometry stage using depth peeling and rendering a layer other than the first one? (Boolean value) If so, the uniform variables UseTexture and Texture can be set. (Used by vtkOpenGLProperty or vtkOpenGLTexture)

39.125 vtkOpenGLRenderWindow

39.125.1 Usage

vtkOpenGLRenderWindow is a concrete implementation of the abstract class vtkRenderWindow. vtkOpenGLRenderer interfaces to the OpenGL graphics library. Application programmers should normally use vtkRenderWindow instead of the OpenGL specific version.

To create an instance of class vtkOpenGLRenderWindow, simply invoke its constructor as follows

```
obj = vtkOpenGLRenderWindow
```

39.125.2 Methods

The class vtkOpenGLRenderWindow has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLRenderWindow class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLRenderWindow = obj.NewInstance ()`
- `vtkOpenGLRenderWindow = obj.SafeDownCast (vtkObject o)`
- `obj.StereoUpdate ()` - Update system if needed due to stereo rendering.
- `int = obj.GetPixelData (int x, int y, int x2, int y2, int front, vtkUnsignedCharArray data)` - Set/Get the pixel data of an image, transmitted as RGBRGB...
- `int = obj.SetPixelData (int x, int y, int x2, int y2, string data, int front)` - Set/Get the pixel data of an image, transmitted as RGBRGB...
- `int = obj.SetPixelData (int x, int y, int x2, int y2, vtkUnsignedCharArray data, int front)` - Set/Get the pixel data of an image, transmitted as RGBRGB...
- `int = obj.GetRGBAPixelData (int x, int y, int x2, int y2, int front, vtkFloatArray data)` - Set/Get the pixel data of an image, transmitted as RGBARGBA...
- `int = obj.SetRGBAPixelData (int x, int y, int x2, int y2, float data, int front, int blend)` - Set/Get the pixel data of an image, transmitted as RGBARGBA...
- `int = obj.SetRGBAPixelData (int x, int y, int x2, int y2, vtkFloatArray data, int front, int blend)` - Set/Get the pixel data of an image, transmitted as RGBARGBA...

- `obj.ReleaseRGBAPixelData (float data)` - Set/Get the pixel data of an image, transmitted as RGBARGBA...
- `int = obj.GetRGBACharPixelData (int x, int y, int x2, int y2, int front, vtkUnsignedCharArray data)` - Set/Get the pixel data of an image, transmitted as RGBARGBA...
- `int = obj.SetRGBACharPixelData (int x, int y, int x2, int y2, string data, int front, int blend)` - Set/Get the pixel data of an image, transmitted as RGBARGBA...
- `int = obj.SetRGBACharPixelData (int x, int y, int x2, int y2, vtkUnsignedCharArray data, int front,` - Set/Get the pixel data of an image, transmitted as RGBARGBA...
- `int = obj.GetZbufferData (int x1, int y1, int x2, int y2, float z)` - Set/Get the zbuffer data from an image
- `int = obj.GetZbufferData (int x1, int y1, int x2, int y2, vtkFloatArray z)` - Set/Get the zbuffer data from an image
- `int = obj.SetZbufferData (int x1, int y1, int x2, int y2, float buffer)` - Set/Get the zbuffer data from an image
- `int = obj.SetZbufferData (int x1, int y1, int x2, int y2, vtkFloatArray buffer)` - Set/Get the zbuffer data from an image
- `int = obj.GetDepthBufferSize ()` - Get the size of the depth buffer.
- `int = obj.GetColorBufferSizes (int rgba)` - Get the size of the color buffer. Returns 0 if not able to determine otherwise sets R G B and A into buffer.
- `obj.OpenGLInit ()` - Initialize OpenGL for this window.
- `int = obj.GetBackLeftBuffer ()` - Return the OpenGL name of the back left buffer. It is GL_BACK_LEFT if GL is bound to the window-system-provided framebuffer. It is vtkgl::COLOR_ATTACHMENT0_EXT if GL is bound to an application-created framebuffer object (GPU-based offscreen rendering) It is used by vtkOpenGLCamera.
- `int = obj.GetBackRightBuffer ()` - Return the OpenGL name of the back right buffer. It is GL_BACK_RIGHT if GL is bound to the window-system-provided framebuffer. It is vtkgl::COLOR_ATTACHMENT0_E if GL is bound to an application-created framebuffer object (GPU-based offscreen rendering) It is used by vtkOpenGLCamera.
- `int = obj.GetFrontLeftBuffer ()` - Return the OpenGL name of the front left buffer. It is GL_FRONT_LEFT if GL is bound to the window-system-provided framebuffer. It is vtkgl::COLOR_ATTACHMENT0_EXT if GL is bound to an application-created framebuffer object (GPU-based offscreen rendering) It is used by vtkOpenGLCamera.
- `int = obj.GetFrontRightBuffer ()` - Return the OpenGL name of the front right buffer. It is GL_FRONT_RIGHT if GL is bound to the window-system-provided framebuffer. It is vtkgl::COLOR_ATTACHMENT0_ if GL is bound to an application-created framebuffer object (GPU-based offscreen rendering) It is used by vtkOpenGLCamera.
- `int = obj.GetBackBuffer ()` - Return the OpenGL name of the back left buffer. It is GL_BACK if GL is bound to the window-system-provided framebuffer. It is vtkgl::COLOR_ATTACHMENT0_EXT if GL is bound to an application-created framebuffer object (GPU-based offscreen rendering) It is used by vtkOpenGLCamera.
- `int = obj.GetFrontBuffer ()` - Return the OpenGL name of the front left buffer. It is GL_FRONT if GL is bound to the window-system-provided framebuffer. It is vtkgl::COLOR_ATTACHMENT0_EXT if GL is bound to an application-created framebuffer object (GPU-based offscreen rendering) It is used by vtkOpenGLCamera.

- `obj.CheckGraphicError ()` - Update graphic error status, regardless of ReportGraphicErrors flag. It means this method can be used in any context and is not restricted to debug mode.
- `int = obj.HasGraphicError ()` - Return the last graphic error status. Initial value is false.
- `string = obj.GetLastGraphicErrorString ()` - Return a string matching the last graphic error status.
- `vtkOpenGLExtensionManager = obj.GetExtensionManager ()` - Returns the extension manager. A new one will be created if one hasn't already been set up.
- `vtkOpenGLHardwareSupport = obj.GetHardwareSupport ()` - Returns an Hardware Support object. A new one will be created if one hasn't already been set up.
- `obj.WaitForCompletion ()` - Block the thread until the actual rendering is finished(). Useful for measurement only.

39.126 vtkOpenGLRepresentationPainter

39.126.1 Usage

This is OpenGL implementation of a painter handling representation i.e. Points, Wireframe, Surface.

To create an instance of class vtkOpenGLRepresentationPainter, simply invoke its constructor as follows

```
obj = vtkOpenGLRepresentationPainter
```

39.126.2 Methods

The class vtkOpenGLRepresentationPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLRepresentationPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLRepresentationPainter = obj.NewInstance ()`
- `vtkOpenGLRepresentationPainter = obj.SafeDownCast (vtkObject o)`
- `double = obj.GetTimeToDraw ()`

39.127 vtkOpenGLScalarsToColorsPainter

39.127.1 Usage

vtkOpenGLScalarsToColorsPainter is a concrete subclass of vtkScalarsToColorsPainter which uses OpenGL for color mapping.

To create an instance of class vtkOpenGLScalarsToColorsPainter, simply invoke its constructor as follows

```
obj = vtkOpenGLScalarsToColorsPainter
```

39.127.2 Methods

The class vtkOpenGLScalarsToColorsPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLScalarsToColorsPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLScalarsToColorsPainter = obj.NewInstance ()`
- `vtkOpenGLScalarsToColorsPainter = obj.SafeDownCast (vtkObject o)`
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this mapper. The parameter window could be used to determine which graphic resources to release.
- `int = obj.GetPremultiplyColorsWithAlpha (vtkActor actor)`

39.128 vtkOpenGLTexture

39.128.1 Usage

vtkOpenGLTexture is a concrete implementation of the abstract class vtkTexture. vtkOpenGLTexture interfaces to the OpenGL rendering library.

To create an instance of class vtkOpenGLTexture, simply invoke its constructor as follows

```
obj = vtkOpenGLTexture
```

39.128.2 Methods

The class vtkOpenGLTexture has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLTexture class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLTexture = obj.NewInstance ()`
- `vtkOpenGLTexture = obj.SafeDownCast (vtkObject o)`
- `obj.Load (vtkRenderer ren)` - Implement base class method.
- `obj.PostRender (vtkRenderer ren)`
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this texture. The parameter window could be used to determine which graphic resources to release. Using the same texture object in multiple render windows is NOT currently supported.
- `long = obj.GetIndex ()` - Get the openGL texture name to which this texture is bound. This is available only if GL version ¿= 1.1

39.129 vtkOverlayPass

39.129.1 Usage

vtkOverlayPass renders the overlay geometry of all the props that have the keys contained in vtkRenderState.

This pass expects an initialized depth buffer and color buffer. Initialized buffers means they have been cleared with farest z-value and background color/gradient/transparent color.

To create an instance of class vtkOverlayPass, simply invoke its constructor as follows

```
obj = vtkOverlayPass
```

39.129.2 Methods

The class vtkOverlayPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOverlayPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOverlayPass = obj.NewInstance ()`
- `vtkOverlayPass = obj.SafeDownCast (vtkObject o)`

39.130 vtkPainter

39.130.1 Usage

This defines the interface for a Painter. Painters are helpers used by Mapper to perform the rendering. The mapper sets up a chain of painters and passes the render request to the painter. Every painter may have a delegate painter to which the render request is forwarded. The Painter may modify the request or data before passing it to the delegate painter. All the information to control the rendering must be passed to the painter using the vtkInformation object. A concrete painter may read special keys from the vtkInformation object and affect the rendering.

To create an instance of class vtkPainter, simply invoke its constructor as follows

```
obj = vtkPainter
```

39.130.2 Methods

The class vtkPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPainter = obj.NewInstance ()`
- `vtkPainter = obj.SafeDownCast (vtkObject o)`
- `vtkInformation = obj.GetInformation ()` - Get/Set the information object associated with this painter.
- `obj.SetInformation (vtkInformation )` - Get/Set the information object associated with this painter.

- `vtkPainter = obj.GetDelegatePainter ()` - Set/Get the painter to which this painter should propagare its draw calls.
- `obj.SetDelegatePainter (vtkPainter )` - Set/Get the painter to which this painter should propagare its draw calls.
- `obj.Register (vtkObjectBase o)` - Take part in garbage collection.
- `obj.UnRegister (vtkObjectBase o)` - Take part in garbage collection.
- `obj.Render (vtkRenderer renderer, vtkActor actor, long typeflags, bool forceCompileOnly)` - Generates rendering primitives of appropriate type(s). Multiple types of primitives can be requested by or-ring the primitive flags. Default implementation calls UpdateDelegatePainter() to update the deletagate painter and then calls RenderInternal(). forceCompileOnly is passed to the display list painters.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this painter. The parameter window could be used to determine which graphic resources to release. The call is propagated to the delegate painter, if any.
- `obj.SetProgress (double )` - Set/Get the execution progress of a process object.
- `double = obj.GetProgressMinValue ()` - Set/Get the execution progress of a process object.
- `double = obj.GetProgressMaxValue ()` - Set/Get the execution progress of a process object.
- `double = obj.GetProgress ()` - Set/Get the execution progress of a process object.
- `double = obj.GetTimeToDraw ()` - Get the time required to draw the geometry last time it was rendered. Default implementation adds the current TimeToDraw with that of the delegate painter.
- `obj.UpdateBounds (double bounds[6])` - Expand or shrink the estimated bounds of the object based on the geometric transformations performed in the painter. If the painter does not modify the geometry, the bounds are passed through.
- `obj.SetInput (vtkDataObject )` - Set the data object to paint. Currently we only support one data object per painter chain.
- `vtkDataObject = obj.GetInput ()` - Set the data object to paint. Currently we only support one data object per painter chain.
- `vtkDataObject = obj.GetOutput ()`

39.131 vtkPainterDeviceAdapter

39.131.1 Usage

This class is an adapter between a vtkPainter and a rendering device (such as an OpenGL machine). Having an abstract adapter allows vtkPainters to be re-used for any rendering system.

Although VTK really only uses OpenGL right now, there are reasons to swap out the rendering functions. Sometimes MESA with mangled names is used. Also, different shader extensions use different functions. Furthermore, Cg also has its own interface.

The interface for this class should be familier to anyone experienced with OpenGL.

To create an instance of class vtkPainterDeviceAdapter, simply invoke its constructor as follows

```
obj = vtkPainterDeviceAdapter
```

39.131.2 Methods

The class vtkPainterDeviceAdapter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPainterDeviceAdapter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPainterDeviceAdapter = obj.NewInstance ()`
- `vtkPainterDeviceAdapter = obj.SafeDownCast (vtkObject o)`
- `obj.BeginPrimitive (int mode)` - Signals the start of sending a primitive to the graphics card. The mode is one of VTK_VERTEX, VTK_POLY_VERTEX, VTK_LINE, VTK_POLY_LINE, VTK_TRIANGLE, VTK_TRIANGLE_STRIP, VTK_POLYGON, or VTK_QUAD. The primitive is defined by the attributes sent between the calls to BeginPrimitive and EndPrimitive. You do not need to call EndPrimitive/BeginPrimitive between primitives that have a constant number of points (i.e. VTK_VERTEX, VTK_LINE, VTK_TRIANGLE, and VTK_QUAD).
- `obj.EndPrimitive ()` - Signals the end of sending a primitive to the graphics card.
- `int = obj.IsAttributesSupported (int attribute)` - Returns if the given attribute type is supported by the device. Returns 1 is supported, 0 otherwise.
- `obj.SetAttributePointer (int index, vtkDataArray attributeArray)` - Sets an array of attributes. This allows you to send all the data for a particular attribute with one call, thus greatly reducing function call overhead. Once set, the array is enabled with EnableAttributeArray, and the data is sent with a call to DrawArrays DrawElements.
- `obj.EnableAttributeArray (int index)` - Enable/disable the attribute array set with SetAttributePointer.
- `obj.DisableAttributeArray (int index)` - Enable/disable the attribute array set with SetAttributePointer.
- `obj.DrawArrays (int mode, vtkIdType first, vtkIdType count)` - Send a section of the enabled attribute pointers to the graphics card to define a primitive. The mode is one of VTK_VERTEX, VTK_POLY_VERTEX, VTK_LINE, VTK_POLY_LINE, VTK_TRIANGLE, VTK_TRIANGLE_STRIP, VTK_POLYGON, or VTK_QUAD. It identifies which type of primitive the attribute data is defining. The parameters first and count identify what part of the attribute arrays define the given primitive. If mode is a primitive that has a constant number of points (i.e. VTK_VERTEX, VTK_LINE, VTK_TRIANGLE, and VTK_QUAD), you may draw multiple primitives with one call to DrawArrays.
- `int = obj.Compatible (vtkRenderer renderer)` - Returns true if this device adapter is compatable with the given vtkRenderer.
- `obj.MakeLighting (int mode)` - Turns lighting on and off.
- `int = obj.QueryLighting ()` - Returns current lighting setting.
- `obj.MakeMultisampling (int mode)` - Turns antialiasing on and off.
- `int = obj.QueryMultisampling ()` - Returns current antialiasing setting.
- `obj.MakeBlending (int mode)` - Turns blending on and off.
- `int = obj.QueryBlending ()` - Returns current blending setting.

- `obj.MakeVertexEmphasis (bool mode)` - Turns emphasis of vertices on or off for vertex selection.
- `obj.MakeVertexEmphasisWithStencilCheck (int )` - @deprecated
- `obj.Stencil (int on)` - Control use of the stencil buffer (for vertex selection).
- `obj.WriteStencil (vtkIdType value)` - Control use of the stencil buffer (for vertex selection).
- `obj.TestStencil (vtkIdType value)` - Control use of the stencil buffer (for vertex selection).

39.132 vtkPainterPolyDataMapper

39.132.1 Usage

PolyDataMapper that uses painters to do the actual rendering. .SECTION Thanks Support for generic vertex attributes in VTK was contributed in collaboration with Stephane Ploix at EDF.

To create an instance of class vtkPainterPolyDataMapper, simply invoke its constructor as follows

```
obj = vtkPainterPolyDataMapper
```

39.132.2 Methods

The class vtkPainterPolyDataMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPainterPolyDataMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPainterPolyDataMapper = obj.NewInstance ()`
- `vtkPainterPolyDataMapper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderPiece (vtkRenderer ren, vtkActor act)` - Implemented by sub classes. Actual rendering is done here.
- `vtkPainter = obj.GetPainter ()` - Get/Set the painter used to do the actual rendering. By default, vtkDefaultPainter is used to build the rendering painter chain for color mapping/clipping etc. followed by a vtkChooserPainter which renders the primitives.
- `obj.SetPainter (vtkPainter )` - Get/Set the painter used to do the actual rendering. By default, vtkDefaultPainter is used to build the rendering painter chain for color mapping/clipping etc. followed by a vtkChooserPainter which renders the primitives.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this mapper. The parameter window could be used to determine which graphic resources to release. Merely propagates the call to the painter.
- `obj.GetBounds (double bounds[6])` - Re-implement the superclass GetBounds method.
- `double = obj.GetBounds ()` - Re-implement the superclass GetBounds method.
- `obj.MapDataArrayToVertexAttribute (string vertexAttributeName, string dataArrayName, int fieldAssoc` - Select a data array from the point/cell data and map it to a generic vertex attribute. vertexAttributeName is the name of the vertex attribute. dataArrayName is the name of the data array. fieldAssociation indicates when the data array is a point data array or cell data array (vtkDataObject::FIELD_ASSOCIATION_POINTS or (vtkDataObject::FIELD_ASSOCIATION_CELLS). componentno indicates which component from the data array must be passed as the attribute. If -1, then all components are passed.

- `obj.MapDataArrayToMultiTextureAttribute (int unit, string dataArrayName, int fieldAssociation, int`
- `obj.RemoveVertexAttributeMapping (string vertexAttributeName)` - Remove a vertex attribute mapping.
- `obj.RemoveAllVertexAttributeMappings ()` - Remove all vertex attributes.
- `vtkPainter = obj.GetSelectionPainter ()` - Get/Set the painter used when rendering the selection pass.
- `obj.SetSelectionPainter (vtkPainter )` - Get/Set the painter used when rendering the selection pass.
- `bool = obj.GetSupportsSelection ()`

39.133 vtkParallelCoordinatesActor

39.133.1 Usage

vtkParallelCoordinatesActor generates a parallel coordinates plot from an input field (i.e., vtkDataObject). Parallel coordinates represent N-dimensional data by using a set of N parallel axes (not orthogonal like the usual x-y-z Cartesian axes). Each N-dimensional point is plotted as a polyline, were each of the N components of the point lie on one of the N axes, and the components are connected by straight lines.

To use this class, you must specify an input data object. You'll probably also want to specify the position of the plot be setting the Position and Position2 instance variables, which define a rectangle in which the plot lies. Another important parameter is the IndependentVariables ivar, which tells the instance how to interpret the field data (independent variables as the rows or columns of the field). There are also many other instance variables that control the look of the plot includes its title, attributes, number of ticks on the axes, etc.

Set the text property/attributes of the title and the labels through the vtkTextProperty objects associated to this actor.

To create an instance of class vtkParallelCoordinatesActor, simply invoke its constructor as follows

```
obj = vtkParallelCoordinatesActor
```

39.133.2 Methods

The class vtkParallelCoordinatesActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParallelCoordinatesActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParallelCoordinatesActor = obj.NewInstance ()`
- `vtkParallelCoordinatesActor = obj.SafeDownCast (vtkObject o)`
- `obj.SetIndependentVariables (int )` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `int = obj.GetIndependentVariablesMinValue ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.

- `int = obj.GetIndependentVariablesMaxValue ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `int = obj.GetIndependentVariables ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `obj.SetIndependentVariablesToColumns ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `obj.SetIndependentVariablesToRows ()` - Specify whether to use the rows or columns as independent variables. If columns, then each row represents a separate point. If rows, then each column represents a separate point.
- `obj.SetTitle (string )` - Set/Get the title of the parallel coordinates plot.
- `string = obj.GetTitle ()` - Set/Get the title of the parallel coordinates plot.
- `obj.SetNumberOfLabels (int )` - Set/Get the number of annotation labels to show along each axis. This values is a suggestion: the number of labels may vary depending on the particulars of the data.
- `int = obj.GetNumberOfLabelsMinValue ()` - Set/Get the number of annotation labels to show along each axis. This values is a suggestion: the number of labels may vary depending on the particulars of the data.
- `int = obj.GetNumberOfLabelsMaxValue ()` - Set/Get the number of annotation labels to show along each axis. This values is a suggestion: the number of labels may vary depending on the particulars of the data.
- `int = obj.GetNumberOfLabels ()` - Set/Get the number of annotation labels to show along each axis. This values is a suggestion: the number of labels may vary depending on the particulars of the data.
- `obj.SetLabelFormat (string )` - Set/Get the format with which to print the labels on the axes.
- `string = obj.GetLabelFormat ()` - Set/Get the format with which to print the labels on the axes.
- `obj.SetTitleTextProperty (vtkTextProperty p)` - Set/Get the title text property.
- `vtkTextProperty = obj.GetTitleTextProperty ()` - Set/Get the title text property.
- `obj.SetLabelTextProperty (vtkTextProperty p)` - Set/Get the labels text property.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - Set/Get the labels text property.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Draw the parallel coordinates plot.
- `int = obj.RenderOverlay (vtkViewport )` - Draw the parallel coordinates plot.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Does this prop have some translucent polygonal geometry?
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.SetInput (vtkDataObject )` - Set the input to the parallel coordinates actor.
- `vtkDataObject = obj.GetInput ()` - Remove a dataset from the list of data to append.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.

39.134 vtkParallelCoordinatesInteractorStyle

39.134.1 Usage

vtkParallelCoordinatesInteractorStyle allows the user to interactively manipulate (rotate, pan, zoomm etc.) the camera. Several events are overloaded from its superclass vtkParallelCoordinatesInteractorStyle, hence the mouse bindings are different. (The bindings keep the camera's view plane normal perpendicular to the x-y plane.) In summary the mouse events are as follows: + Left Mouse button triggers window level events + CTRL Left Mouse spins the camera around its view plane normal + SHIFT Left Mouse pans the camera + CTRL SHIFT Left Mouse dollys (a positional zoom) the camera + Middle mouse button pans the camera + Right mouse button dollys the camera. + SHIFT Right Mouse triggers pick events

Note that the renderer's actors are not moved; instead the camera is moved.

To create an instance of class vtkParallelCoordinatesInteractorStyle, simply invoke its constructor as follows

```
obj = vtkParallelCoordinatesInteractorStyle
```

39.134.2 Methods

The class vtkParallelCoordinatesInteractorStyle has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParallelCoordinatesInteractorStyle class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParallelCoordinatesInteractorStyle = obj.NewInstance ()`
- `vtkParallelCoordinatesInteractorStyle = obj.SafeDownCast (vtkObject o)`
- `int = obj. GetCursorStartPosition ()` - Get the cursor positions in pixel coords
- `int = obj. GetCursorCurrentPosition ()` - Get the cursor positions in pixel coords
- `int = obj. GetCursorLastPosition ()` - Get the cursor positions in pixel coords
- `obj.GetCursorStartPosition (vtkViewport viewport, double pos[2])` - Get the cursor positions in a given coordinate system
- `obj.GetCursorCurrentPosition (vtkViewport viewport, double pos[2])` - Get the cursor positions in a given coordinate system
- `obj.GetCursorLastPosition (vtkViewport viewport, double pos[2])` - Get the cursor positions in a given coordinate system
- `obj.OnMouseMove ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeftButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnMiddleButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.

- `obj.OnRightButtonDown ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnRightButtonUp ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.OnLeave ()` - Event bindings controlling the effects of pressing mouse buttons or moving the mouse.
- `obj.StartInspect (int x, int y)`
- `obj.Inspect (int x, int y)`
- `obj.EndInspect ()`
- `obj.StartZoom ()`
- `obj.Zoom ()`
- `obj.EndZoom ()`
- `obj.StartPan ()`
- `obj.Pan ()`
- `obj.EndPan ()`
- `obj.OnChar ()` - Override the "fly-to" (f keypress) for images.

39.135 vtkPicker

39.135.1 Usage

vtkPicker is used to select instances of vtkProp3D by shooting a ray into a graphics window and intersecting with the actor's bounding box. The ray is defined from a point defined in window (or pixel) coordinates, and a point located from the camera's position.

vtkPicker may return more than one vtkProp3D, since more than one bounding box may be intersected. vtkPicker returns an unsorted list of props that were hit, and a list of the corresponding world points of the hits. For the vtkProp3D that is closest to the camera, vtkPicker returns the pick coordinates in world and untransformed mapper space, the prop itself, the data set, and the mapper. For vtkPicker the closest prop is the one whose center point (i.e., center of bounding box) projected on the view ray is closest to the camera. Subclasses of vtkPicker use other methods for computing the pick point.

To create an instance of class vtkPicker, simply invoke its constructor as follows

```
obj = vtkPicker
```

39.135.2 Methods

The class vtkPicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPicker = obj.NewInstance ()`
- `vtkPicker = obj.SafeDownCast (vtkObject o)`

- `obj.SetTolerance (double )` - Specify tolerance for performing pick operation. Tolerance is specified as fraction of rendering window size. (Rendering window size is measured across diagonal.)
- `double = obj.GetTolerance ()` - Specify tolerance for performing pick operation. Tolerance is specified as fraction of rendering window size. (Rendering window size is measured across diagonal.)
- `double = obj. GetMapperPosition ()` - Return position in mapper (i.e., non-transformed) coordinates of pick point.
- `vtkAbstractMapper3D = obj.GetMapper ()` - Return mapper that was picked (if any).
- `vtkDataSet = obj.GetDataSet ()` - Get a pointer to the dataset that was picked (if any). If nothing was picked then NULL is returned.
- `vtkProp3DCollection = obj.GetProp3Ds ()` - Return a collection of all the prop 3D's that were intersected by the pick ray. This collection is not sorted.
- `vtkActorCollection = obj.GetActors ()` - Return a collection of all the actors that were intersected. This collection is not sorted. (This is a convenience method to maintain backward compatibility.)
- `vtkPoints = obj.GetPickedPositions ()` - Return a list of the points the the actors returned by GetProp3Ds were intersected at. The order of this list will match the order of GetProp3Ds.
- `int = obj.Pick (double selectionX, double selectionY, double selectionZ, vtkRenderer renderer)` - Perform pick operation with selection point provided. Normally the first two values for the selection point are x-y pixel coordinate, and the third value is =0. Return non-zero if something was successfully picked.
- `int = obj.Pick (double selectionPt[3], vtkRenderer ren)` - Perform pick operation with selection point provided. Normally the first two values for the selection point are x-y pixel coordinate, and the third value is =0. Return non-zero if something was successfully picked.

39.136 vtkPixelBufferObject

39.136.1 Usage

Provides low-level access to GPU memory. Used to pass raw data to GPU. The data is uploaded into a pixel buffer.

To create an instance of class vtkPixelBufferObject, simply invoke its constructor as follows

```
obj = vtkPixelBufferObject
```

39.136.2 Methods

The class vtkPixelBufferObject has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPixelBufferObject class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPixelBufferObject = obj.NewInstance ()`
- `vtkPixelBufferObject = obj.SafeDownCast (vtkObject o)`

- `obj.SetContext (vtkRenderWindow context)` - Get/Set the context. Context must be a vtkOpenGLRenderWindow. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions.
- `vtkRenderWindow = obj.GetContext ()` - Get/Set the context. Context must be a vtkOpenGLRenderWindow. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions.
- `int = obj.GetUsage ()` - Usage is a performance hint. Valid values are: - StreamDraw specified once by A, used few times S - StreamRead specified once by R, queried a few times by A - StreamCopy specified once by R, used a few times S - StaticDraw specified once by A, used many times S - StaticRead specificed once by R, queried many times by A - StaticCopy specified once by R, used many times S - DynamicDraw respecified repeatedly by A, used many times S - DynamicRead respecified repeatedly by R, queried many times by A - DynamicCopy respecified repeatedly by R, used many times S A: the application S: as the source for GL drawing and image specification commands. R: reading data from the GL Initial value is StaticDraw, as in OpenGL spec.
- `obj.SetUsage (int )` - Usage is a performance hint. Valid values are: - StreamDraw specified once by A, used few times S - StreamRead specified once by R, queried a few times by A - StreamCopy specified once by R, used a few times S - StaticDraw specified once by A, used many times S - StaticRead specificed once by R, queried many times by A - StaticCopy specified once by R, used many times S - DynamicDraw respecified repeatedly by A, used many times S - DynamicRead respecified repeatedly by R, queried many times by A - DynamicCopy respecified repeatedly by R, used many times S A: the application S: as the source for GL drawing and image specification commands. R: reading data from the GL Initial value is StaticDraw, as in OpenGL spec.
- `int = obj.GetType ()` - Get the type with which the data is loaded into the GPU. eg. VTK_FLOAT for float32, VTK_CHAR for byte, VTK_UNSIGNED_CHAR for unsigned byte etc.
- `int = obj.GetSize ()` - Get the size of the data loaded into the GPU. Size is in the number of elements of the uploaded Type.
- `int = obj.GetHandle ()` - Get the openGL buffer handle.
- `obj.BindToPackedBuffer ()`
- `obj.BindToUnPackedBuffer ()` - Inactivate the buffer.
- `obj.UnBind ()` - Inactivate the buffer.

39.137 vtkPointPicker

39.137.1 Usage

To create an instance of class vtkPointPicker, simply invoke its constructor as follows

```
obj = vtkPointPicker
```

39.137.2 Methods

The class vtkPointPicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointPicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkPointPicker = obj.NewInstance ()`
- `vtkPointPicker = obj.SafeDownCast (vtkObject o)`
- `vtkIdType = obj.GetPointId ()` - Get the id of the picked point. If PointId = -1, nothing was picked.

39.138 vtkPointSetToLabelHierarchy

39.138.1 Usage

Every point in the input vtkPoints object is taken to be an anchor point for a label. Statistics on the input points are used to subdivide an octree referencing the points until the points each octree node contains have a variance close to the node size and a limited population (¡ 100).

To create an instance of class vtkPointSetToLabelHierarchy, simply invoke its constructor as follows

```
obj = vtkPointSetToLabelHierarchy
```

39.138.2 Methods

The class vtkPointSetToLabelHierarchy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointSetToLabelHierarchy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointSetToLabelHierarchy = obj.NewInstance ()`
- `vtkPointSetToLabelHierarchy = obj.SafeDownCast (vtkObject o)`
- `obj.SetTargetLabelCount (int )` - Set/get the "ideal" number of labels to associate with each node in the output hierarchy.
- `int = obj.GetTargetLabelCount ()` - Set/get the "ideal" number of labels to associate with each node in the output hierarchy.
- `obj.SetMaximumDepth (int )` - Set/get the maximum tree depth in the output hierarchy.
- `int = obj.GetMaximumDepth ()` - Set/get the maximum tree depth in the output hierarchy.
- `obj.SetUseUnicodeStrings (bool )` - Whether to use unicode strings.
- `bool = obj.GetUseUnicodeStrings ()` - Whether to use unicode strings.
- `obj.UseUnicodeStringsOn ()` - Whether to use unicode strings.
- `obj.UseUnicodeStringsOff ()` - Whether to use unicode strings.
- `obj.SetLabelArrayName (string name)` - Set/get the label array name.
- `string = obj.GetLabelArrayName ()` - Set/get the label array name.
- `obj.SetSizeArrayName (string name)` - Set/get the priority array name.
- `string = obj.GetSizeArrayName ()` - Set/get the priority array name.
- `obj.SetPriorityArrayName (string name)` - Set/get the priority array name.

- `string = obj.GetPriorityArrayName ()` - Set/get the priority array name.
- `obj.SetIconIndexArrayName (string name)` - Set/get the icon index array name.
- `string = obj.GetIconIndexArrayName ()` - Set/get the icon index array name.
- `obj.SetOrientationArrayName (string name)` - Set/get the text orientation array name.
- `string = obj.GetOrientationArrayName ()` - Set/get the text orientation array name.
- `obj.SetBoundedSizeArrayName (string name)` - Set/get the maximum text width (in world coordinates) array name.
- `string = obj.GetBoundedSizeArrayName ()` - Set/get the maximum text width (in world coordinates) array name.
- `obj.SetTextProperty (vtkTextProperty tprop)` - Set/get the text property assigned to the hierarchy.
- `vtkTextProperty = obj.GetTextProperty ()` - Set/get the text property assigned to the hierarchy.

39.139 vtkPointsPainter

39.139.1 Usage

This painter tries to paint points efficiently. Request to Render any other primitive are ignored and not passed to the delegate painter, if any. This painter cannot handle cell colors/normals. If they are present the request is passed on to the Delegate painter. If this class is able to render the primitive, the render request is not propagated to the delegate painter.

To create an instance of class vtkPointsPainter, simply invoke its constructor as follows

```
obj = vtkPointsPainter
```

39.139.2 Methods

The class vtkPointsPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointsPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointsPainter = obj.NewInstance ()`
- `vtkPointsPainter = obj.SafeDownCast (vtkObject o)`

39.140 vtkPolyDataMapper

39.140.1 Usage

vtkPolyDataMapper is a class that maps polygonal data (i.e., vtkPolyData) to graphics primitives. vtkPolyDataMapper serves as a superclass for device-specific poly data mappers, that actually do the mapping to the rendering/graphics hardware/software.

To create an instance of class vtkPolyDataMapper, simply invoke its constructor as follows

```
obj = vtkPolyDataMapper
```

39.140.2 Methods

The class vtkPolyDataMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataMapper = obj.NewInstance ()`
- `vtkPolyDataMapper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderPiece (vtkRenderer ren, vtkActor act)` - Implemented by sub classes. Actual rendering is done here.
- `obj.Render (vtkRenderer ren, vtkActor act)` - This calls RenderPiece (in a for loop is streaming is necessary).
- `obj.SetInput (vtkPolyData in)` - Specify the input data to map.
- `vtkPolyData = obj.GetInput ()` - Specify the input data to map.
- `obj.Update ()` - Update that sets the update piece first.
- `obj.SetPiece (int )` - If you want only a part of the data, specify by setting the piece.
- `int = obj.GetPiece ()` - If you want only a part of the data, specify by setting the piece.
- `obj.SetNumberOfPieces (int )` - If you want only a part of the data, specify by setting the piece.
- `int = obj.GetNumberOfPieces ()` - If you want only a part of the data, specify by setting the piece.
- `obj.SetNumberOfSubPieces (int )` - If you want only a part of the data, specify by setting the piece.
- `int = obj.GetNumberOfSubPieces ()` - If you want only a part of the data, specify by setting the piece.
- `obj.SetGhostLevel (int )` - Set the number of ghost cells to return.
- `int = obj.GetGhostLevel ()` - Set the number of ghost cells to return.
- `double = obj.GetBounds ()` - Return bounding box (array of six doubles) of data expressed as (xmin,xmax, ymin,ymax, zmin,zmax).
- `obj.GetBounds (double bounds[6])` - Return bounding box (array of six doubles) of data expressed as (xmin,xmax, ymin,ymax, zmin,zmax).
- `obj.ShallowCopy (vtkAbstractMapper m)` - Make a shallow copy of this mapper.
- `obj.MapDataArrayToVertexAttribute (string vertexAttributeName, string dataArrayName, int fieldAssoc` - Select a data array from the point/cell data and map it to a generic vertex attribute. vertexAttributeName is the name of the vertex attribute. dataArrayName is the name of the data array. fieldAssociation indicates when the data array is a point data array or cell data array (vtkDataObject::FIELD_ASSOCIATION_POINTS or (vtkDataObject::FIELD_ASSOCIATION_CELLS). componentno indicates which component from the data array must be passed as the attribute. If -1, then all components are passed.
- `obj.MapDataArrayToMultiTextureAttribute (int unit, string dataArrayName, int fieldAssociation, int`
- `obj.RemoveVertexAttributeMapping (string vertexAttributeName)` - Remove a vertex attribute mapping.
- `obj.RemoveAllVertexAttributeMappings ()` - Remove all vertex attributes.

39.141 vtkPolyDataMapper2D

39.141.1 Usage

vtkPolyDataMapper2D is a mapper that renders 3D polygonal data (vtkPolyData) onto the 2D image plane (i.e., the renderer's viewport). By default, the 3D data is transformed into 2D data by ignoring the z-coordinate of the 3D points in vtkPolyData, and taking the x-y values as local display values (i.e., pixel coordinates). Alternatively, you can provide a vtkCoordinate object that will transform the data into local display coordinates (use the vtkCoordinate::SetCoordinateSystem() methods to indicate which coordinate system you are transforming the data from).

To create an instance of class vtkPolyDataMapper2D, simply invoke its constructor as follows

```
obj = vtkPolyDataMapper2D
```

39.141.2 Methods

The class vtkPolyDataMapper2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataMapper2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataMapper2D = obj.NewInstance ()`
- `vtkPolyDataMapper2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkPolyData in)` - Set the input to the mapper.
- `vtkPolyData = obj.GetInput ()` - Set the input to the mapper.
- `obj.SetLookupTable (vtkScalarsToColors lut)` - Specify a lookup table for the mapper to use.
- `vtkScalarsToColors = obj.GetLookupTable ()` - Specify a lookup table for the mapper to use.
- `obj.CreateDefaultLookupTable ()` - Create default lookup table. Generally used to create one when none is available with the scalar data.
- `obj.SetScalarVisibility (int )` - Turn on/off flag to control whether scalar data is used to color objects.
- `int = obj.GetScalarVisibility ()` - Turn on/off flag to control whether scalar data is used to color objects.
- `obj.ScalarVisibilityOn ()` - Turn on/off flag to control whether scalar data is used to color objects.
- `obj.ScalarVisibilityOff ()` - Turn on/off flag to control whether scalar data is used to color objects.
- `obj.SetColorMode (int )` - Control how the scalar data is mapped to colors. By default (ColorModeToDefault), unsigned char scalars are treated as colors, and NOT mapped through the lookup table, while everything else is. Setting ColorModeToMapScalars means that all scalar data will be mapped through the lookup table. (Note that for multi-component scalars, the particular component to use for mapping can be specified using the ColorByArrayComponent() method.)
- `int = obj.GetColorMode ()` - Control how the scalar data is mapped to colors. By default (ColorModeToDefault), unsigned char scalars are treated as colors, and NOT mapped through the lookup table, while everything else is. Setting ColorModeToMapScalars means that all scalar data will be mapped through the lookup table. (Note that for multi-component scalars, the particular component to use for mapping can be specified using the ColorByArrayComponent() method.)

- `obj.SetColorModeToDefault ()` - Control how the scalar data is mapped to colors. By default (ColorModeToDefault), unsigned char scalars are treated as colors, and NOT mapped through the lookup table, while everything else is. Setting ColorModeToMapScalars means that all scalar data will be mapped through the lookup table. (Note that for multi-component scalars, the particular component to use for mapping can be specified using the ColorByArrayComponent() method.)

- `obj.SetColorModeToMapScalars ()` - Control how the scalar data is mapped to colors. By default (ColorModeToDefault), unsigned char scalars are treated as colors, and NOT mapped through the lookup table, while everything else is. Setting ColorModeToMapScalars means that all scalar data will be mapped through the lookup table. (Note that for multi-component scalars, the particular component to use for mapping can be specified using the ColorByArrayComponent() method.)

- `string = obj.GetColorModeAsString ()` - Return the method of coloring scalar data.

- `obj.SetUseLookupTableScalarRange (int )` - Control whether the mapper sets the lookuptable range based on its own ScalarRange, or whether it will use the LookupTable ScalarRange regardless of it's own setting. By default the Mapper is allowed to set the LookupTable range, but users who are sharing LookupTables between mappers/actors will probably wish to force the mapper to use the LookupTable unchanged.

- `int = obj.GetUseLookupTableScalarRange ()` - Control whether the mapper sets the lookuptable range based on its own ScalarRange, or whether it will use the LookupTable ScalarRange regardless of it's own setting. By default the Mapper is allowed to set the LookupTable range, but users who are sharing LookupTables between mappers/actors will probably wish to force the mapper to use the LookupTable unchanged.

- `obj.UseLookupTableScalarRangeOn ()` - Control whether the mapper sets the lookuptable range based on its own ScalarRange, or whether it will use the LookupTable ScalarRange regardless of it's own setting. By default the Mapper is allowed to set the LookupTable range, but users who are sharing LookupTables between mappers/actors will probably wish to force the mapper to use the LookupTable unchanged.

- `obj.UseLookupTableScalarRangeOff ()` - Control whether the mapper sets the lookuptable range based on its own ScalarRange, or whether it will use the LookupTable ScalarRange regardless of it's own setting. By default the Mapper is allowed to set the LookupTable range, but users who are sharing LookupTables between mappers/actors will probably wish to force the mapper to use the LookupTable unchanged.

- `obj.SetScalarRange (double , double )` - Specify range in terms of scalar minimum and maximum (smin,smax). These values are used to map scalars into lookup table. Has no effect when UseLookupTableScalarRange is true.

- `obj.SetScalarRange (double  a[2])` - Specify range in terms of scalar minimum and maximum (smin,smax). These values are used to map scalars into lookup table. Has no effect when UseLookupTableScalarRange is true.

- `double = obj. GetScalarRange ()` - Specify range in terms of scalar minimum and maximum (smin,smax). These values are used to map scalars into lookup table. Has no effect when UseLookupTableScalarRange is true.

- `obj.SetScalarMode (int )` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call ColorByArrayComponent before you call GetColors.

- `int = obj.GetScalarMode ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call ColorByArrayComponent before you call GetColors.

- `obj.SetScalarModeToDefault ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call ColorByArrayComponent before you call GetColors.

- `obj.SetScalarModeToUsePointData ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call ColorByArrayComponent before you call GetColors.

- `obj.SetScalarModeToUseCellData ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call ColorByArrayComponent before you call GetColors.

- `obj.SetScalarModeToUsePointFieldData ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call ColorByArrayComponent before you call GetColors.

- `obj.SetScalarModeToUseCellFieldData ()` - Control how the filter works with scalar point data and cell attribute data. By default (ScalarModeToDefault), the filter will use point data, and if no point data is available, then cell data is used. Alternatively you can explicitly set the filter to use point data (ScalarModeToUsePointData) or cell data (ScalarModeToUseCellData). You can also choose to get the scalars from an array in point field data (ScalarModeToUsePointFieldData) or cell field data (ScalarModeToUseCellFieldData). If scalars are coming from a field data array, you must call ColorByArrayComponent before you call GetColors.

- `obj.ColorByArrayComponent (int arrayNum, int component)` - Choose which component of which field data array to color by.

- `obj.ColorByArrayComponent (string arrayName, int component)` - Choose which component of which field data array to color by.

- `string = obj.GetArrayName ()` - Get the array name or number and component to color by.

- `int = obj.GetArrayId ()` - Get the array name or number and component to color by.

- `int = obj.GetArrayAccessMode ()` - Get the array name or number and component to color by.
- `int = obj.GetArrayComponent ()` - Overload standard modified time function. If lookup table is modified, then this object is modified as well.
- `long = obj.GetMTime ()` - Overload standard modified time function. If lookup table is modified, then this object is modified as well.
- `obj.SetTransformCoordinate (vtkCoordinate )` - Specify a vtkCoordinate object to be used to transform the vtkPolyData point coordinates. By default (no vtkCoordinate specified), the point coordinates are taken as local display coordinates.
- `vtkCoordinate = obj.GetTransformCoordinate ()` - Specify a vtkCoordinate object to be used to transform the vtkPolyData point coordinates. By default (no vtkCoordinate specified), the point coordinates are taken as local display coordinates.
- `vtkUnsignedCharArray = obj.MapScalars (double alpha)` - Map the scalars (if there are any scalars and ScalarVisibility is on) through the lookup table, returning an unsigned char RGBA array. This is typically done as part of the rendering process. The alpha parameter allows the blending of the scalars with an additional alpha (typically which comes from a vtkActor, etc.)
- `obj.ShallowCopy (vtkAbstractMapper m)` - Make a shallow copy of this mapper.

39.142 vtkPolyDataPainter

39.142.1 Usage

vtkPolyDataPainter encapsulates a method of drawing poly data. This is a subset of what a mapper does. The painter does no maintenance of the rendering state (camera, lights, etc.). It is solely responsible for issuing rendering commands that build graphics primitives.

To simplify coding, an implementation of vtkPolyDataPainter is allowed to support only certain types of poly data or certain types of primitives.

To create an instance of class vtkPolyDataPainter, simply invoke its constructor as follows

```
obj = vtkPolyDataPainter
```

39.142.2 Methods

The class vtkPolyDataPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolyDataPainter = obj.NewInstance ()`
- `vtkPolyDataPainter = obj.SafeDownCast (vtkObject o)`
- `vtkPolyData = obj.GetInputAsPolyData ()` - Get/set the poly data to render.
- `vtkPolyData = obj.GetOutputAsPolyData ()` - Get the output polydata from this Painter. The default implementation forwards the input polydata as the output.
- `obj.Render (vtkRenderer renderer, vtkActor actor, long typeflags, bool forceCompileOnly)` - Overridden to stop the render call if input polydata is not set, since PolyDataPainter cannot paint without any polydata input.

39.143 vtkPolygonsPainter

39.143.1 Usage

This painter renders Polys in vtkPolyData. It can render the polys in any representation (VTK_POINTS, VTK_WIREFRAME, VTK_SURFACE).

To create an instance of class vtkPolygonsPainter, simply invoke its constructor as follows

```
obj = vtkPolygonsPainter
```

39.143.2 Methods

The class vtkPolygonsPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolygonsPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPolygonsPainter = obj.NewInstance ()`
- `vtkPolygonsPainter = obj.SafeDownCast (vtkObject o)`

39.144 vtkPOVExporter

39.144.1 Usage

This Exporter can be attached to a render window in order to generate scene description files for the Persistance of Vision Raytracer www.povray.org.

.SECTION Thanks Li-Ta Lo (ollie@lanl.gov) and Jim Ahrens (ahrens@lanl.gov) Los Alamos National Laboratory

To create an instance of class vtkPOVExporter, simply invoke its constructor as follows

```
obj = vtkPOVExporter
```

39.144.2 Methods

The class vtkPOVExporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPOVExporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPOVExporter = obj.NewInstance ()`
- `vtkPOVExporter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )`
- `string = obj.GetFileName ()`

39.145 vtkPrimitivePainter

39.145.1 Usage

This is the abstract superclass for classes that handle single type of primitive i.e. verts, lines, polys or tstrips. Concrete subclasses will pass a Render() call to the delegate painter, if any, only if it could not render. .SECTION Thanks Support for generic vertex attributes in VTK was contributed in collaboration with Stephane Ploix at EDF.

To create an instance of class vtkPrimitivePainter, simply invoke its constructor as follows

```
obj = vtkPrimitivePainter
```

39.145.2 Methods

The class vtkPrimitivePainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPrimitivePainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPrimitivePainter = obj.NewInstance ()`
- `vtkPrimitivePainter = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetSupportedPrimitive ()` - Get the type of primitive supported by this painter. This must be set by concrete subclasses.

39.146 vtkProp3D

39.146.1 Usage

vtkProp3D is an abstract class used to represent an entity in a rendering scene (i.e., vtkProp3D is a vtkProp with an associated transformation matrix). It handles functions related to the position, orientation and scaling. It combines these instance variables into one 4x4 transformation matrix as follows: [x y z 1] = [x y z 1] Translate(-origin) Scale(scale) Rot(y) Rot(x) Rot (z) Trans(origin) Trans(position). Both vtkActor and vtkVolume are specializations of class vtkProp. The constructor defaults to: origin(0,0,0) position=(0,0,0) orientation=(0,0,0), no user defined matrix or transform, and no texture map.

To create an instance of class vtkProp3D, simply invoke its constructor as follows

```
obj = vtkProp3D
```

39.146.2 Methods

The class vtkProp3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProp3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProp3D = obj.NewInstance ()`
- `vtkProp3D = obj.SafeDownCast (vtkObject o)`
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this vtkProp3D.

- `obj.SetPosition (double \_arg1, double \_arg2, double \_arg3)` - Set/Get/Add the position of the Prop3D in world coordinates.
- `obj.SetPosition (double \_arg[3])` - Set/Get/Add the position of the Prop3D in world coordinates.
- `double = obj. GetPosition ()` - Set/Get/Add the position of the Prop3D in world coordinates.
- `obj.AddPosition (double deltaPosition[3])` - Set/Get/Add the position of the Prop3D in world coordinates.
- `obj.AddPosition (double deltaX, double deltaY, double deltaZ)` - Set/Get/Add the position of the Prop3D in world coordinates.
- `obj.SetOrigin (double \_arg1, double \_arg2, double \_arg3)` - Set/Get the origin of the Prop3D. This is the point about which all rotations take place.
- `obj.SetOrigin (double \_arg[3])` - Set/Get the origin of the Prop3D. This is the point about which all rotations take place.
- `double = obj. GetOrigin ()` - Set/Get the origin of the Prop3D. This is the point about which all rotations take place.
- `obj.SetScale (double \_arg1, double \_arg2, double \_arg3)` - Set/Get the scale of the actor. Scaling in performed independently on the X, Y and Z axis. A scale of zero is illegal and will be replaced with one.
- `obj.SetScale (double \_arg[3])` - Set/Get the scale of the actor. Scaling in performed independently on the X, Y and Z axis. A scale of zero is illegal and will be replaced with one.
- `double = obj. GetScale ()` - Set/Get the scale of the actor. Scaling in performed independently on the X, Y and Z axis. A scale of zero is illegal and will be replaced with one.
- `obj.SetScale (double s)` - Method to set the scale isotropically
- `obj.SetUserTransform (vtkLinearTransform transform)` - In addition to the instance variables such as position and orientation, you can add an additional transformation for your own use. This transformation is concatenated with the actor's internal transformation, which you implicitly create through the use of SetPosition(), SetOrigin() and SetOrientation(). ¡p¿If the internal transformation is identity (i.e. if you don't set the Position, Origin, or Orientation) then the actors final transformation will be the UserTransform, concatenated with the UserMatrix if the UserMatrix is present.
- `vtkLinearTransform = obj.GetUserTransform ()` - In addition to the instance variables such as position and orientation, you can add an additional transformation for your own use. This transformation is concatenated with the actor's internal transformation, which you implicitly create through the use of SetPosition(), SetOrigin() and SetOrientation(). ¡p¿If the internal transformation is identity (i.e. if you don't set the Position, Origin, or Orientation) then the actors final transformation will be the UserTransform, concatenated with the UserMatrix if the UserMatrix is present.
- `obj.SetUserMatrix (vtkMatrix4x4 matrix)` - The UserMatrix can be used in place of UserTransform.
- `vtkMatrix4x4 = obj.GetUserMatrix ()` - The UserMatrix can be used in place of UserTransform.
- `obj.GetMatrix (vtkMatrix4x4 m)` - Return a reference to the Prop3D's 4x4 composite matrix. Get the matrix from the position, origin, scale and orientation This matrix is cached, so multiple GetMatrix() calls will be efficient.
- `obj.GetMatrix (double m[16])` - Return a reference to the Prop3D's 4x4 composite matrix. Get the matrix from the position, origin, scale and orientation This matrix is cached, so multiple GetMatrix() calls will be efficient.

- `obj.GetBounds (double bounds[6])` - Get the bounds for this Prop3D as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax).
- `double = obj.GetBounds ()` - Get the bounds for this Prop3D as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax).
- `double = obj.GetCenter ()` - Get the center of the bounding box in world coordinates.
- `double = obj.GetXRange ()` - Get the Prop3D's x range in world coordinates.
- `double = obj.GetYRange ()` - Get the Prop3D's y range in world coordinates.
- `double = obj.GetZRange ()` - Get the Prop3D's z range in world coordinates.
- `double = obj.GetLength ()` - Get the length of the diagonal of the bounding box.
- `obj.RotateX (double )` - Rotate the Prop3D in degrees about the X axis using the right hand rule. The axis is the Prop3D's X axis, which can change as other rotations are performed. To rotate about the world X axis use RotateWXYZ (angle, 1, 0, 0). This rotation is applied before all others in the current transformation matrix.
- `obj.RotateY (double )` - Rotate the Prop3D in degrees about the Y axis using the right hand rule. The axis is the Prop3D's Y axis, which can change as other rotations are performed. To rotate about the world Y axis use RotateWXYZ (angle, 0, 1, 0). This rotation is applied before all others in the current transformation matrix.
- `obj.RotateZ (double )` - Rotate the Prop3D in degrees about the Z axis using the right hand rule. The axis is the Prop3D's Z axis, which can change as other rotations are performed. To rotate about the world Z axis use RotateWXYZ (angle, 0, 0, 1). This rotation is applied before all others in the current transformation matrix.
- `obj.RotateWXYZ (double , double , double , double )` - Rotate the Prop3D in degrees about an arbitrary axis specified by the last three arguments. The axis is specified in world coordinates. To rotate an about its model axes, use RotateX, RotateY, RotateZ.
- `obj.SetOrientation (double , double , double )` - Sets the orientation of the Prop3D. Orientation is specified as X,Y and Z rotations in that order, but they are performed as RotateZ, RotateX, and finally RotateY.
- `obj.SetOrientation (double a[3])` - Sets the orientation of the Prop3D. Orientation is specified as X,Y and Z rotations in that order, but they are performed as RotateZ, RotateX, and finally RotateY.
- `double = obj.GetOrientation ()` - Returns the orientation of the Prop3D as s vector of X,Y and Z rotation. The ordering in which these rotations must be done to generate the same matrix is RotateZ, RotateX, and finally RotateY. See also SetOrientation.
- `obj.GetOrientation (double o[3])` - Returns the orientation of the Prop3D as s vector of X,Y and Z rotation. The ordering in which these rotations must be done to generate the same matrix is RotateZ, RotateX, and finally RotateY. See also SetOrientation.
- `double = obj.GetOrientationWXYZ ()` - Returns the WXYZ orientation of the Prop3D.
- `obj.AddOrientation (double , double , double )` - Add to the current orientation. See SetOrientation and GetOrientation for more details. This basically does a GetOrientation, adds the passed in arguments, and then calls SetOrientation.
- `obj.AddOrientation (double a[3])` - Add to the current orientation. See SetOrientation and GetOrientation for more details. This basically does a GetOrientation, adds the passed in arguments, and then calls SetOrientation.

- `obj.PokeMatrix (vtkMatrix4x4 matrix)` - This method modifies the vtkProp3D so that its transformation state is set to the matrix specified. The method does this by setting appropriate transformation-related ivars to initial values (i.e., not transformed), and placing the user-supplied matrix into the UserMatrix of this vtkProp3D. If the method is called again with a NULL matrix, then the original state of the vtkProp3D will be restored. This method is used to support picking and assembly structures.
- `obj.InitPathTraversal ()` - Overload vtkProp's method for setting up assembly paths. See the documentation for vtkProp.
- `long = obj.GetMTime ()` - Get the vtkProp3D's mtime
- `long = obj.GetUserTransformMatrixMTime ()` - Get the modified time of the user matrix or user transform.
- `obj.ComputeMatrix ()` - Generate the matrix based on ivars
- `vtkMatrix4x4 = obj.GetMatrix ()` - Is the matrix for this actor identity
- `int = obj.GetIsIdentity ()` - Is the matrix for this actor identity

39.147 vtkProp3DCollection

39.147.1 Usage

vtkProp3DCollection represents and provides methods to manipulate a list of 3D props (i.e., vtkProp3D and subclasses). The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkProp3DCollection, simply invoke its constructor as follows

```
obj = vtkProp3DCollection
```

39.147.2 Methods

The class vtkProp3DCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProp3DCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProp3DCollection = obj.NewInstance ()`
- `vtkProp3DCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkProp3D p)` - Add an actor to the list.
- `vtkProp3D = obj.GetNextProp3D ()` - Get the next actor in the list.
- `vtkProp3D = obj.GetLastProp3D ()` - Get the last actor in the list.

39.148 vtkProperty

39.148.1 Usage

vtkProperty is an object that represents lighting and other surface properties of a geometric object. The primary properties that can be set are colors (overall, ambient, diffuse, specular, and edge color); specular power; opacity of the object; the representation of the object (points, wireframe, or surface); and the shading

method to be used (flat, Gouraud, and Phong). Also, some special graphics features like backface properties can be set and manipulated with this object.

To create an instance of class vtkProperty, simply invoke its constructor as follows

```
obj = vtkProperty
```

39.148.2 Methods

The class vtkProperty has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProperty class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProperty = obj.NewInstance ()`
- `vtkProperty = obj.SafeDownCast (vtkObject o)`
- `obj.DeepCopy (vtkProperty p)` - Assign one property to another.
- `obj.Render (vtkActor , vtkRenderer )` - This method causes the property to set up whatever is required for its instance variables. This is actually handled by a subclass of vtkProperty, which is created automatically. This method includes the invoking actor as an argument which can be used by property devices that require the actor.
- `obj.BackfaceRender (vtkActor , vtkRenderer )` - This method renders the property as a backface property. TwoSidedLighting must be turned off to see any backface properties. Note that only colors and opacity are used for backface properties. Other properties such as Representation, Culling are specified by the Property.
- `bool = obj.GetLighting ()` - Set/Get lighting flag for an object. Initial value is true.
- `obj.SetLighting (bool )` - Set/Get lighting flag for an object. Initial value is true.
- `obj.LightingOn ()` - Set/Get lighting flag for an object. Initial value is true.
- `obj.LightingOff ()` - Set/Get lighting flag for an object. Initial value is true.
- `obj.SetInterpolation (int )` - Set the shading interpolation method for an object.
- `int = obj.GetInterpolationMinValue ()` - Set the shading interpolation method for an object.
- `int = obj.GetInterpolationMaxValue ()` - Set the shading interpolation method for an object.
- `int = obj.GetInterpolation ()` - Set the shading interpolation method for an object.
- `obj.SetInterpolationToFlat ()` - Set the shading interpolation method for an object.
- `obj.SetInterpolationToGouraud ()` - Set the shading interpolation method for an object.
- `obj.SetInterpolationToPhong ()` - Set the shading interpolation method for an object.
- `string = obj.GetInterpolationAsString ()` - Set the shading interpolation method for an object.
- `obj.SetRepresentation (int )` - Control the surface geometry representation for the object.
- `int = obj.GetRepresentationMinValue ()` - Control the surface geometry representation for the object.
- `int = obj.GetRepresentationMaxValue ()` - Control the surface geometry representation for the object.

- `int = obj.GetRepresentation ()` - Control the surface geometry representation for the object.
- `obj.SetRepresentationToPoints ()` - Control the surface geometry representation for the object.
- `obj.SetRepresentationToWireframe ()` - Control the surface geometry representation for the object.
- `obj.SetRepresentationToSurface ()` - Control the surface geometry representation for the object.
- `string = obj.GetRepresentationAsString ()` - Control the surface geometry representation for the object.
- `obj.SetColor (double r, double g, double b)` - Set the color of the object. Has the side effect of setting the ambient diffuse and specular colors as well. This is basically a quick overall color setting method.
- `obj.SetColor (double a[3])` - Set the color of the object. Has the side effect of setting the ambient diffuse and specular colors as well. This is basically a quick overall color setting method.
- `double = obj.GetColor ()` - Set the color of the object. Has the side effect of setting the ambient diffuse and specular colors as well. This is basically a quick overall color setting method.
- `obj.GetColor (double rgb[3])` - Set the color of the object. Has the side effect of setting the ambient diffuse and specular colors as well. This is basically a quick overall color setting method.
- `obj.SetAmbient (double )` - Set/Get the ambient lighting coefficient.
- `double = obj.GetAmbientMinValue ()` - Set/Get the ambient lighting coefficient.
- `double = obj.GetAmbientMaxValue ()` - Set/Get the ambient lighting coefficient.
- `double = obj.GetAmbient ()` - Set/Get the ambient lighting coefficient.
- `obj.SetDiffuse (double )` - Set/Get the diffuse lighting coefficient.
- `double = obj.GetDiffuseMinValue ()` - Set/Get the diffuse lighting coefficient.
- `double = obj.GetDiffuseMaxValue ()` - Set/Get the diffuse lighting coefficient.
- `double = obj.GetDiffuse ()` - Set/Get the diffuse lighting coefficient.
- `obj.SetSpecular (double )` - Set/Get the specular lighting coefficient.
- `double = obj.GetSpecularMinValue ()` - Set/Get the specular lighting coefficient.
- `double = obj.GetSpecularMaxValue ()` - Set/Get the specular lighting coefficient.
- `double = obj.GetSpecular ()` - Set/Get the specular lighting coefficient.
- `obj.SetSpecularPower (double )` - Set/Get the specular power.
- `double = obj.GetSpecularPowerMinValue ()` - Set/Get the specular power.
- `double = obj.GetSpecularPowerMaxValue ()` - Set/Get the specular power.
- `double = obj.GetSpecularPower ()` - Set/Get the specular power.
- `obj.SetOpacity (double )` - Set/Get the object's opacity. 1.0 is totally opaque and 0.0 is completely transparent.
- `double = obj.GetOpacityMinValue ()` - Set/Get the object's opacity. 1.0 is totally opaque and 0.0 is completely transparent.
- `double = obj.GetOpacityMaxValue ()` - Set/Get the object's opacity. 1.0 is totally opaque and 0.0 is completely transparent.

- `double = obj.GetOpacity ()` - Set/Get the object's opacity. 1.0 is totally opaque and 0.0 is completely transparent.
- `obj.SetAmbientColor (double , double , double )` - Set/Get the ambient surface color. Not all renderers support separate ambient and diffuse colors. From a physical standpoint it really doesn't make too much sense to have both. For the rendering libraries that don't support both, the diffuse color is used.
- `obj.SetAmbientColor (double a[3])` - Set/Get the ambient surface color. Not all renderers support separate ambient and diffuse colors. From a physical standpoint it really doesn't make too much sense to have both. For the rendering libraries that don't support both, the diffuse color is used.
- `double = obj. GetAmbientColor ()` - Set/Get the ambient surface color. Not all renderers support separate ambient and diffuse colors. From a physical standpoint it really doesn't make too much sense to have both. For the rendering libraries that don't support both, the diffuse color is used.
- `obj.SetDiffuseColor (double , double , double )` - Set/Get the diffuse surface color.
- `obj.SetDiffuseColor (double a[3])` - Set/Get the diffuse surface color.
- `double = obj. GetDiffuseColor ()` - Set/Get the diffuse surface color.
- `obj.SetSpecularColor (double , double , double )` - Set/Get the specular surface color.
- `obj.SetSpecularColor (double a[3])` - Set/Get the specular surface color.
- `double = obj. GetSpecularColor ()` - Set/Get the specular surface color.
- `int = obj.GetEdgeVisibility ()` - Turn on/off the visibility of edges. On some renderers it is possible to render the edges of geometric primitives separately from the interior.
- `obj.SetEdgeVisibility (int )` - Turn on/off the visibility of edges. On some renderers it is possible to render the edges of geometric primitives separately from the interior.
- `obj.EdgeVisibilityOn ()` - Turn on/off the visibility of edges. On some renderers it is possible to render the edges of geometric primitives separately from the interior.
- `obj.EdgeVisibilityOff ()` - Turn on/off the visibility of edges. On some renderers it is possible to render the edges of geometric primitives separately from the interior.
- `obj.SetEdgeColor (double , double , double )` - Set/Get the color of primitive edges (if edge visibility is enabled).
- `obj.SetEdgeColor (double a[3])` - Set/Get the color of primitive edges (if edge visibility is enabled).
- `double = obj. GetEdgeColor ()` - Set/Get the color of primitive edges (if edge visibility is enabled).
- `obj.SetLineWidth (float )` - Set/Get the width of a Line. The width is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetLineWidthMinValue ()` - Set/Get the width of a Line. The width is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetLineWidthMaxValue ()` - Set/Get the width of a Line. The width is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetLineWidth ()` - Set/Get the width of a Line. The width is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `obj.SetLineStipplePattern (int )` - Set/Get the stippling pattern of a Line, as a 16-bit binary pattern (1 = pixel on, 0 = pixel off). This is only implemented for OpenGL. The default is 0xFFFF.

- `int = obj.GetLineStipplePattern ()` - Set/Get the stippling pattern of a Line, as a 16-bit binary pattern (1 = pixel on, 0 = pixel off). This is only implemented for OpenGL. The default is 0xFFFF.
- `obj.SetLineStippleRepeatFactor (int )` - Set/Get the stippling repeat factor of a Line, which specifies how many times each bit in the pattern is to be repeated. This is only implemented for OpenGL. The default is 1.
- `int = obj.GetLineStippleRepeatFactorMinValue ()` - Set/Get the stippling repeat factor of a Line, which specifies how many times each bit in the pattern is to be repeated. This is only implemented for OpenGL. The default is 1.
- `int = obj.GetLineStippleRepeatFactorMaxValue ()` - Set/Get the stippling repeat factor of a Line, which specifies how many times each bit in the pattern is to be repeated. This is only implemented for OpenGL. The default is 1.
- `int = obj.GetLineStippleRepeatFactor ()` - Set/Get the stippling repeat factor of a Line, which specifies how many times each bit in the pattern is to be repeated. This is only implemented for OpenGL. The default is 1.
- `obj.SetPointSize (float )` - Set/Get the diameter of a point. The size is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetPointSizeMinValue ()` - Set/Get the diameter of a point. The size is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetPointSizeMaxValue ()` - Set/Get the diameter of a point. The size is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `float = obj.GetPointSize ()` - Set/Get the diameter of a point. The size is expressed in screen units. This is only implemented for OpenGL. The default is 1.0.
- `int = obj.GetBackfaceCulling ()` - Turn on/off fast culling of polygons based on orientation of normal with respect to camera. If backface culling is on, polygons facing away from camera are not drawn.
- `obj.SetBackfaceCulling (int )` - Turn on/off fast culling of polygons based on orientation of normal with respect to camera. If backface culling is on, polygons facing away from camera are not drawn.
- `obj.BackfaceCullingOn ()` - Turn on/off fast culling of polygons based on orientation of normal with respect to camera. If backface culling is on, polygons facing away from camera are not drawn.
- `obj.BackfaceCullingOff ()` - Turn on/off fast culling of polygons based on orientation of normal with respect to camera. If backface culling is on, polygons facing away from camera are not drawn.
- `int = obj.GetFrontfaceCulling ()` - Turn on/off fast culling of polygons based on orientation of normal with respect to camera. If frontface culling is on, polygons facing towards camera are not drawn.
- `obj.SetFrontfaceCulling (int )` - Turn on/off fast culling of polygons based on orientation of normal with respect to camera. If frontface culling is on, polygons facing towards camera are not drawn.
- `obj.FrontfaceCullingOn ()` - Turn on/off fast culling of polygons based on orientation of normal with respect to camera. If frontface culling is on, polygons facing towards camera are not drawn.
- `obj.FrontfaceCullingOff ()` - Turn on/off fast culling of polygons based on orientation of normal with respect to camera. If frontface culling is on, polygons facing towards camera are not drawn.
- `vtkXMLMaterial = obj.GetMaterial ()` - Get the material representation used for shading. The material will be used only when shading is enabled.

- `string = obj.GetMaterialName ()` - Returns the name of the material currenly loaded, if any.
- `obj.LoadMaterial (string name)` - Load the material. The material can be the name of a built-on material or the filename for a VTK material XML description.
- `obj.LoadMaterialFromString (string materialxml)` - Load the material given the contents of the material file.
- `obj.LoadMaterial (vtkXMLMaterial )` - Load the material given the material representation.
- `obj.SetShading (int )` - Enable/Disable shading. When shading is enabled, the Material must be set.
- `int = obj.GetShading ()` - Enable/Disable shading. When shading is enabled, the Material must be set.
- `obj.ShadingOn ()` - Enable/Disable shading. When shading is enabled, the Material must be set.
- `obj.ShadingOff ()` - Enable/Disable shading. When shading is enabled, the Material must be set.
- `vtkShaderProgram = obj.GetShaderProgram ()` - Get the Shader program. If Material is not set/or not loaded properly, this will return null.
- `obj.AddShaderVariable (string name, int numVars, int x)` - Provide values to initialize shader variables. Useful to initialize shader variables that change over time (animation, GUI widgets inputs, etc.) - name - hardware name of the uniform variable - numVars - number of variables being set - x - values
- `obj.AddShaderVariable (string name, int numVars, float x)` - Provide values to initialize shader variables. Useful to initialize shader variables that change over time (animation, GUI widgets inputs, etc.) - name - hardware name of the uniform variable - numVars - number of variables being set - x - values
- `obj.AddShaderVariable (string name, int numVars, double x)` - Provide values to initialize shader variables. Useful to initialize shader variables that change over time (animation, GUI widgets inputs, etc.) - name - hardware name of the uniform variable - numVars - number of variables being set - x - values
- `obj.AddShaderVariable (string name, int v)` - Methods to provide to add shader variables from tcl.
- `obj.AddShaderVariable (string name, float v)` - Methods to provide to add shader variables from tcl.
- `obj.AddShaderVariable (string name, double v)` - Methods to provide to add shader variables from tcl.
- `obj.AddShaderVariable (string name, int v1, int v2)` - Methods to provide to add shader variables from tcl.
- `obj.AddShaderVariable (string name, float v1, float v2)` - Methods to provide to add shader variables from tcl.
- `obj.AddShaderVariable (string name, double v1, double v2)` - Methods to provide to add shader variables from tcl.
- `obj.AddShaderVariable (string name, int v1, int v2, int v3)` - Methods to provide to add shader variables from tcl.
- `obj.AddShaderVariable (string name, float v1, float v2, float v3)` - Methods to provide to add shader variables from tcl.

- `obj.AddShaderVariable (string name, double v1, double v2, double v3)` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. A property does not need to have an associated texture map and multiple properties can share one texture. Textures must be assigned unique names.
- `obj.SetTexture (string name, vtkTexture texture)` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. A property does not need to have an associated texture map and multiple properties can share one texture. Textures must be assigned unique names.
- `vtkTexture = obj.GetTexture (string name)` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. A property does not need to have an associated texture map and multiple properties can share one texture. Textures must be assigned unique names.
- `obj.SetTexture (int unit, vtkTexture texture)` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. A property does not need to have an associated texture map and multiple properties can share one texture. Textures must be assigned unique names.
- `vtkTexture = obj.GetTexture (int unit)` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. A property does not need to have an associated texture map and multiple properties can share one texture. Textures must be assigned unique names.
- `obj.RemoveTexture (int unit)` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. A property does not need to have an associated texture map and multiple properties can share one texture. Textures must be assigned unique names.
- `obj.RemoveTexture (string name)` - Remove a texture from the collection. Note that the indices of all the subsquent textures, if any, will change.
- `obj.RemoveAllTextures ()` - Remove all the textures.
- `int = obj.GetNumberOfTextures ()` - Returns the number of textures in this property.
- `obj.ReleaseGraphicsResources (vtkWindow win)` - Release any graphics resources that are being consumed by this property. The parameter window could be used to determine which graphic resources to release.

39.149 vtkPropPicker

39.149.1 Usage

vtkPropPicker is used to pick an actor/prop given a selection point (in display coordinates) and a renderer. This class uses graphics hardware/rendering system to pick rapidly (as compared to using ray casting as does vtkCellPicker and vtkPointPicker). This class determines the actor/prop and pick position in world coordinates; point and cell ids are not determined.

To create an instance of class vtkPropPicker, simply invoke its constructor as follows

```
obj = vtkPropPicker
```

39.149.2 Methods

The class vtkPropPicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPropPicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkPropPicker = obj.NewInstance ()`
- `vtkPropPicker = obj.SafeDownCast (vtkObject o)`
- `int = obj.PickProp (double selectionX, double selectionY, vtkRenderer renderer)` - Perform the pick and set the PickedProp ivar. If something is picked, a 1 is returned, otherwise 0 is returned. Use the GetViewProp() method to get the instance of vtkProp that was picked. Props are picked from the renderers list of pickable Props.
- `int = obj.PickProp (double selectionX, double selectionY, vtkRenderer renderer, vtkPropCollection p` - Perform a pick from the user-provided list of vtkProps and not from the list of vtkProps that the render maintains.
- `int = obj.Pick (double selectionX, double selectionY, double selectionZ, vtkRenderer renderer)` - Overide superclasses' Pick() method.
- `int = obj.Pick (double selectionPt[3], vtkRenderer renderer)` - Overide superclasses' Pick() method.

39.150 vtkQImageToImageSource

39.150.1 Usage

vtkQImageToImageSource produces image data from a QImage.

To create an instance of class vtkQImageToImageSource, simply invoke its constructor as follows

```
obj = vtkQImageToImageSource
```

39.150.2 Methods

The class vtkQImageToImageSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQImageToImageSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQImageToImageSource = obj.NewInstance ()`
- `vtkQImageToImageSource = obj.SafeDownCast (vtkObject o)`

39.151 vtkQtInitialization

39.151.1 Usage

Utility class that initializes Qt by creating an instance of QApplication in its ctor, if one doesn't already exist. This is mainly of use in ParaView with filters that use Qt in their implementation - create an instance of vtkQtInitialization prior to instantiating any filters that require Qt.

To create an instance of class vtkQtInitialization, simply invoke its constructor as follows

```
obj = vtkQtInitialization
```

39.151.2 Methods

The class vtkQtInitialization has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQtInitialization class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQtInitialization = obj.NewInstance ()`
- `vtkQtInitialization = obj.SafeDownCast (vtkObject o)`

39.152 vtkQtLabelRenderStrategy

39.152.1 Usage

This class uses Qt to render labels and compute sizes. The labels are rendered to a QImage, then EndFrame() converts that image to a vtkImageData and textures the image onto a quad spanning the render area.

To create an instance of class vtkQtLabelRenderStrategy, simply invoke its constructor as follows

```
obj = vtkQtLabelRenderStrategy
```

39.152.2 Methods

The class vtkQtLabelRenderStrategy has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQtLabelRenderStrategy class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQtLabelRenderStrategy = obj.NewInstance ()`
- `vtkQtLabelRenderStrategy = obj.SafeDownCast (vtkObject o)`
- `obj.StartFrame ()` - Start a rendering frame. Renderer must be set.
- `obj.EndFrame ()` - End a rendering frame.
- `obj.ReleaseGraphicsResources (vtkWindow window)` - Release any graphics resources that are being consumed by this strategy. The parameter window could be used to determine which graphic resources to release.

39.153 vtkQtTreeRingLabelMapper

39.153.1 Usage

vtkQtTreeRingLabelMapper is a mapper that renders text on a tree map. A tree map is a vtkTree with an associated 4-tuple array used for storing the boundary rectangle for each vertex in the tree. The user must specify the array name used for storing the rectangles.

The mapper iterates through the tree and attempts and renders a label inside the vertex's rectangle as long as the following conditions hold: 1. The vertex level is within the range of levels specified for labeling. 2. The label can fully fit inside its box. 3. The label does not overlap an ancestor's label.

To create an instance of class vtkQtTreeRingLabelMapper, simply invoke its constructor as follows

```
obj = vtkQtTreeRingLabelMapper
```

39.153.2 Methods

The class vtkQtTreeRingLabelMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQtTreeRingLabelMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkQtTreeRingLabelMapper = obj.NewInstance ()`
- `vtkQtTreeRingLabelMapper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderOpaqueGeometry (vtkViewport viewport, vtkActor2D actor)` - Draw the text to the screen at each input point.
- `obj.RenderOverlay (vtkViewport viewport, vtkActor2D actor)` - Draw the text to the screen at each input point.
- `vtkTree = obj.GetInputTree ()` - The input to this filter.
- `obj.SetSectorsArrayName (string name)` - The name of the 4-tuple array used for
- `obj.SetLabelTextProperty (vtkTextProperty p)` - Set/Get the text property. Note that multiple type text properties (set with a second integer parameter) are not currently supported, but are provided to avoid compiler warnings.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - Set/Get the text property. Note that multiple type text properties (set with a second integer parameter) are not currently supported, but are provided to avoid compiler warnings.
- `obj.SetLabelTextProperty (vtkTextProperty p, int type)` - Set/Get the text property. Note that multiple type text properties (set with a second integer parameter) are not currently supported, but are provided to avoid compiler warnings.
- `vtkTextProperty = obj.GetLabelTextProperty (int type)` - Set/Get the name of the text rotation array.
- `obj.SetTextRotationArrayName (string )` - Set/Get the name of the text rotation array.
- `string = obj.GetTextRotationArrayName ()` - Set/Get the name of the text rotation array.
- `long = obj.GetMTime ()` - Return the object's MTime. This is overridden to include the timestamp of its internal class.
- `obj.SetRenderer (vtkRenderer ren)`
- `vtkRenderer = obj.GetRenderer ()`

39.154 vtkQuadricLODActor

39.154.1 Usage

vtkQuadricLODActor implements a specific strategy for level-of-detail using the vtkQuadricClustering decimation algorithm. It supports only two levels of detail: full resolution and a decimated version. The decimated LOD is generated using a tuned strategy to produce output consistent with the requested interactive frame rate (i.e., the vtkRenderWindowInteractor's DesiredUpdateRate). It also makes use of display lists for performance, and adjusts the vtkQuadricClustering algorithm to take into account the dimensionality of

the data (e.g., 2D, x-y surfaces may be binned into n x n x 1 to reduce extra polygons in the z-direction). Finally, the filter may optionally be set in "Static" mode (this works with the vtkMapper::SetStatic() method). 'Enabling Static results in a one time execution of the Mapper's pipeline. After that, the pipeline no longer updated (unless manually forced to do so).

To create an instance of class vtkQuadricLODActor, simply invoke its constructor as follows

```
obj = vtkQuadricLODActor
```

39.154.2 Methods

The class vtkQuadricLODActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuadricLODActor class.

- `string = obj.GetClassName ()` - Standard class methods.
- `int = obj.IsA (string name)` - Standard class methods.
- `vtkQuadricLODActor = obj.NewInstance ()` - Standard class methods.
- `vtkQuadricLODActor = obj.SafeDownCast (vtkObject o)` - Standard class methods.
- `obj.SetDeferLODConstruction (int )` - Specify whether to build the LOD immediately (i.e., on the first render) or to wait until the LOD is requested in a subsequent render. By default, LOD construction is not deferred (DeferLODConstruction is false).
- `int = obj.GetDeferLODConstruction ()` - Specify whether to build the LOD immediately (i.e., on the first render) or to wait until the LOD is requested in a subsequent render. By default, LOD construction is not deferred (DeferLODConstruction is false).
- `obj.DeferLODConstructionOn ()` - Specify whether to build the LOD immediately (i.e., on the first render) or to wait until the LOD is requested in a subsequent render. By default, LOD construction is not deferred (DeferLODConstruction is false).
- `obj.DeferLODConstructionOff ()` - Specify whether to build the LOD immediately (i.e., on the first render) or to wait until the LOD is requested in a subsequent render. By default, LOD construction is not deferred (DeferLODConstruction is false).
- `obj.SetStatic (int )` - Turn on/off a flag to control whether the underlying pipeline is static. If static, this means that the data pipeline executes once and then not again until the user manually modifies this class. By default, Static is off because trying to debug this is tricky, and you should only use it when you know what you are doing.
- `int = obj.GetStatic ()` - Turn on/off a flag to control whether the underlying pipeline is static. If static, this means that the data pipeline executes once and then not again until the user manually modifies this class. By default, Static is off because trying to debug this is tricky, and you should only use it when you know what you are doing.
- `obj.StaticOn ()` - Turn on/off a flag to control whether the underlying pipeline is static. If static, this means that the data pipeline executes once and then not again until the user manually modifies this class. By default, Static is off because trying to debug this is tricky, and you should only use it when you know what you are doing.
- `obj.StaticOff ()` - Turn on/off a flag to control whether the underlying pipeline is static. If static, this means that the data pipeline executes once and then not again until the user manually modifies this class. By default, Static is off because trying to debug this is tricky, and you should only use it when you know what you are doing.

- `obj.SetDataConfiguration (int )` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `int = obj.GetDataConfigurationMinValue ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `int = obj.GetDataConfigurationMaxValue ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `int = obj.GetDataConfiguration ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `obj.SetDataConfigurationToUnknown ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `obj.SetDataConfigurationToXLine ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `obj.SetDataConfigurationToYLine ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies

in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `obj.SetDataConfigurationToZLine ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `obj.SetDataConfigurationToXYPlane ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `obj.SetDataConfigurationToYZPlane ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `obj.SetDataConfigurationToXZPlane ()` - Force the binning of the quadric clustering according to application knowledge relative to the dimension of the data. For example, if you know your data lies in a 2D x-y plane, the performance of the quadric clustering algorithm can be greatly improved by indicating this (i.e., the number of resulting triangles, and the quality of the decimation version is better). Setting this parameter forces the binning to be configured consistent with the dimnesionality of the data, and the collapse dimension ratio is ignored. Specifying the value of DataConfiguration to UNKNOWN (the default value) means that the class will attempt to figure the dimension of the class automatically using the CollapseDimensionRatio ivar.

- `obj.SetDataConfigurationToXYZVolume ()` - If the data configuration is set to UNKNOWN, this class attempts to figure out the dimensionality of the data using CollapseDimensionRatio. This ivar is the ratio of short edge of the input bounding box to its long edge, which is then used to collapse the data dimension (and set the quadric bin size in that direction to one). By default, this value is 0.05.

- `obj.SetCollapseDimensionRatio (double )` - If the data configuration is set to UNKNOWN, this class attempts to figure out the dimensionality of the data using CollapseDimensionRatio. This ivar is the ratio of short edge of the input bounding box to its long edge, which is then used to collapse the data dimension (and set the quadric bin size in that direction to one). By default, this value is 0.05.

- `double = obj.GetCollapseDimensionRatioMinValue ()` - If the data configuration is set to UNKNOWN, this class attempts to figure out the dimensionality of the data using CollapseDimensionRatio. This ivar is the ratio of short edge of the input bounding box to its long edge, which is then used

to collapse the data dimension (and set the quadric bin size in that direction to one). By default, this value is 0.05.

- `double = obj.GetCollapseDimensionRatioMaxValue ()` - If the data configuration is set to UNKNOWN, this class attempts to figure out the dimensionality of the data using CollapseDimensionRatio. This ivar is the ratio of short edge of the input bounding box to its long edge, which is then used to collapse the data dimension (and set the quadric bin size in that direction to one). By default, this value is 0.05.

- `double = obj.GetCollapseDimensionRatio ()` - If the data configuration is set to UNKNOWN, this class attempts to figure out the dimensionality of the data using CollapseDimensionRatio. This ivar is the ratio of short edge of the input bounding box to its long edge, which is then used to collapse the data dimension (and set the quadric bin size in that direction to one). By default, this value is 0.05.

- `obj.SetLODFilter (vtkQuadricClustering lodFilter)` - This class will create a vtkQuadricClustering algorithm automatically. However, if you would like to specify the filter to use, or to access it and configure it, these method provide access to the filter.

- `vtkQuadricClustering = obj.GetLODFilter ()` - This class will create a vtkQuadricClustering algorithm automatically. However, if you would like to specify the filter to use, or to access it and configure it, these method provide access to the filter.

- `obj.SetMaximumDisplayListSize (int )` - Specify the maximum display list size. This variable is used to determine whether to use display lists (ImmediateModeRenderingOff) or not. Controlling display list size is important to prevent program crashes (i.e., overly large display lists on some graphics hardware will cause faults). The display list size is the length of the vtkCellArray representing the topology of the input vtkPolyData.

- `int = obj.GetMaximumDisplayListSizeMinValue ()` - Specify the maximum display list size. This variable is used to determine whether to use display lists (ImmediateModeRenderingOff) or not. Controlling display list size is important to prevent program crashes (i.e., overly large display lists on some graphics hardware will cause faults). The display list size is the length of the vtkCellArray representing the topology of the input vtkPolyData.

- `int = obj.GetMaximumDisplayListSizeMaxValue ()` - Specify the maximum display list size. This variable is used to determine whether to use display lists (ImmediateModeRenderingOff) or not. Controlling display list size is important to prevent program crashes (i.e., overly large display lists on some graphics hardware will cause faults). The display list size is the length of the vtkCellArray representing the topology of the input vtkPolyData.

- `int = obj.GetMaximumDisplayListSize ()` - Specify the maximum display list size. This variable is used to determine whether to use display lists (ImmediateModeRenderingOff) or not. Controlling display list size is important to prevent program crashes (i.e., overly large display lists on some graphics hardware will cause faults). The display list size is the length of the vtkCellArray representing the topology of the input vtkPolyData.

- `obj.SetPropType (int )` - Indicate that this actor is actually a follower. By default, the prop type is a vtkActor.

- `int = obj.GetPropTypeMinValue ()` - Indicate that this actor is actually a follower. By default, the prop type is a vtkActor.

- `int = obj.GetPropTypeMaxValue ()` - Indicate that this actor is actually a follower. By default, the prop type is a vtkActor.

- `int = obj.GetPropType ()` - Indicate that this actor is actually a follower. By default, the prop type is a vtkActor.

- `obj.SetPropTypeToFollower ()` - Indicate that this actor is actually a follower. By default, the prop type is a vtkActor.
- `obj.SetPropTypeToActor ()` - Set/Get the camera to follow. This method is only applicable when the prop type is set to a vtkFollower.
- `obj.SetCamera (vtkCamera )` - Set/Get the camera to follow. This method is only applicable when the prop type is set to a vtkFollower.
- `vtkCamera = obj.GetCamera ()` - Set/Get the camera to follow. This method is only applicable when the prop type is set to a vtkFollower.
- `obj.Render (vtkRenderer , vtkMapper )` - This causes the actor to be rendered. Depending on the frame rate request, it will use either a full resolution render or an interactive render (i.e., it will use the decimated geometry).
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - This method is used internally by the rendering process. We overide the superclass method to properly set the estimated render time.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of an LOD actor. Overloads the virtual vtkProp method.

39.155 vtkQuaternionInterpolator

39.155.1 Usage

This class is used to interpolate a series of quaternions representing the rotations of a 3D object. The interpolation may be linear in form (using spherical linear interpolation SLERP), or via spline interpolation (using SQUAD). In either case the interpolation is specialized to quaternions since the interpolation occurs on the surface of the unit quaternion sphere.

To use this class, specify at least two pairs of (t,q[4]) with the AddQuaternion() method. Next interpolate the tuples with the InterpolateQuaternion(t,q[4]) method, where "t" must be in the range of (t_min,t_max) parameter values specified by the AddQuaternion() method (t is clamped otherwise), and q[4] is filled in by the method.

There are several important background references. Ken Shoemake described the practical application of quaternions for the interpolation of rotation (K. Shoemake, "Animating rotation with quaternion curves", Computer Graphics (Siggraph '85) 19(3):245–254, 1985). Another fine reference (available on-line) is E. B. Dam, M. Koch, and M. Lillholm, Technical Report DIKU-TR-98/5, Dept. of Computer Science, University of Copenhagen, Denmark.

To create an instance of class vtkQuaternionInterpolator, simply invoke its constructor as follows

```
obj = vtkQuaternionInterpolator
```

39.155.2 Methods

The class vtkQuaternionInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkQuaternionInterpolator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkQuaternionInterpolator = obj.NewInstance ()`
- `vtkQuaternionInterpolator = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfQuaternions ()` - Return the number of quaternions in the list of quaternions to be interpolated.
- `double = obj.GetMinimumT ()` - Obtain some information about the interpolation range. The numbers returned (corresponding to parameter t, usually thought of as time) are undefined if the list of transforms is empty. This is a convenience method for interpolation.
- `double = obj.GetMaximumT ()` - Obtain some information about the interpolation range. The numbers returned (corresponding to parameter t, usually thought of as time) are undefined if the list of transforms is empty. This is a convenience method for interpolation.
- `obj.Initialize ()` - Reset the class so that it contains no data; i.e., the array of (t,q[4]) information is discarded.
- `obj.AddQuaternion (double t, double q[4])` - Add another quaternion to the list of quaternions to be interpolated. Note that using the same time t value more than once replaces the previous quaternion at t. At least one quaternions must be added to define an interpolation functios.
- `obj.RemoveQuaternion (double t)` - Delete the quaternion at a particular parameter t. If there is no quaternion tuple defined at t, then the method does nothing.
- `obj.InterpolateQuaternion (double t, double q[4])` - Interpolate the list of quaternions and determine a new quaternion (i.e., fill in the quaternion provided). If t is outside the range of (min,max) values, then t is clamped to lie within the range.
- `obj.SetInterpolationType (int )` - Specify which type of function to use for interpolation. By default (SetInterpolationFunctionToSpline()), cubic spline interpolation using a modifed Kochanek basis is employed. Otherwise, if SetInterpolationFunctionToLinear() is invoked, linear spherical interpolation is used between each pair of quaternions.
- `int = obj.GetInterpolationTypeMinValue ()` - Specify which type of function to use for interpolation. By default (SetInterpolationFunctionToSpline()), cubic spline interpolation using a modifed Kochanek basis is employed. Otherwise, if SetInterpolationFunctionToLinear() is invoked, linear spherical interpolation is used between each pair of quaternions.
- `int = obj.GetInterpolationTypeMaxValue ()` - Specify which type of function to use for interpolation. By default (SetInterpolationFunctionToSpline()), cubic spline interpolation using a modifed Kochanek basis is employed. Otherwise, if SetInterpolationFunctionToLinear() is invoked, linear spherical interpolation is used between each pair of quaternions.
- `int = obj.GetInterpolationType ()` - Specify which type of function to use for interpolation. By default (SetInterpolationFunctionToSpline()), cubic spline interpolation using a modifed Kochanek basis is employed. Otherwise, if SetInterpolationFunctionToLinear() is invoked, linear spherical interpolation is used between each pair of quaternions.
- `obj.SetInterpolationTypeToLinear ()` - Specify which type of function to use for interpolation. By default (SetInterpolationFunctionToSpline()), cubic spline interpolation using a modifed Kochanek basis is employed. Otherwise, if SetInterpolationFunctionToLinear() is invoked, linear spherical interpolation is used between each pair of quaternions.
- `obj.SetInterpolationTypeToSpline ()`

39.156 vtkRenderedAreaPicker

39.156.1 Usage

Like vtkAreaPicker, this class picks all props within a selection area on the screen. The difference is in implementation. This class uses graphics hardware to perform the test where the other uses software bounding box/frustum intersection testing.

This picker is more conservative than vtkAreaPicker. It will reject some objects that pass the bounding box test of vtkAreaPicker. This will happen, for instance, when picking through a corner of the bounding box when the data set does not have any visible geometry in that corner.

To create an instance of class vtkRenderedAreaPicker, simply invoke its constructor as follows

```
obj = vtkRenderedAreaPicker
```

39.156.2 Methods

The class vtkRenderedAreaPicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderedAreaPicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderedAreaPicker = obj.NewInstance ()`
- `vtkRenderedAreaPicker = obj.SafeDownCast (vtkObject o)`
- `int = obj.AreaPick (double x0, double y0, double x1, double y1, vtkRenderer renderer)` - Perform pick operation in volume behind the given screen coordinates. Props intersecting the selection frustum will be accesible via GetProp3D. GetPlanes returns a vtkImplicitFunciton suitable for vtkExtractGeometry.

39.157 vtkRenderer

39.157.1 Usage

vtkRenderer provides an abstract specification for renderers. A renderer is an object that controls the rendering process for objects. Rendering is the process of converting geometry, a specification for lights, and a camera view into an image. vtkRenderer also performs coordinate transformation between world coordinates, view coordinates (the computer graphics rendering coordinate system), and display coordinates (the actual screen coordinates on the display device). Certain advanced rendering features such as two-sided lighting can also be controlled.

To create an instance of class vtkRenderer, simply invoke its constructor as follows

```
obj = vtkRenderer
```

39.157.2 Methods

The class vtkRenderer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderer class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkRenderer = obj.NewInstance ()`
- `vtkRenderer = obj.SafeDownCast (vtkObject o)`
- `obj.AddActor (vtkProp p)` - Add/Remove different types of props to the renderer. These methods are all synonyms to AddViewProp and RemoveViewProp. They are here for convenience and backwards compatibility.
- `obj.AddVolume (vtkProp p)` - Add/Remove different types of props to the renderer. These methods are all synonyms to AddViewProp and RemoveViewProp. They are here for convenience and backwards compatibility.
- `obj.RemoveActor (vtkProp p)` - Add/Remove different types of props to the renderer. These methods are all synonyms to AddViewProp and RemoveViewProp. They are here for convenience and backwards compatibility.
- `obj.RemoveVolume (vtkProp p)` - Add/Remove different types of props to the renderer. These methods are all synonyms to AddViewProp and RemoveViewProp. They are here for convenience and backwards compatibility.
- `obj.AddLight (vtkLight )` - Add a light to the list of lights.
- `obj.RemoveLight (vtkLight )` - Remove a light from the list of lights.
- `obj.RemoveAllLights ()` - Remove all lights from the list of lights.
- `vtkLightCollection = obj.GetLights ()` - Return the collection of lights.
- `obj.SetLightCollection (vtkLightCollection lights)` - Set the collection of lights. We cannot name it SetLights because of TestSetGet
- `obj.CreateLight (void )` - Create and add a light to renderer.
- `vtkLight = obj.MakeLight ()` - Create a new Light sutible for use with this type of Renderer. For example, a vtkMesaRenderer should create a vtkMesaLight in this function. The default is to just call vtkLight::New.
- `int = obj.GetTwoSidedLighting ()` - Turn on/off two-sided lighting of surfaces. If two-sided lighting is off, then only the side of the surface facing the light(s) will be lit, and the other side dark. If two-sided lighting on, both sides of the surface will be lit.
- `obj.SetTwoSidedLighting (int )` - Turn on/off two-sided lighting of surfaces. If two-sided lighting is off, then only the side of the surface facing the light(s) will be lit, and the other side dark. If two-sided lighting on, both sides of the surface will be lit.
- `obj.TwoSidedLightingOn ()` - Turn on/off two-sided lighting of surfaces. If two-sided lighting is off, then only the side of the surface facing the light(s) will be lit, and the other side dark. If two-sided lighting on, both sides of the surface will be lit.
- `obj.TwoSidedLightingOff ()` - Turn on/off two-sided lighting of surfaces. If two-sided lighting is off, then only the side of the surface facing the light(s) will be lit, and the other side dark. If two-sided lighting on, both sides of the surface will be lit.
- `obj.SetLightFollowCamera (int )` - Turn on/off the automatic repositioning of lights as the camera moves. If LightFollowCamera is on, lights that are designated as Headlights or CameraLights will be adjusted to move with this renderer's camera. If LightFollowCamera is off, the lights will not be adjusted.

  (Note: In previous versions of vtk, this light-tracking functionality was part of the interactors, not the renderer. For backwards compatibility, the older, more limited interactor behavior is enabled by default. To disable this mode, turn the interactor's LightFollowCamera flag OFF, and leave the renderer's LightFollowCamera flag ON.)

- `int = obj.GetLightFollowCamera ()` - Turn on/off the automatic repositioning of lights as the camera moves. If LightFollowCamera is on, lights that are designated as Headlights or CameraLights will be adjusted to move with this renderer's camera. If LightFollowCamera is off, the lights will not be adjusted.

  (Note: In previous versions of vtk, this light-tracking functionality was part of the interactors, not the renderer. For backwards compatibility, the older, more limited interactor behavior is enabled by default. To disable this mode, turn the interactor's LightFollowCamera flag OFF, and leave the renderer's LightFollowCamera flag ON.)

- `obj.LightFollowCameraOn ()` - Turn on/off the automatic repositioning of lights as the camera moves. If LightFollowCamera is on, lights that are designated as Headlights or CameraLights will be adjusted to move with this renderer's camera. If LightFollowCamera is off, the lights will not be adjusted.

  (Note: In previous versions of vtk, this light-tracking functionality was part of the interactors, not the renderer. For backwards compatibility, the older, more limited interactor behavior is enabled by default. To disable this mode, turn the interactor's LightFollowCamera flag OFF, and leave the renderer's LightFollowCamera flag ON.)

- `obj.LightFollowCameraOff ()` - Turn on/off the automatic repositioning of lights as the camera moves. If LightFollowCamera is on, lights that are designated as Headlights or CameraLights will be adjusted to move with this renderer's camera. If LightFollowCamera is off, the lights will not be adjusted.

  (Note: In previous versions of vtk, this light-tracking functionality was part of the interactors, not the renderer. For backwards compatibility, the older, more limited interactor behavior is enabled by default. To disable this mode, turn the interactor's LightFollowCamera flag OFF, and leave the renderer's LightFollowCamera flag ON.)

- `int = obj.GetAutomaticLightCreation ()` - Turn on/off a flag which disables the automatic light creation capability. Normally in VTK if no lights are associated with the renderer, then a light is automatically created. However, in special circumstances this feature is undesirable, so the following boolean is provided to disable automatic light creation. (Turn AutomaticLightCreation off if you do not want lights to be created.)

- `obj.SetAutomaticLightCreation (int )` - Turn on/off a flag which disables the automatic light creation capability. Normally in VTK if no lights are associated with the renderer, then a light is automatically created. However, in special circumstances this feature is undesirable, so the following boolean is provided to disable automatic light creation. (Turn AutomaticLightCreation off if you do not want lights to be created.)

- `obj.AutomaticLightCreationOn ()` - Turn on/off a flag which disables the automatic light creation capability. Normally in VTK if no lights are associated with the renderer, then a light is automatically created. However, in special circumstances this feature is undesirable, so the following boolean is provided to disable automatic light creation. (Turn AutomaticLightCreation off if you do not want lights to be created.)

- `obj.AutomaticLightCreationOff ()` - Turn on/off a flag which disables the automatic light creation capability. Normally in VTK if no lights are associated with the renderer, then a light is automatically created. However, in special circumstances this feature is undesirable, so the following boolean is provided to disable automatic light creation. (Turn AutomaticLightCreation off if you do not want lights to be created.)

- `int = obj.UpdateLightsGeometryToFollowCamera (void )` - Ask the lights in the scene that are not in world space (for instance, Headlights or CameraLights that are attached to the camera) to update their geometry to match the active camera.

- `vtkVolumeCollection = obj.GetVolumes ()` - Return the collection of volumes.

- `vtkActorCollection = obj.GetActors ()` - Return any actors in this renderer.
- `obj.SetActiveCamera (vtkCamera )` - Specify the camera to use for this renderer.
- `vtkCamera = obj.GetActiveCamera ()` - Get the current camera. If there is not camera assigned to the renderer already, a new one is created automatically. This does *not* reset the camera.
- `vtkCamera = obj.MakeCamera ()` - Create a new Camera sutible for use with this type of Renderer. For example, a vtkMesaRenderer should create a vtkMesaCamera in this function. The default is to just call vtkCamera::New.
- `obj.SetErase (int )` - When this flag is off, the renderer will not erase the background or the Zbuffer. It is used to have overlapping renderers. Both the RenderWindow Erase and Render Erase must be on for the camera to clear the renderer. By default, Erase is on.
- `int = obj.GetErase ()` - When this flag is off, the renderer will not erase the background or the Zbuffer. It is used to have overlapping renderers. Both the RenderWindow Erase and Render Erase must be on for the camera to clear the renderer. By default, Erase is on.
- `obj.EraseOn ()` - When this flag is off, the renderer will not erase the background or the Zbuffer. It is used to have overlapping renderers. Both the RenderWindow Erase and Render Erase must be on for the camera to clear the renderer. By default, Erase is on.
- `obj.EraseOff ()` - When this flag is off, the renderer will not erase the background or the Zbuffer. It is used to have overlapping renderers. Both the RenderWindow Erase and Render Erase must be on for the camera to clear the renderer. By default, Erase is on.
- `obj.SetDraw (int )` - When this flag is off, render commands are ignored. It is used to either multiplex a vtkRenderWindow or render only part of a vtkRenderWindow. By default, Draw is on.
- `int = obj.GetDraw ()` - When this flag is off, render commands are ignored. It is used to either multiplex a vtkRenderWindow or render only part of a vtkRenderWindow. By default, Draw is on.
- `obj.DrawOn ()` - When this flag is off, render commands are ignored. It is used to either multiplex a vtkRenderWindow or render only part of a vtkRenderWindow. By default, Draw is on.
- `obj.DrawOff ()` - When this flag is off, render commands are ignored. It is used to either multiplex a vtkRenderWindow or render only part of a vtkRenderWindow. By default, Draw is on.
- `obj.AddCuller (vtkCuller )` - Add an culler to the list of cullers.
- `obj.RemoveCuller (vtkCuller )` - Remove an actor from the list of cullers.
- `vtkCullerCollection = obj.GetCullers ()` - Return the collection of cullers.
- `obj.SetAmbient (double , double , double )` - Set the intensity of ambient lighting.
- `obj.SetAmbient (double  a[3])` - Set the intensity of ambient lighting.
- `double = obj. GetAmbient ()` - Set the intensity of ambient lighting.
- `obj.SetAllocatedRenderTime (double )` - Set/Get the amount of time this renderer is allowed to spend rendering its scene. This is used by vtkLODActor's.
- `double = obj.GetAllocatedRenderTime ()` - Set/Get the amount of time this renderer is allowed to spend rendering its scene. This is used by vtkLODActor's.
- `double = obj.GetTimeFactor ()` - Get the ratio between allocated time and actual render time. TimeFactor has been taken out of the render process. It is still computed in case someone finds it useful. It may be taken away in the future.

- `obj.Render ()` - CALLED BY vtkRenderWindow ONLY. End-user pass your way and call vtkRenderWindow::Render(). Create an image. This is a superclass method which will in turn call the DeviceRender method of Subclasses of vtkRenderer.
- `obj.DeviceRender ()` - Create an image. Subclasses of vtkRenderer must implement this method.
- `obj.DeviceRenderTranslucentPolygonalGeometry ()` - Render translucent polygonal geometry. Default implementation just call UpdateTranslucentPolygonalGeometry(). Subclasses of vtkRenderer that can deal with depth peeling must override this method. It updates boolean ivar LastRenderingUsedDepthPeeling.
- `obj.Clear ()` - Clear the image to the background color.
- `int = obj.VisibleActorCount ()` - Returns the number of visible actors.
- `int = obj.VisibleVolumeCount ()` - Returns the number of visible volumes.
- `obj.ComputeVisiblePropBounds (double bounds[6])` - Compute the bounding box of all the visible props Used in ResetCamera() and ResetCameraClippingRange()
- `double = obj.ComputeVisiblePropBounds ()` - Wrapper-friendly version of ComputeVisiblePropBounds
- `obj.ResetCameraClippingRange ()` - Reset the camera clipping range based on the bounds of the visible actors. This ensures that no props are cut off
- `obj.ResetCameraClippingRange (double bounds[6])` - Reset the camera clipping range based on a bounding box. This method is called from ResetCameraClippingRange()
- `obj.ResetCameraClippingRange (double xmin, double xmax, double ymin, double ymax, double zmin, doub` - Reset the camera clipping range based on a bounding box. This method is called from ResetCameraClippingRange()
- `obj.SetNearClippingPlaneTolerance (double )` - Specify tolerance for near clipping plane distance to the camera as a percentage of the far clipping plane distance. By default this will be set to 0.01 for 16 bit zbuffers and 0.001 for higher depth z buffers
- `double = obj.GetNearClippingPlaneToleranceMinValue ()` - Specify tolerance for near clipping plane distance to the camera as a percentage of the far clipping plane distance. By default this will be set to 0.01 for 16 bit zbuffers and 0.001 for higher depth z buffers
- `double = obj.GetNearClippingPlaneToleranceMaxValue ()` - Specify tolerance for near clipping plane distance to the camera as a percentage of the far clipping plane distance. By default this will be set to 0.01 for 16 bit zbuffers and 0.001 for higher depth z buffers
- `double = obj.GetNearClippingPlaneTolerance ()` - Specify tolerance for near clipping plane distance to the camera as a percentage of the far clipping plane distance. By default this will be set to 0.01 for 16 bit zbuffers and 0.001 for higher depth z buffers
- `obj.ResetCamera ()` - Automatically set up the camera based on the visible actors. The camera will reposition itself to view the center point of the actors, and move along its initial view plane normal (i.e., vector defined from camera position to focal point) so that all of the actors can be seen.
- `obj.ResetCamera (double bounds[6])` - Automatically set up the camera based on a specified bounding box (xmin,xmax, ymin,ymax, zmin,zmax). Camera will reposition itself so that its focal point is the center of the bounding box, and adjust its distance and position to preserve its initial view plane normal (i.e., vector defined from camera position to focal point). Note: is the view plane is parallel to the view up axis, the view up axis will be reset to one of the three coordinate axes.

- `obj.ResetCamera (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Alternative version of ResetCamera(bounds[6]);
- `obj.SetRenderWindow (vtkRenderWindow )` - Specify the rendering window in which to draw. This is automatically set when the renderer is created by MakeRenderer. The user probably shouldn't ever need to call this method.
- `vtkRenderWindow = obj.GetRenderWindow ()` - Specify the rendering window in which to draw. This is automatically set when the renderer is created by MakeRenderer. The user probably shouldn't ever need to call this method.
- `vtkWindow = obj.GetVTKWindow ()` - Specify the rendering window in which to draw. This is automatically set when the renderer is created by MakeRenderer. The user probably shouldn't ever need to call this method.
- `obj.SetBackingStore (int )` - Turn on/off using backing store. This may cause the re-rendering time to be slightly slower when the view changes. But it is much faster when the image has not changed, such as during an expose event.
- `int = obj.GetBackingStore ()` - Turn on/off using backing store. This may cause the re-rendering time to be slightly slower when the view changes. But it is much faster when the image has not changed, such as during an expose event.
- `obj.BackingStoreOn ()` - Turn on/off using backing store. This may cause the re-rendering time to be slightly slower when the view changes. But it is much faster when the image has not changed, such as during an expose event.
- `obj.BackingStoreOff ()` - Turn on/off using backing store. This may cause the re-rendering time to be slightly slower when the view changes. But it is much faster when the image has not changed, such as during an expose event.
- `obj.SetInteractive (int )` - Turn on/off interactive status. An interactive renderer is one that can receive events from an interactor. Should only be set if there are multiple renderers in the same section of the viewport.
- `int = obj.GetInteractive ()` - Turn on/off interactive status. An interactive renderer is one that can receive events from an interactor. Should only be set if there are multiple renderers in the same section of the viewport.
- `obj.InteractiveOn ()` - Turn on/off interactive status. An interactive renderer is one that can receive events from an interactor. Should only be set if there are multiple renderers in the same section of the viewport.
- `obj.InteractiveOff ()` - Turn on/off interactive status. An interactive renderer is one that can receive events from an interactor. Should only be set if there are multiple renderers in the same section of the viewport.
- `obj.SetLayer (int )` - Set/Get the layer that this renderer belongs to. This is only used if there are layered renderers.
- `int = obj.GetLayer ()` - Set/Get the layer that this renderer belongs to. This is only used if there are layered renderers.
- `obj.SetPreserveDepthBuffer (int )` - Normally a renderer is treated as transparent if Layer ¿ 0. To treat a renderer at Layer 0 as transparent, set this flag to true.
- `int = obj.GetPreserveDepthBuffer ()` - Normally a renderer is treated as transparent if Layer ¿ 0. To treat a renderer at Layer 0 as transparent, set this flag to true.

- `obj.PreserveDepthBufferOn ()` - Normally a renderer is treated as transparent if Layer ¿ 0. To treat a renderer at Layer 0 as transparent, set this flag to true.
- `obj.PreserveDepthBufferOff ()` - Normally a renderer is treated as transparent if Layer ¿ 0. To treat a renderer at Layer 0 as transparent, set this flag to true.
- `int = obj.Transparent ()` - Returns a boolean indicating if this renderer is transparent. It is transparent if it is not in the deepest layer of its render window.
- `obj.WorldToView ()` - Convert world point coordinates to view coordinates.
- `obj.ViewToWorld ()` - Convert view point coordinates to world coordinates.
- `double = obj.GetZ (int x, int y)` - Given a pixel location, return the Z value. The z value is normalized (0,1) between the front and back clipping planes.
- `long = obj.GetMTime ()` - Return the MTime of the renderer also considering its ivars.
- `double = obj.GetLastRenderTimeInSeconds ()` - Get the time required, in seconds, for the last Render call.
- `int = obj.GetNumberOfPropsRendered ()` - Should be used internally only during a render Get the number of props that were rendered using a RenderOpaqueGeometry or RenderTranslucentPolygonalGeometry call. This is used to know if something is in the frame buffer.
- `vtkAssemblyPath = obj.PickProp (double selectionX, double selectionY)` - Return the prop (via a vtkAssemblyPath) that has the highest z value at the given x, y position in the viewport. Basically, the top most prop that renders the pixel at selectionX, selectionY will be returned. If nothing was picked then NULL is returned. This method selects from the renderers Prop list.
- `vtkAssemblyPath = obj.PickProp (double selectionX1, double selectionY1, double selectionX2, double` - Return the prop (via a vtkAssemblyPath) that has the highest z value at the given x, y position in the viewport. Basically, the top most prop that renders the pixel at selectionX, selectionY will be returned. If nothing was picked then NULL is returned. This method selects from the renderers Prop list.
- `obj.StereoMidpoint ()` - Do anything necessary between rendering the left and right viewpoints in a stereo render. Doesn't do anything except in the derived vtkIceTRenderer in ParaView.
- `double = obj.GetTiledAspectRatio ()` - Compute the aspect ratio of this renderer for the current tile. When tiled displays are used the aspect ratio of the renderer for a given tile may be diferent that the aspect ratio of the renderer when rendered in it entirity
- `int = obj.IsActiveCameraCreated ()` - Turn on/off rendering of translucent material with depth peeling technique. The render window must have alpha bits (ie call SetAlphaBitPlanes(1)) and no multisample buffer (ie call SetMultiSamples(0)) to support depth peeling. If UseDepthPeeling is on and the GPU supports it, depth peeling is used for rendering translucent materials. If UseDepthPeeling is off, alpha blending is used. Initial value is off.
- `obj.SetUseDepthPeeling (int )` - Turn on/off rendering of translucent material with depth peeling technique. The render window must have alpha bits (ie call SetAlphaBitPlanes(1)) and no multisample buffer (ie call SetMultiSamples(0)) to support depth peeling. If UseDepthPeeling is on and the GPU supports it, depth peeling is used for rendering translucent materials. If UseDepthPeeling is off, alpha blending is used. Initial value is off.
- `int = obj.GetUseDepthPeeling ()` - Turn on/off rendering of translucent material with depth peeling technique. The render window must have alpha bits (ie call SetAlphaBitPlanes(1)) and no multisample buffer (ie call SetMultiSamples(0)) to support depth peeling. If UseDepthPeeling is on and the GPU supports it, depth peeling is used for rendering translucent materials. If UseDepthPeeling is off, alpha blending is used. Initial value is off.

- `obj.UseDepthPeelingOn ()` - Turn on/off rendering of translucent material with depth peeling technique. The render window must have alpha bits (ie call SetAlphaBitPlanes(1)) and no multisample buffer (ie call SetMultiSamples(0)) to support depth peeling. If UseDepthPeeling is on and the GPU supports it, depth peeling is used for rendering translucent materials. If UseDepthPeeling is off, alpha blending is used. Initial value is off.

- `obj.UseDepthPeelingOff ()` - Turn on/off rendering of translucent material with depth peeling technique. The render window must have alpha bits (ie call SetAlphaBitPlanes(1)) and no multisample buffer (ie call SetMultiSamples(0)) to support depth peeling. If UseDepthPeeling is on and the GPU supports it, depth peeling is used for rendering translucent materials. If UseDepthPeeling is off, alpha blending is used. Initial value is off.

- `obj.SetOcclusionRatio (double )` - In case of use of depth peeling technique for rendering translucent material, define the threshold under which the algorithm stops to iterate over peel layers. This is the ratio of the number of pixels that have been touched by the last layer over the total number of pixels of the viewport area. Initial value is 0.0, meaning rendering have to be exact. Greater values may speed-up the rendering with small impact on the quality.

- `double = obj.GetOcclusionRatioMinValue ()` - In case of use of depth peeling technique for rendering translucent material, define the threshold under which the algorithm stops to iterate over peel layers. This is the ratio of the number of pixels that have been touched by the last layer over the total number of pixels of the viewport area. Initial value is 0.0, meaning rendering have to be exact. Greater values may speed-up the rendering with small impact on the quality.

- `double = obj.GetOcclusionRatioMaxValue ()` - In case of use of depth peeling technique for rendering translucent material, define the threshold under which the algorithm stops to iterate over peel layers. This is the ratio of the number of pixels that have been touched by the last layer over the total number of pixels of the viewport area. Initial value is 0.0, meaning rendering have to be exact. Greater values may speed-up the rendering with small impact on the quality.

- `double = obj.GetOcclusionRatio ()` - In case of use of depth peeling technique for rendering translucent material, define the threshold under which the algorithm stops to iterate over peel layers. This is the ratio of the number of pixels that have been touched by the last layer over the total number of pixels of the viewport area. Initial value is 0.0, meaning rendering have to be exact. Greater values may speed-up the rendering with small impact on the quality.

- `obj.SetMaximumNumberOfPeels (int )` - In case of depth peeling, define the maximum number of peeling layers. Initial value is 4. A special value of 0 means no maximum limit. It has to be a positive value.

- `int = obj.GetMaximumNumberOfPeels ()` - In case of depth peeling, define the maximum number of peeling layers. Initial value is 4. A special value of 0 means no maximum limit. It has to be a positive value.

- `int = obj.GetLastRenderingUsedDepthPeeling ()` - Tells if the last call to DeviceRenderTranslucentPolygonalGeometry() actually used depth peeling. Initial value is false.

- `obj.SetDelegate (vtkRendererDelegate d)` - Set/Get a custom Render call. Allows to hook a Render call from an external project.It will be used in place of vtkRenderer::Render() if it is not NULL and its Used ivar is set to true. Initial value is NULL.

- `vtkRendererDelegate = obj.GetDelegate ()` - Set/Get a custom Render call. Allows to hook a Render call from an external project.It will be used in place of vtkRenderer::Render() if it is not NULL and its Used ivar is set to true. Initial value is NULL.

- `obj.SetPass (vtkRenderPass p)` - Set/Get a custom render pass. Initial value is NULL.

- `vtkRenderPass = obj.GetPass ()` - Set/Get a custom render pass. Initial value is NULL.

- `vtkHardwareSelector = obj.GetSelector ()` - Get the current hardware selector. If the Selector is set, it implies the current render pass is for selection. Mappers/Properties may choose to behave differently when rendering for hardware selection.
- `obj.SetBackgroundTexture (vtkTexture )` - Set/Get the texture to be used for the background. If set and enabled this gets the priority over the gradient background.
- `vtkTexture = obj.GetBackgroundTexture ()` - Set/Get the texture to be used for the background. If set and enabled this gets the priority over the gradient background.
- `obj.SetTexturedBackground (bool )` - Set/Get whether this viewport should have a textured background. Default is off.
- `bool = obj.GetTexturedBackground ()` - Set/Get whether this viewport should have a textured background. Default is off.
- `obj.TexturedBackgroundOn ()` - Set/Get whether this viewport should have a textured background. Default is off.
- `obj.TexturedBackgroundOff ()` - Set/Get whether this viewport should have a textured background. Default is off.

39.158 vtkRendererCollection

39.158.1 Usage

vtkRendererCollection represents and provides methods to manipulate a list of renderers (i.e., vtkRenderer and subclasses). The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkRendererCollection, simply invoke its constructor as follows

```
obj = vtkRendererCollection
```

39.158.2 Methods

The class vtkRendererCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRendererCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRendererCollection = obj.NewInstance ()`
- `vtkRendererCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkRenderer a)` - Get the next Renderer in the list. Return NULL when at the end of the list.
- `vtkRenderer = obj.GetNextItem ()` - Get the next Renderer in the list. Return NULL when at the end of the list.
- `obj.Render ()` - Forward the Render() method to each renderer in the list.
- `vtkRenderer = obj.GetFirstRenderer ()` - Get the first Renderer in the list. Return NULL when at the end of the list.

39.159 vtkRendererDelegate

39.159.1 Usage

vtkRendererDelegate is an abstract class with a pure virtual method Render. This method replaces the Render method of vtkRenderer to allow custom rendering from an external project. A RendererDelegate is connected to a vtkRenderer with method SetDelegate(). An external project just has to provide a concrete implementation of vtkRendererDelegate.

To create an instance of class vtkRendererDelegate, simply invoke its constructor as follows

```
obj = vtkRendererDelegate
```

39.159.2 Methods

The class vtkRendererDelegate has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRendererDelegate class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRendererDelegate = obj.NewInstance ()`
- `vtkRendererDelegate = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer r)` - Render the props of vtkRenderer if Used is on.
- `obj.SetUsed (bool )` - Tells if the delegate has to be used by the renderer or not. Initial value is off.
- `bool = obj.GetUsed ()` - Tells if the delegate has to be used by the renderer or not. Initial value is off.
- `obj.UsedOn ()` - Tells if the delegate has to be used by the renderer or not. Initial value is off.
- `obj.UsedOff ()` - Tells if the delegate has to be used by the renderer or not. Initial value is off.

39.160 vtkRendererSource

39.160.1 Usage

vtkRendererSource is a source object that gets its input from a renderer and converts it to structured points. This can then be used in a visualization pipeline. You must explicitly send a Modify() to this object to get it to reload its data from the renderer. Consider using vtkWindowToImageFilter instead of this class.

The data placed into the output is the renderer's image rgb values. Optionally, you can also grab the image depth (e.g., z-buffer) values, and place then into the output (point) field data.

To create an instance of class vtkRendererSource, simply invoke its constructor as follows

```
obj = vtkRendererSource
```

39.160.2 Methods

The class vtkRendererSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRendererSource class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkRendererSource = obj.NewInstance ()`
- `vtkRendererSource = obj.SafeDownCast (vtkObject o)`
- `long = obj.GetMTime ()` - Return the MTime also considering the Renderer.
- `obj.SetInput (vtkRenderer )` - Indicates what renderer to get the pixel data from.
- `vtkRenderer = obj.GetInput ()` - Returns which renderer is being used as the source for the pixel data.
- `obj.SetWholeWindow (int )` - Use the entire RenderWindow as a data source or just the Renderer. The default is zero, just the Renderer.
- `int = obj.GetWholeWindow ()` - Use the entire RenderWindow as a data source or just the Renderer. The default is zero, just the Renderer.
- `obj.WholeWindowOn ()` - Use the entire RenderWindow as a data source or just the Renderer. The default is zero, just the Renderer.
- `obj.WholeWindowOff ()` - Use the entire RenderWindow as a data source or just the Renderer. The default is zero, just the Renderer.
- `obj.SetRenderFlag (int )` - If this flag is on, the Executing causes a render first.
- `int = obj.GetRenderFlag ()` - If this flag is on, the Executing causes a render first.
- `obj.RenderFlagOn ()` - If this flag is on, the Executing causes a render first.
- `obj.RenderFlagOff ()` - If this flag is on, the Executing causes a render first.
- `obj.SetDepthValues (int )` - A boolean value to control whether to grab z-buffer (i.e., depth values) along with the image data. The z-buffer data is placed into a field data attributes named "ZBuffer" .
- `int = obj.GetDepthValues ()` - A boolean value to control whether to grab z-buffer (i.e., depth values) along with the image data. The z-buffer data is placed into a field data attributes named "ZBuffer" .
- `obj.DepthValuesOn ()` - A boolean value to control whether to grab z-buffer (i.e., depth values) along with the image data. The z-buffer data is placed into a field data attributes named "ZBuffer" .
- `obj.DepthValuesOff ()` - A boolean value to control whether to grab z-buffer (i.e., depth values) along with the image data. The z-buffer data is placed into a field data attributes named "ZBuffer" .
- `obj.SetDepthValuesInScalars (int )` - A boolean value to control whether to grab z-buffer (i.e., depth values) along with the image data. The z-buffer data is placed in the scalars as a fourth Z component (shift and scaled to map the full 0..255 range).
- `int = obj.GetDepthValuesInScalars ()` - A boolean value to control whether to grab z-buffer (i.e., depth values) along with the image data. The z-buffer data is placed in the scalars as a fourth Z component (shift and scaled to map the full 0..255 range).
- `obj.DepthValuesInScalarsOn ()` - A boolean value to control whether to grab z-buffer (i.e., depth values) along with the image data. The z-buffer data is placed in the scalars as a fourth Z component (shift and scaled to map the full 0..255 range).
- `obj.DepthValuesInScalarsOff ()` - A boolean value to control whether to grab z-buffer (i.e., depth values) along with the image data. The z-buffer data is placed in the scalars as a fourth Z component (shift and scaled to map the full 0..255 range).
- `vtkImageData = obj.GetOutput ()` - Get the output data object for a port on this algorithm.

39.161 vtkRenderPass

39.161.1 Usage

vtkRenderPass is a deferred class with a simple deferred method Render. This method performs a rendering pass of the scene described in vtkRenderState. Subclasses define what really happens during rendering.

Directions to write a subclass of vtkRenderPass: It is up to the subclass to decide if it needs to delegate part of its job to some other vtkRenderPass objects ("delegates"). - The subclass has to define ivar to set/get its delegates. - The documentation of the subclass has to describe: - what each delegate is supposed to perform - if a delegate is supposed to be used once or multiple times - what it expects to have in the framebuffer before starting (status of colorbuffers, depth buffer, stencil buffer) - what it will change in the framebuffer. - A pass cannot modify the vtkRenderState where it will perform but it can build a new vtkRenderState (it can change the FrameBuffer, change the prop array, changed the required prop properties keys (usually adding some to a copy of the existing list) but it has to keep the same vtkRenderer object), make it current and pass it to its delegate. - at the end of the execution of Render, the pass has to ensure the current vtkRenderState is the one it has in argument.

To create an instance of class vtkRenderPass, simply invoke its constructor as follows

```
obj = vtkRenderPass
```

39.161.2 Methods

The class vtkRenderPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderPass = obj.NewInstance ()`
- `vtkRenderPass = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfRenderedProps ()` - Number of props rendered at the last Render call.
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Release graphics resources and ask components to release their own resources. Default implementation is empty.

39.162 vtkRenderPassCollection

39.162.1 Usage

vtkRenderPassCollection represents a list of RenderPasses (i.e., vtkRenderPass and subclasses) and provides methods to manipulate the list. The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkRenderPassCollection, simply invoke its constructor as follows

```
obj = vtkRenderPassCollection
```

39.162.2 Methods

The class vtkRenderPassCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderPassCollection class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkRenderPassCollection = obj.NewInstance ()`
- `vtkRenderPassCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkRenderPass pass)` - Add an RenderPass to the list.
- `vtkRenderPass = obj.GetNextRenderPass ()` - Get the next RenderPass in the list.
- `vtkRenderPass = obj.GetLastRenderPass ()` - Get the last RenderPass in the list.

39.163 vtkRenderWindow

39.163.1 Usage

vtkRenderWindow is an abstract object to specify the behavior of a rendering window. A rendering window is a window in a graphical user interface where renderers draw their images. Methods are provided to synchronize the rendering process, set window size, and control double buffering. The window also allows rendering in stereo. The interlaced render stereo type is for output to a VRex stereo projector. All of the odd horizontal lines are from the left eye, and the even lines are from the right eye. The user has to make the render window aligned with the VRex projector, or the eye will be swapped.

To create an instance of class vtkRenderWindow, simply invoke its constructor as follows

```
obj = vtkRenderWindow
```

39.163.2 Methods

The class vtkRenderWindow has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderWindow class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderWindow = obj.NewInstance ()`
- `vtkRenderWindow = obj.SafeDownCast (vtkObject o)`
- `obj.AddRenderer (vtkRenderer )` - Add a renderer to the list of renderers.
- `obj.RemoveRenderer (vtkRenderer )` - Remove a renderer from the list of renderers.
- `int = obj.HasRenderer (vtkRenderer )` - Query if a renderer is in the list of renderers.
- `vtkRendererCollection = obj.GetRenderers ()` - Return the collection of renderers in the render window.
- `obj.Render ()` - Ask each renderer owned by this RenderWindow to render its image and synchronize this process.
- `obj.Start ()` - Initialize the rendering process.
- `obj.Finalize ()` - Finalize the rendering process.
- `obj.Frame ()` - A termination method performed at the end of the rendering process to do things like swapping buffers (if necessary) or similar actions.

- `obj.WaitForCompletion ()` - Block the thread until the actual rendering is finished(). Useful for measurement only.
- `obj.CopyResultFrame ()` - Performed at the end of the rendering process to generate image. This is typically done right before swapping buffers.
- `vtkRenderWindowInteractor = obj.MakeRenderWindowInteractor ()` - Create an interactor to control renderers in this window. We need to know what type of interactor to create, because we might be in X Windows or MS Windows.
- `obj.HideCursor ()` - Hide or Show the mouse cursor, it is nice to be able to hide the default cursor if you want VTK to display a 3D cursor instead. Set cursor position in window (note that (0,0) is the lower left corner).
- `obj.ShowCursor ()` - Hide or Show the mouse cursor, it is nice to be able to hide the default cursor if you want VTK to display a 3D cursor instead. Set cursor position in window (note that (0,0) is the lower left corner).
- `obj.SetCursorPosition (int , int )` - Hide or Show the mouse cursor, it is nice to be able to hide the default cursor if you want VTK to display a 3D cursor instead. Set cursor position in window (note that (0,0) is the lower left corner).
- `obj.SetCurrentCursor (int )` - Change the shape of the cursor.
- `int = obj.GetCurrentCursor ()` - Change the shape of the cursor.
- `obj.SetFullScreen (int )` - Turn on/off rendering full screen window size.
- `int = obj.GetFullScreen ()` - Turn on/off rendering full screen window size.
- `obj.FullScreenOn ()` - Turn on/off rendering full screen window size.
- `obj.FullScreenOff ()` - Turn on/off rendering full screen window size.
- `obj.SetBorders (int )` - Turn on/off window manager borders. Typically, you shouldn't turn the borders off, because that bypasses the window manager and can cause undesirable behavior.
- `int = obj.GetBorders ()` - Turn on/off window manager borders. Typically, you shouldn't turn the borders off, because that bypasses the window manager and can cause undesirable behavior.
- `obj.BordersOn ()` - Turn on/off window manager borders. Typically, you shouldn't turn the borders off, because that bypasses the window manager and can cause undesirable behavior.
- `obj.BordersOff ()` - Turn on/off window manager borders. Typically, you shouldn't turn the borders off, because that bypasses the window manager and can cause undesirable behavior.
- `int = obj.GetStereoCapableWindow ()` - Prescribe that the window be created in a stereo-capable mode. This method must be called before the window is realized. Default is off.
- `obj.StereoCapableWindowOn ()` - Prescribe that the window be created in a stereo-capable mode. This method must be called before the window is realized. Default is off.
- `obj.StereoCapableWindowOff ()` - Prescribe that the window be created in a stereo-capable mode. This method must be called before the window is realized. Default is off.
- `obj.SetStereoCapableWindow (int capable)` - Prescribe that the window be created in a stereo-capable mode. This method must be called before the window is realized. Default is off.
- `int = obj.GetStereoRender ()` - Turn on/off stereo rendering.
- `obj.SetStereoRender (int stereo)` - Turn on/off stereo rendering.

- `obj.StereoRenderOn ()` - Turn on/off stereo rendering.
- `obj.StereoRenderOff ()` - Turn on/off stereo rendering.
- `obj.SetAlphaBitPlanes (int )` - Turn on/off the use of alpha bitplanes.
- `int = obj.GetAlphaBitPlanes ()` - Turn on/off the use of alpha bitplanes.
- `obj.AlphaBitPlanesOn ()` - Turn on/off the use of alpha bitplanes.
- `obj.AlphaBitPlanesOff ()` - Turn on/off the use of alpha bitplanes.
- `obj.SetPointSmoothing (int )` - Turn on/off point smoothing. Default is off. This must be applied before the first Render.
- `int = obj.GetPointSmoothing ()` - Turn on/off point smoothing. Default is off. This must be applied before the first Render.
- `obj.PointSmoothingOn ()` - Turn on/off point smoothing. Default is off. This must be applied before the first Render.
- `obj.PointSmoothingOff ()` - Turn on/off point smoothing. Default is off. This must be applied before the first Render.
- `obj.SetLineSmoothing (int )` - Turn on/off line smoothing. Default is off. This must be applied before the first Render.
- `int = obj.GetLineSmoothing ()` - Turn on/off line smoothing. Default is off. This must be applied before the first Render.
- `obj.LineSmoothingOn ()` - Turn on/off line smoothing. Default is off. This must be applied before the first Render.
- `obj.LineSmoothingOff ()` - Turn on/off line smoothing. Default is off. This must be applied before the first Render.
- `obj.SetPolygonSmoothing (int )` - Turn on/off polygon smoothing. Default is off. This must be applied before the first Render.
- `int = obj.GetPolygonSmoothing ()` - Turn on/off polygon smoothing. Default is off. This must be applied before the first Render.
- `obj.PolygonSmoothingOn ()` - Turn on/off polygon smoothing. Default is off. This must be applied before the first Render.
- `obj.PolygonSmoothingOff ()` - Turn on/off polygon smoothing. Default is off. This must be applied before the first Render.
- `int = obj.GetStereoType ()` - Set/Get what type of stereo rendering to use. CrystalEyes mode uses frame-sequential capabilities available in OpenGL to drive LCD shutter glasses and stereo projectors. RedBlue mode is a simple type of stereo for use with red-blue glasses. Anaglyph mode is a superset of RedBlue mode, but the color output channels can be configured using the AnaglyphColorMask and the color of the original image can be (somewhat) maintained using AnaglyphColorSaturation; the default colors for Anaglyph mode is red-cyan. Interlaced stereo mode produces a composite image where horizontal lines alternate between left and right views. StereoLeft and StereoRight modes choose one or the other stereo view. Dresden mode is yet another stereoscopic interleaving.

- `obj.SetStereoType (int )` - Set/Get what type of stereo rendering to use. CrystalEyes mode uses frame-sequential capabilities available in OpenGL to drive LCD shutter glasses and stereo projectors. RedBlue mode is a simple type of stereo for use with red-blue glasses. Anaglyph mode is a superset of RedBlue mode, but the color output channels can be configured using the AnaglyphColorMask and the color of the original image can be (somewhat) maintained using AnaglyphColorSaturation; the default colors for Anaglyph mode is red-cyan. Interlaced stereo mode produces a composite image where horizontal lines alternate between left and right views. StereoLeft and StereoRight modes choose one or the other stereo view. Dresden mode is yet another stereoscopic interleaving.
- `obj.SetStereoTypeToCrystalEyes ()` - Set/Get what type of stereo rendering to use. CrystalEyes mode uses frame-sequential capabilities available in OpenGL to drive LCD shutter glasses and stereo projectors. RedBlue mode is a simple type of stereo for use with red-blue glasses. Anaglyph mode is a superset of RedBlue mode, but the color output channels can be configured using the AnaglyphColorMask and the color of the original image can be (somewhat) maintained using AnaglyphColorSaturation; the default colors for Anaglyph mode is red-cyan. Interlaced stereo mode produces a composite image where horizontal lines alternate between left and right views. StereoLeft and StereoRight modes choose one or the other stereo view. Dresden mode is yet another stereoscopic interleaving.
- `obj.SetStereoTypeToRedBlue ()` - Set/Get what type of stereo rendering to use. CrystalEyes mode uses frame-sequential capabilities available in OpenGL to drive LCD shutter glasses and stereo projectors. RedBlue mode is a simple type of stereo for use with red-blue glasses. Anaglyph mode is a superset of RedBlue mode, but the color output channels can be configured using the AnaglyphColorMask and the color of the original image can be (somewhat) maintained using AnaglyphColorSaturation; the default colors for Anaglyph mode is red-cyan. Interlaced stereo mode produces a composite image where horizontal lines alternate between left and right views. StereoLeft and StereoRight modes choose one or the other stereo view. Dresden mode is yet another stereoscopic interleaving.
- `obj.SetStereoTypeToInterlaced ()` - Set/Get what type of stereo rendering to use. CrystalEyes mode uses frame-sequential capabilities available in OpenGL to drive LCD shutter glasses and stereo projectors. RedBlue mode is a simple type of stereo for use with red-blue glasses. Anaglyph mode is a superset of RedBlue mode, but the color output channels can be configured using the AnaglyphColorMask and the color of the original image can be (somewhat) maintained using AnaglyphColorSaturation; the default colors for Anaglyph mode is red-cyan. Interlaced stereo mode produces a composite image where horizontal lines alternate between left and right views. StereoLeft and StereoRight modes choose one or the other stereo view. Dresden mode is yet another stereoscopic interleaving.
- `obj.SetStereoTypeToLeft ()` - Set/Get what type of stereo rendering to use. CrystalEyes mode uses frame-sequential capabilities available in OpenGL to drive LCD shutter glasses and stereo projectors. RedBlue mode is a simple type of stereo for use with red-blue glasses. Anaglyph mode is a superset of RedBlue mode, but the color output channels can be configured using the AnaglyphColorMask and the color of the original image can be (somewhat) maintained using AnaglyphColorSaturation; the default colors for Anaglyph mode is red-cyan. Interlaced stereo mode produces a composite image where horizontal lines alternate between left and right views. StereoLeft and StereoRight modes choose one or the other stereo view. Dresden mode is yet another stereoscopic interleaving.
- `obj.SetStereoTypeToRight ()` - Set/Get what type of stereo rendering to use. CrystalEyes mode uses frame-sequential capabilities available in OpenGL to drive LCD shutter glasses and stereo projectors. RedBlue mode is a simple type of stereo for use with red-blue glasses. Anaglyph mode is a superset of RedBlue mode, but the color output channels can be configured using the AnaglyphColorMask and the color of the original image can be (somewhat) maintained using AnaglyphColorSaturation; the default colors for Anaglyph mode is red-cyan. Interlaced stereo mode produces a composite image where horizontal lines alternate between left and right views. StereoLeft and StereoRight modes choose one or the other stereo view. Dresden mode is yet another stereoscopic interleaving.
- `obj.SetStereoTypeToDresden ()` - Set/Get what type of stereo rendering to use. CrystalEyes mode uses frame-sequential capabilities available in OpenGL to drive LCD shutter glasses and stereo projectors. RedBlue mode is a simple type of stereo for use with red-blue glasses. Anaglyph mode is a superset

of RedBlue mode, but the color output channels can be configured using the AnaglyphColorMask and the color of the original image can be (somewhat) maintained using AnaglyphColorSaturation; the default colors for Anaglyph mode is red-cyan. Interlaced stereo mode produces a composite image where horizontal lines alternate between left and right views. StereoLeft and StereoRight modes choose one or the other stereo view. Dresden mode is yet another stereoscopic interleaving.

- `obj.SetStereoTypeToAnaglyph ()` - Set/Get what type of stereo rendering to use. CrystalEyes mode uses frame-sequential capabilities available in OpenGL to drive LCD shutter glasses and stereo projectors. RedBlue mode is a simple type of stereo for use with red-blue glasses. Anaglyph mode is a superset of RedBlue mode, but the color output channels can be configured using the AnaglyphColorMask and the color of the original image can be (somewhat) maintained using AnaglyphColorSaturation; the default colors for Anaglyph mode is red-cyan. Interlaced stereo mode produces a composite image where horizontal lines alternate between left and right views. StereoLeft and StereoRight modes choose one or the other stereo view. Dresden mode is yet another stereoscopic interleaving.
- `obj.SetStereoTypeToCheckerboard ()`
- `string = obj.GetStereoTypeAsString ()`
- `obj.StereoUpdate ()` - Update the system, if needed, due to stereo rendering. For some stereo methods, subclasses might need to switch some hardware settings here.
- `obj.StereoMidpoint ()` - Intermediate method performs operations required between the rendering of the left and right eye.
- `obj.StereoRenderComplete ()` - Handles work required once both views have been rendered when using stereo rendering.
- `obj.SetAnaglyphColorSaturation (float )`
- `float = obj.GetAnaglyphColorSaturationMinValue ()`
- `float = obj.GetAnaglyphColorSaturationMaxValue ()`
- `float = obj.GetAnaglyphColorSaturation ()`
- `obj.SetAnaglyphColorMask (int , int )`
- `obj.SetAnaglyphColorMask (int  a[2])`
- `int = obj. GetAnaglyphColorMask ()`
- `obj.WindowRemap ()` - Remap the rendering window. This probably only works on UNIX right now. It is useful for changing properties that can't normally be changed once the window is up.
- `obj.SetSwapBuffers (int )` - Turn on/off buffer swapping between images.
- `int = obj.GetSwapBuffers ()` - Turn on/off buffer swapping between images.
- `obj.SwapBuffersOn ()` - Turn on/off buffer swapping between images.
- `obj.SwapBuffersOff ()` - Turn on/off buffer swapping between images.
- `int = obj.SetPixelData (int x, int y, int x2, int y2, string data, int front)` - Set/Get the pixel data of an image, transmitted as RGBRGBRGB. The front argument indicates if the front buffer should be used or the back buffer. It is the caller's responsibility to delete the resulting array. It is very important to realize that the memory in this array is organized from the bottom of the window to the top. The origin of the screen is in the lower left corner. The y axis increases as you go up the screen. So the storage of pixels is from left to right and from bottom to top. (x,y) is any corner of the rectangle. (x2,y2) is its opposite corner on the diagonal.

- `int = obj.SetPixelData (int x, int y, int x2, int y2, vtkUnsignedCharArray data, int front)` - Set/Get the pixel data of an image, transmitted as RGBRGBRGB. The front argument indicates if the front buffer should be used or the back buffer. It is the caller's responsibility to delete the resulting array. It is very important to realize that the memory in this array is organized from the bottom of the window to the top. The origin of the screen is in the lower left corner. The y axis increases as you go up the screen. So the storage of pixels is from left to right and from bottom to top. (x,y) is any corner of the rectangle. (x2,y2) is its opposite corner on the diagonal.

- `int = obj.GetRGBAPixelData (int x, int y, int x2, int y2, int front, vtkFloatArray data)` - Same as Get/SetPixelData except that the image also contains an alpha component. The image is transmitted as RGBARGBARGBA... each of which is a float value. The "blend" parameter controls whether the SetRGBAPixelData method blends the data with the previous contents of the frame buffer or completely replaces the frame buffer data.

- `int = obj.SetRGBAPixelData (int x, int y, int x2, int y2, float , int front, int blend)` - Same as Get/SetPixelData except that the image also contains an alpha component. The image is transmitted as RGBARGBARGBA... each of which is a float value. The "blend" parameter controls whether the SetRGBAPixelData method blends the data with the previous contents of the frame buffer or completely replaces the frame buffer data.

- `int = obj.SetRGBAPixelData (int , int , int , int , vtkFloatArray , int , int blend)` - Same as Get/SetPixelData except that the image also contains an alpha component. The image is transmitted as RGBARGBARGBA... each of which is a float value. The "blend" parameter controls whether the SetRGBAPixelData method blends the data with the previous contents of the frame buffer or completely replaces the frame buffer data.

- `obj.ReleaseRGBAPixelData (float data)` - Same as Get/SetPixelData except that the image also contains an alpha component. The image is transmitted as RGBARGBARGBA... each of which is a float value. The "blend" parameter controls whether the SetRGBAPixelData method blends the data with the previous contents of the frame buffer or completely replaces the frame buffer data.

- `int = obj.GetRGBACharPixelData (int x, int y, int x2, int y2, int front, vtkUnsignedCharArray data)` - Same as Get/SetPixelData except that the image also contains an alpha component. The image is transmitted as RGBARGBARGBA... each of which is a float value. The "blend" parameter controls whether the SetRGBAPixelData method blends the data with the previous contents of the frame buffer or completely replaces the frame buffer data.

- `int = obj.SetRGBACharPixelData (int x, int y, int x2, int y2, string data, int front, int blend)` - Same as Get/SetPixelData except that the image also contains an alpha component. The image is transmitted as RGBARGBARGBA... each of which is a float value. The "blend" parameter controls whether the SetRGBAPixelData method blends the data with the previous contents of the frame buffer or completely replaces the frame buffer data.

- `int = obj.SetRGBACharPixelData (int x, int y, int x2, int y2, vtkUnsignedCharArray data, int front,` - Same as Get/SetPixelData except that the image also contains an alpha component. The image is transmitted as RGBARGBARGBA... each of which is a float value. The "blend" parameter controls whether the SetRGBAPixelData method blends the data with the previous contents of the frame buffer or completely replaces the frame buffer data.

- `int = obj.GetZbufferData (int x, int y, int x2, int y2, float z)` - Set/Get the zbuffer data from the frame buffer. (x,y) is any corner of the rectangle. (x2,y2) is its opposite corner on the diagonal.

- `int = obj.GetZbufferData (int x, int y, int x2, int y2, vtkFloatArray z)` - Set/Get the zbuffer data from the frame buffer. (x,y) is any corner of the rectangle. (x2,y2) is its opposite corner on the diagonal.

- `int = obj.SetZbufferData (int x, int y, int x2, int y2, float z)` - Set/Get the zbuffer data from the frame buffer. (x,y) is any corner of the rectangle. (x2,y2) is its opposite corner on the diagonal.

- `int = obj.SetZbufferData (int x, int y, int x2, int y2, vtkFloatArray z)` - Set/Get the zbuffer data from the frame buffer. (x,y) is any corner of the rectangle. (x2,y2) is its opposite corner on the diagonal.
- `float = obj.GetZbufferDataAtPoint (int x, int y)` - Set the number of frames for doing antialiasing. The default is zero. Typically five or six will yield reasonable results without taking too long.
- `int = obj.GetAAFrames ()` - Set the number of frames for doing antialiasing. The default is zero. Typically five or six will yield reasonable results without taking too long.
- `obj.SetAAFrames (int )` - Set the number of frames for doing antialiasing. The default is zero. Typically five or six will yield reasonable results without taking too long.
- `int = obj.GetFDFrames ()` - Set the number of frames for doing focal depth. The default is zero. Depending on how your scene is organized you can get away with as few as four frames for focal depth or you might need thirty. One thing to note is that if you are using focal depth frames, then you will not need many (if any) frames for antialiasing.
- `obj.SetFDFrames (int )` - Set the number of frames for doing focal depth. The default is zero. Depending on how your scene is organized you can get away with as few as four frames for focal depth or you might need thirty. One thing to note is that if you are using focal depth frames, then you will not need many (if any) frames for antialiasing.
- `int = obj.GetSubFrames ()` - Set the number of sub frames for doing motion blur. The default is zero. Once this is set greater than one, you will no longer see a new frame for every Render(). If you set this to five, you will need to do five Render() invocations before seeing the result. This isn't very impressive unless something is changing between the Renders. Changing this value may reset the current subframe count.
- `obj.SetSubFrames (int subFrames)` - Set the number of sub frames for doing motion blur. The default is zero. Once this is set greater than one, you will no longer see a new frame for every Render(). If you set this to five, you will need to do five Render() invocations before seeing the result. This isn't very impressive unless something is changing between the Renders. Changing this value may reset the current subframe count.
- `int = obj.GetNeverRendered ()` - This flag is set if the window hasn't rendered since it was created
- `int = obj.GetAbortRender ()` - This is a flag that can be set to interrupt a rendering that is in progress.
- `obj.SetAbortRender (int )` - This is a flag that can be set to interrupt a rendering that is in progress.
- `int = obj.GetInAbortCheck ()` - This is a flag that can be set to interrupt a rendering that is in progress.
- `obj.SetInAbortCheck (int )` - This is a flag that can be set to interrupt a rendering that is in progress.
- `int = obj.CheckAbortStatus ()` - This is a flag that can be set to interrupt a rendering that is in progress.
- `int = obj.GetIsPicking ()`
- `obj.SetIsPicking (int )`
- `obj.IsPickingOn ()`
- `obj.IsPickingOff ()`

- `int = obj.GetEventPending ()` - Check to see if a mouse button has been pressed. All other events are ignored by this method. Ideally, you want to abort the render on any event which causes the DesiredUpdateRate to switch from a high-quality rate to a more interactive rate.
- `int = obj.CheckInRenderStatus ()` - Clear status (after an exception was thrown for example)
- `obj.ClearInRenderStatus ()` - Set/Get the desired update rate. This is used with the vtkLODActor class. When using level of detail actors you need to specify what update rate you require. The LODActors then will pick the correct resolution to meet your desired update rate in frames per second. A value of zero indicates that they can use all the time they want to.
- `obj.SetDesiredUpdateRate (double )` - Set/Get the desired update rate. This is used with the vtkLODActor class. When using level of detail actors you need to specify what update rate you require. The LODActors then will pick the correct resolution to meet your desired update rate in frames per second. A value of zero indicates that they can use all the time they want to.
- `double = obj.GetDesiredUpdateRate ()` - Set/Get the desired update rate. This is used with the vtkLODActor class. When using level of detail actors you need to specify what update rate you require. The LODActors then will pick the correct resolution to meet your desired update rate in frames per second. A value of zero indicates that they can use all the time they want to.
- `int = obj.GetNumberOfLayers ()` - Get the number of layers for renderers. Each renderer should have its layer set individually. Some algorithms iterate through all layers, so it is not wise to set the number of layers to be exorbitantly large (say bigger than 100).
- `obj.SetNumberOfLayers (int )` - Get the number of layers for renderers. Each renderer should have its layer set individually. Some algorithms iterate through all layers, so it is not wise to set the number of layers to be exorbitantly large (say bigger than 100).
- `int = obj.GetNumberOfLayersMinValue ()` - Get the number of layers for renderers. Each renderer should have its layer set individually. Some algorithms iterate through all layers, so it is not wise to set the number of layers to be exorbitantly large (say bigger than 100).
- `int = obj.GetNumberOfLayersMaxValue ()` - Get the number of layers for renderers. Each renderer should have its layer set individually. Some algorithms iterate through all layers, so it is not wise to set the number of layers to be exorbitantly large (say bigger than 100).
- `vtkRenderWindowInteractor = obj.GetInteractor ()` - Get the interactor associated with this render window
- `obj.SetInteractor (vtkRenderWindowInteractor )` - Set the interactor to the render window
- `obj.UnRegister (vtkObjectBase o)` - This Method detects loops of RenderWindow¡-¿Interactor, so objects are freed properly.
- `obj.SetWindowInfo (string )` - Dummy stubs for vtkWindow API.
- `obj.SetNextWindowInfo (string )` - Dummy stubs for vtkWindow API.
- `obj.SetParentInfo (string )` - Dummy stubs for vtkWindow API.
- `obj.MakeCurrent ()` - Attempt to make this window the current graphics context for the calling thread.
- `bool = obj.IsCurrent ()` - Tells if this window is the current graphics context for the calling thread.
- `obj.SetForceMakeCurrent ()` - If called, allow MakeCurrent() to skip cache-check when called. MakeCurrent() reverts to original behavior of cache-checking on the next render.
- `string = obj.ReportCapabilities ()` - Get report of capabilities for the render window

- `int = obj.SupportsOpenGL ()` - Does this render window support OpenGL? 0-false, 1-true
- `int = obj.IsDirect ()` - Is this render window using hardware acceleration? 0-false, 1-true
- `int = obj.GetDepthBufferSize ()` - This method should be defined by the subclass. How many bits of precision are there in the zbuffer?
- `int = obj.GetColorBufferSizes (int rgba)` - Get the size of the color buffer. Returns 0 if not able to determine otherwise sets R G B and A into buffer.
- `vtkPainterDeviceAdapter = obj.GetPainterDeviceAdapter ()` - Get the vtkPainterDeviceAdapter which can be used to paint on this render window.
- `obj.SetMultiSamples (int )` - Set / Get the number of multisamples to use for hardware antialiasing.
- `int = obj.GetMultiSamples ()` - Set / Get the number of multisamples to use for hardware antialiasing.
- `obj.SetStencilCapable (int )` - Set / Get the availability of the stencil buffer.
- `int = obj.GetStencilCapable ()` - Set / Get the availability of the stencil buffer.
- `obj.StencilCapableOn ()` - Set / Get the availability of the stencil buffer.
- `obj.StencilCapableOff ()` - Set / Get the availability of the stencil buffer.
- `obj.SetReportGraphicErrors (int )` - Turn on/off report of graphic errors. Initial value is false (off). This flag is used by vtkGraphicErrorMacro.
- `int = obj.GetReportGraphicErrors ()` - Turn on/off report of graphic errors. Initial value is false (off). This flag is used by vtkGraphicErrorMacro.
- `obj.ReportGraphicErrorsOn ()` - Turn on/off report of graphic errors. Initial value is false (off). This flag is used by vtkGraphicErrorMacro.
- `obj.ReportGraphicErrorsOff ()` - Turn on/off report of graphic errors. Initial value is false (off). This flag is used by vtkGraphicErrorMacro.
- `obj.CheckGraphicError ()` - Update graphic error status, regardless of ReportGraphicErrors flag. It means this method can be used in any context and is not restricted to debug mode.
- `int = obj.HasGraphicError ()` - Return the last graphic error status. Initial value is false.
- `string = obj.GetLastGraphicErrorString ()` - Return a string matching the last graphic error status.

39.164 vtkRenderWindowCollection

39.164.1 Usage

vtkRenderWindowCollection represents and provides methods to manipulate a list of RenderWindows. The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkRenderWindowCollection, simply invoke its constructor as follows

```
obj = vtkRenderWindowCollection
```

39.164.2 Methods

The class vtkRenderWindowCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderWindowCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderWindowCollection = obj.NewInstance ()`
- `vtkRenderWindowCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkRenderWindow a)` - Get the next RenderWindow in the list. Return NULL when at the end of the list.
- `vtkRenderWindow = obj.GetNextItem ()`

39.165 vtkRenderWindowInteractor

39.165.1 Usage

vtkRenderWindowInteractor provides a platform-independent interaction mechanism for mouse/key/time events. It serves as a base class for platform-dependent implementations that handle routing of mouse/key/timer messages to vtkInteractorObserver and its subclasses. vtkRenderWindowInteractor also provides controls for picking, rendering frame rate, and headlights.

vtkRenderWindowInteractor has changed from previous implementations and now serves only as a shell to hold user preferences and route messages to vtkInteractorStyle. Callbacks are available for many events. Platform specific subclasses should provide methods for manipulating timers, TerminateApp, and an event loop if required via Initialize/Start/Enable/Disable.

To create an instance of class vtkRenderWindowInteractor, simply invoke its constructor as follows

```
obj = vtkRenderWindowInteractor
```

39.165.2 Methods

The class vtkRenderWindowInteractor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderWindowInteractor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderWindowInteractor = obj.NewInstance ()`
- `vtkRenderWindowInteractor = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Prepare for handling events. This must be called before the interactor will work.
- `obj.ReInitialize ()` - This Method detects loops of RenderWindow-Interactor, so objects are freed properly.
- `obj.UnRegister (vtkObjectBase o)` - This Method detects loops of RenderWindow-Interactor, so objects are freed properly.

- `obj.Start ()` - Enable/Disable interactions. By default interactors are enabled when initialized. Initialize() must be called prior to enabling/disabling interaction. These methods are used when a window/widget is being shared by multiple renderers and interactors. This allows a "modal" display where one interactor is active when its data is to be displayed and all other interactors associated with the widget are disabled when their data is not displayed.

- `obj.Enable ()` - Enable/Disable interactions. By default interactors are enabled when initialized. Initialize() must be called prior to enabling/disabling interaction. These methods are used when a window/widget is being shared by multiple renderers and interactors. This allows a "modal" display where one interactor is active when its data is to be displayed and all other interactors associated with the widget are disabled when their data is not displayed.

- `obj.Disable ()` - Enable/Disable interactions. By default interactors are enabled when initialized. Initialize() must be called prior to enabling/disabling interaction. These methods are used when a window/widget is being shared by multiple renderers and interactors. This allows a "modal" display where one interactor is active when its data is to be displayed and all other interactors associated with the widget are disabled when their data is not displayed.

- `int = obj.GetEnabled ()` - Enable/Disable interactions. By default interactors are enabled when initialized. Initialize() must be called prior to enabling/disabling interaction. These methods are used when a window/widget is being shared by multiple renderers and interactors. This allows a "modal" display where one interactor is active when its data is to be displayed and all other interactors associated with the widget are disabled when their data is not displayed.

- `obj.EnableRenderOn ()` - Enable/Disable whether vtkRenderWindowInteractor::Render() calls this-¿RenderWindow-¿Render().

- `obj.EnableRenderOff ()` - Enable/Disable whether vtkRenderWindowInteractor::Render() calls this-¿RenderWindow-¿Render().

- `obj.SetEnableRender (bool )` - Enable/Disable whether vtkRenderWindowInteractor::Render() calls this-¿RenderWindow-¿Render().

- `bool = obj.GetEnableRender ()` - Enable/Disable whether vtkRenderWindowInteractor::Render() calls this-¿RenderWindow-¿Render().

- `obj.SetRenderWindow (vtkRenderWindow aren)` - Set/Get the rendering window being controlled by this object.

- `vtkRenderWindow = obj.GetRenderWindow ()` - Set/Get the rendering window being controlled by this object.

- `obj.UpdateSize (int x, int y)` - Event loop notification member for window size change. Window size is measured in pixels.

- `int = obj.CreateTimer (int timerType)` - This class provides two groups of methods for manipulating timers. The first group (CreateTimer(timerType) and DestroyTimer()) implicitly use an internal timer id (and are present for backward compatibility). The second group (CreateRepeatingTimer(long),CreateOneShotTimer(long), ResetTimer(int),DestroyTimer(int)) use timer ids so multiple timers can be independently managed. In the first group, the CreateTimer() method takes an argument indicating whether the timer is created the first time (timerType==VTKI_TIMER_FIRST) or whether it is being reset (timerType==VTKI_TIMER_UPDATE). (In initial implementations of VTK this was how one shot and repeating timers were managed.) In the second group, the create methods take a timer duration argument (in milliseconds) and return a timer id. Thus the ResetTimer(timerId) and DestroyTimer(timerId) methods take this timer id and operate on the timer as appropriate. Methods are also available for determining

- `int = obj.DestroyTimer ()` - This class provides two groups of methods for manipulating timers. The first group (CreateTimer(timerType) and DestroyTimer()) implicitly use an internal timer id (and are present for backward compatibility). The second group (CreateRepeatingTimer(long),CreateOneShotTimer(long), ResetTimer(int),DestroyTimer(int)) use timer ids so multiple timers can be independently managed. In the first group, the CreateTimer() method takes an argument indicating whether the timer is created the first time (timerType==VTKI_TIMER_FIRST) or whether it is being reset (timerType==VTKI_TIMER_UPDATE (In initial implementations of VTK this was how one shot and repeating timers were managed.) In the second group, the create methods take a timer duration argument (in milliseconds) and return a timer id. Thus the ResetTimer(timerId) and DestroyTimer(timerId) methods take this timer id and operate on the timer as appropriate. Methods are also available for determining

- `int = obj.CreateRepeatingTimer (long duration)` - This class provides two groups of methods for manipulating timers. The first group (CreateTimer(timerType) and DestroyTimer()) implicitly use an internal timer id (and are present for backward compatibility). The second group (CreateRepeatingTimer(long),CreateOneShotTimer(long), ResetTimer(int),DestroyTimer(int)) use timer ids so multiple timers can be independently managed. In the first group, the CreateTimer() method takes an argument indicating whether the timer is created the first time (timerType==VTKI_TIMER_FIRST) or whether it is being reset (timerType==VTKI_TIMER_UPDATE). (In initial implementations of VTK this was how one shot and repeating timers were managed.) In the second group, the create methods take a timer duration argument (in milliseconds) and return a timer id. Thus the ResetTimer(timerId) and DestroyTimer(timerId) methods take this timer id and operate on the timer as appropriate. Methods are also available for determining

- `int = obj.CreateOneShotTimer (long duration)` - This class provides two groups of methods for manipulating timers. The first group (CreateTimer(timerType) and DestroyTimer()) implicitly use an internal timer id (and are present for backward compatibility). The second group (CreateRepeatingTimer(long),CreateOneShotTimer(long), ResetTimer(int),DestroyTimer(int)) use timer ids so multiple timers can be independently managed. In the first group, the CreateTimer() method takes an argument indicating whether the timer is created the first time (timerType==VTKI_TIMER_FIRST) or whether it is being reset (timerType==VTKI_TIMER_UPDATE). (In initial implementations of VTK this was how one shot and repeating timers were managed.) In the second group, the create methods take a timer duration argument (in milliseconds) and return a timer id. Thus the ResetTimer(timerId) and DestroyTimer(timerId) methods take this timer id and operate on the timer as appropriate. Methods are also available for determining

- `int = obj.IsOneShotTimer (int timerId)` - This class provides two groups of methods for manipulating timers. The first group (CreateTimer(timerType) and DestroyTimer()) implicitly use an internal timer id (and are present for backward compatibility). The second group (CreateRepeatingTimer(long),CreateOneShotTimer(long), ResetTimer(int),DestroyTimer(int)) use timer ids so multiple timers can be independently managed. In the first group, the CreateTimer() method takes an argument indicating whether the timer is created the first time (timerType==VTKI_TIMER_FIRST) or whether it is being reset (timerType==VTKI_TIMER_UPDATE). (In initial implementations of VTK this was how one shot and repeating timers were managed.) In the second group, the create methods take a timer duration argument (in milliseconds) and return a timer id. Thus the ResetTimer(timerId) and DestroyTimer(timerId) methods take this timer id and operate on the timer as appropriate. Methods are also available for determining

- `long = obj.GetTimerDuration (int timerId)` - This class provides two groups of methods for manipulating timers. The first group (CreateTimer(timerType) and DestroyTimer()) implicitly use an internal timer id (and are present for backward compatibility). The second group (CreateRepeatingTimer(long),CreateOneShotTimer(long), ResetTimer(int),DestroyTimer(int)) use timer ids so multiple timers can be independently managed. In the first group, the CreateTimer() method takes an argument indicating whether the timer is created the first time (timerType==VTKI_TIMER_FIRST) or whether it is being reset (timerType==VTKI_TIMER_UPDATE). (In initial implementations of VTK this was how one shot and repeating timers were managed.) In the second group, the create methods take a timer duration argument (in milliseconds) and return a timer id. Thus the ResetTimer(timerId) and

DestroyTimer(timerId) methods take this timer id and operate on the timer as appropriate. Methods are also available for determining

- `int = obj.ResetTimer (int timerId)` - This class provides two groups of methods for manipulating timers. The first group (CreateTimer(timerType) and DestroyTimer()) implicitly use an internal timer id (and are present for backward compatibility). The second group (CreateRepeatingTimer(long),CreateOneShotTimer(long), ResetTimer(int),DestroyTimer(int)) use timer ids so multiple timers can be independently managed. In the first group, the CreateTimer() method takes an argument indicating whether the timer is created the first time (timerType==VTKI_TIMER_FIRST) or whether it is being reset (timerType==VTKI_TIMER_UPDATE). (In initial implementations of VTK this was how one shot and repeating timers were managed.) In the second group, the create methods take a timer duration argument (in milliseconds) and return a timer id. Thus the ResetTimer(timerId) and DestroyTimer(timerId) methods take this timer id and operate on the timer as appropriate. Methods are also available for determining

- `int = obj.DestroyTimer (int timerId)` - This class provides two groups of methods for manipulating timers. The first group (CreateTimer(timerType) and DestroyTimer()) implicitly use an internal timer id (and are present for backward compatibility). The second group (CreateRepeatingTimer(long),CreateOneShotTimer(long), ResetTimer(int),DestroyTimer(int)) use timer ids so multiple timers can be independently managed. In the first group, the CreateTimer() method takes an argument indicating whether the timer is created the first time (timerType==VTKI_TIMER_FIRST) or whether it is being reset (timerType==VTKI_TIMER_UPDATE). (In initial implementations of VTK this was how one shot and repeating timers were managed.) In the second group, the create methods take a timer duration argument (in milliseconds) and return a timer id. Thus the ResetTimer(timerId) and DestroyTimer(timerId) methods take this timer id and operate on the timer as appropriate. Methods are also available for determining

- `int = obj.GetVTKTimerId (int platformTimerId)` - This class provides two groups of methods for manipulating timers. The first group (CreateTimer(timerType) and DestroyTimer()) implicitly use an internal timer id (and are present for backward compatibility). The second group (CreateRepeatingTimer(long),CreateOneShotTimer(long), ResetTimer(int),DestroyTimer(int)) use timer ids so multiple timers can be independently managed. In the first group, the CreateTimer() method takes an argument indicating whether the timer is created the first time (timerType==VTKI_TIMER_FIRST) or whether it is being reset (timerType==VTKI_TIMER_UPDATE). (In initial implementations of VTK this was how one shot and repeating timers were managed.) In the second group, the create methods take a timer duration argument (in milliseconds) and return a timer id. Thus the ResetTimer(timerId) and DestroyTimer(timerId) methods take this timer id and operate on the timer as appropriate. Methods are also available for determining

- `obj.SetTimerDuration (long )` - Specify the default timer interval (in milliseconds). (This is used in conjunction with the timer methods described previously, e.g., CreateTimer() uses this value; and CreateRepeatingTimer(duration) and CreateOneShotTimer(duration) use the default value if the parameter "duration" is less than or equal to zero.) Care must be taken when adjusting the timer interval from the default value of 10 milliseconds–it may adversely affect the interactors.

- `GetTimerDurationMinValue = obj.()` - Specify the default timer interval (in milliseconds). (This is used in conjunction with the timer methods described previously, e.g., CreateTimer() uses this value; and CreateRepeatingTimer(duration) and CreateOneShotTimer(duration) use the default value if the parameter "duration" is less than or equal to zero.) Care must be taken when adjusting the timer interval from the default value of 10 milliseconds–it may adversely affect the interactors.

- `GetTimerDurationMaxValue = obj.()` - Specify the default timer interval (in milliseconds). (This is used in conjunction with the timer methods described previously, e.g., CreateTimer() uses this value; and CreateRepeatingTimer(duration) and CreateOneShotTimer(duration) use the default value if the parameter "duration" is less than or equal to zero.) Care must be taken when adjusting the timer interval from the default value of 10 milliseconds–it may adversely affect the interactors.

- `long = obj.GetTimerDuration ()` - Specify the default timer interval (in milliseconds). (This is used in conjunction with the timer methods described previously, e.g., CreateTimer() uses this value; and CreateRepeatingTimer(duration) and CreateOneShotTimer(duration) use the default value if the parameter "duration" is less than or equal to zero.) Care must be taken when adjusting the timer interval from the default value of 10 milliseconds–it may adversely affect the interactors.

- `obj.SetTimerEventId (int )` - These methods are used to communicate information about the currently firing CreateTimerEvent or DestroyTimerEvent. The caller of CreateTimerEvent sets up TimerEventId, TimerEventType and TimerEventDuration. The observer of CreateTimerEvent should set up an appropriate platform specific timer based on those values and set the TimerEventPlatformId before returning. The caller of DestroyTimerEvent sets up TimerEventPlatformId. The observer of DestroyTimerEvent should simply destroy the platform specific timer created by CreateTimerEvent. See vtkGenericRenderWindowInteractor's InternalCreateTimer and InternalDestroyTimer for an example.

- `int = obj.GetTimerEventId ()` - These methods are used to communicate information about the currently firing CreateTimerEvent or DestroyTimerEvent. The caller of CreateTimerEvent sets up TimerEventId, TimerEventType and TimerEventDuration. The observer of CreateTimerEvent should set up an appropriate platform specific timer based on those values and set the TimerEventPlatformId before returning. The caller of DestroyTimerEvent sets up TimerEventPlatformId. The observer of DestroyTimerEvent should simply destroy the platform specific timer created by CreateTimerEvent. See vtkGenericRenderWindowInteractor's InternalCreateTimer and InternalDestroyTimer for an example.

- `obj.SetTimerEventType (int )` - These methods are used to communicate information about the currently firing CreateTimerEvent or DestroyTimerEvent. The caller of CreateTimerEvent sets up TimerEventId, TimerEventType and TimerEventDuration. The observer of CreateTimerEvent should set up an appropriate platform specific timer based on those values and set the TimerEventPlatformId before returning. The caller of DestroyTimerEvent sets up TimerEventPlatformId. The observer of DestroyTimerEvent should simply destroy the platform specific timer created by CreateTimerEvent. See vtkGenericRenderWindowInteractor's InternalCreateTimer and InternalDestroyTimer for an example.

- `int = obj.GetTimerEventType ()` - These methods are used to communicate information about the currently firing CreateTimerEvent or DestroyTimerEvent. The caller of CreateTimerEvent sets up TimerEventId, TimerEventType and TimerEventDuration. The observer of CreateTimerEvent should set up an appropriate platform specific timer based on those values and set the TimerEventPlatformId before returning. The caller of DestroyTimerEvent sets up TimerEventPlatformId. The observer of DestroyTimerEvent should simply destroy the platform specific timer created by CreateTimerEvent. See vtkGenericRenderWindowInteractor's InternalCreateTimer and InternalDestroyTimer for an example.

- `obj.SetTimerEventDuration (int )` - These methods are used to communicate information about the currently firing CreateTimerEvent or DestroyTimerEvent. The caller of CreateTimerEvent sets up TimerEventId, TimerEventType and TimerEventDuration. The observer of CreateTimerEvent should set up an appropriate platform specific timer based on those values and set the TimerEventPlatformId before returning. The caller of DestroyTimerEvent sets up TimerEventPlatformId. The observer of DestroyTimerEvent should simply destroy the platform specific timer created by CreateTimerEvent. See vtkGenericRenderWindowInteractor's InternalCreateTimer and InternalDestroyTimer for an example.

- `int = obj.GetTimerEventDuration ()` - These methods are used to communicate information about the currently firing CreateTimerEvent or DestroyTimerEvent. The caller of CreateTimerEvent sets up TimerEventId, TimerEventType and TimerEventDuration. The observer of CreateTimerEvent should set up an appropriate platform specific timer based on those values and set the TimerEventPlatformId before returning. The caller of DestroyTimerEvent sets up TimerEventPlatformId. The observer of DestroyTimerEvent should simply destroy the platform specific timer created by CreateTimerEvent. See vtkGenericRenderWindowInteractor's InternalCreateTimer and InternalDestroyTimer for an example.

- `obj.SetTimerEventPlatformId (int )` - These methods are used to communicate information about the currently firing CreateTimerEvent or DestroyTimerEvent. The caller of CreateTimerEvent sets up TimerEventId, TimerEventType and TimerEventDuration. The observer of CreateTimerEvent should

set up an appropriate platform specific timer based on those values and set the TimerEventPlatformId before returning. The caller of DestroyTimerEvent sets up TimerEventPlatformId. The observer of DestroyTimerEvent should simply destroy the platform specific timer created by CreateTimerEvent. See vtkGenericRenderWindowInteractor's InternalCreateTimer and InternalDestroyTimer for an example.

- `int = obj.GetTimerEventPlatformId ()` - These methods are used to communicate information about the currently firing CreateTimerEvent or DestroyTimerEvent. The caller of CreateTimerEvent sets up TimerEventId, TimerEventType and TimerEventDuration. The observer of CreateTimerEvent should set up an appropriate platform specific timer based on those values and set the TimerEventPlatformId before returning. The caller of DestroyTimerEvent sets up TimerEventPlatformId. The observer of DestroyTimerEvent should simply destroy the platform specific timer created by CreateTimerEvent. See vtkGenericRenderWindowInteractor's InternalCreateTimer and InternalDestroyTimer for an example.

- `obj.TerminateApp (void )` - External switching between joystick/trackball/new? modes. Initial value is a vtkInteractorStyleSwitch object.

- `obj.SetInteractorStyle (vtkInteractorObserver )` - External switching between joystick/trackball/new? modes. Initial value is a vtkInteractorStyleSwitch object.

- `vtkInteractorObserver = obj.GetInteractorStyle ()` - External switching between joystick/trackball/new? modes. Initial value is a vtkInteractorStyleSwitch object.

- `obj.SetLightFollowCamera (int )` - Turn on/off the automatic repositioning of lights as the camera moves. Default is On.

- `int = obj.GetLightFollowCamera ()` - Turn on/off the automatic repositioning of lights as the camera moves. Default is On.

- `obj.LightFollowCameraOn ()` - Turn on/off the automatic repositioning of lights as the camera moves. Default is On.

- `obj.LightFollowCameraOff ()` - Turn on/off the automatic repositioning of lights as the camera moves. Default is On.

- `obj.SetDesiredUpdateRate (double )` - Set/Get the desired update rate. This is used by vtkLODActor's to tell them how quickly they need to render. This update is in effect only when the camera is being rotated, or zoomed. When the interactor is still, the StillUpdateRate is used instead. The default is 15.

- `double = obj.GetDesiredUpdateRateMinValue ()` - Set/Get the desired update rate. This is used by vtkLODActor's to tell them how quickly they need to render. This update is in effect only when the camera is being rotated, or zoomed. When the interactor is still, the StillUpdateRate is used instead. The default is 15.

- `double = obj.GetDesiredUpdateRateMaxValue ()` - Set/Get the desired update rate. This is used by vtkLODActor's to tell them how quickly they need to render. This update is in effect only when the camera is being rotated, or zoomed. When the interactor is still, the StillUpdateRate is used instead. The default is 15.

- `double = obj.GetDesiredUpdateRate ()` - Set/Get the desired update rate. This is used by vtkLODActor's to tell them how quickly they need to render. This update is in effect only when the camera is being rotated, or zoomed. When the interactor is still, the StillUpdateRate is used instead. The default is 15.

- `obj.SetStillUpdateRate (double )` - Set/Get the desired update rate when movement has stopped. For the non-still update rate, see the SetDesiredUpdateRate method. The default is 0.0001

- `double = obj.GetStillUpdateRateMinValue ()` - Set/Get the desired update rate when movement has stopped. For the non-still update rate, see the SetDesiredUpdateRate method. The default is 0.0001
- `double = obj.GetStillUpdateRateMaxValue ()` - Set/Get the desired update rate when movement has stopped. For the non-still update rate, see the SetDesiredUpdateRate method. The default is 0.0001
- `double = obj.GetStillUpdateRate ()` - Set/Get the desired update rate when movement has stopped. For the non-still update rate, see the SetDesiredUpdateRate method. The default is 0.0001
- `int = obj.GetInitialized ()` - See whether interactor has been initialized yet. Default is 0.
- `obj.SetPicker (vtkAbstractPicker )` - Set/Get the object used to perform pick operations. In order to pick instances of vtkProp, the picker must be a subclass of vtkAbstractPropPicker, meaning that it can identify a particular instance of vtkProp.
- `vtkAbstractPicker = obj.GetPicker ()` - Set/Get the object used to perform pick operations. In order to pick instances of vtkProp, the picker must be a subclass of vtkAbstractPropPicker, meaning that it can identify a particular instance of vtkProp.
- `vtkAbstractPropPicker = obj.CreateDefaultPicker ()` - Create default picker. Used to create one when none is specified. Default is an instance of vtkPropPicker.
- `obj.ExitCallback ()` - These methods correspond to the the Exit, User and Pick callbacks. They allow for the Style to invoke them.
- `obj.UserCallback ()` - These methods correspond to the the Exit, User and Pick callbacks. They allow for the Style to invoke them.
- `obj.StartPickCallback ()` - These methods correspond to the the Exit, User and Pick callbacks. They allow for the Style to invoke them.
- `obj.EndPickCallback ()` - These methods correspond to the the Exit, User and Pick callbacks. They allow for the Style to invoke them.
- `obj.GetMousePosition (int x, int y)` - Hide or show the mouse cursor, it is nice to be able to hide the default cursor if you want VTK to display a 3D cursor instead.
- `obj.HideCursor ()` - Hide or show the mouse cursor, it is nice to be able to hide the default cursor if you want VTK to display a 3D cursor instead.
- `obj.ShowCursor ()` - Hide or show the mouse cursor, it is nice to be able to hide the default cursor if you want VTK to display a 3D cursor instead.
- `obj.Render ()` - Render the scene. Just pass the render call on to the associated vtkRenderWindow.
- `obj.FlyTo (vtkRenderer ren, double x, double y, double z)` - Given a position x, move the current camera's focal point to x. The movement is animated over the number of frames specified in NumberOfFlyFrames. The LOD desired frame rate is used.
- `obj.FlyTo (vtkRenderer ren, double x)` - Given a position x, move the current camera's focal point to x. The movement is animated over the number of frames specified in NumberOfFlyFrames. The LOD desired frame rate is used.
- `obj.FlyToImage (vtkRenderer ren, double x, double y)` - Given a position x, move the current camera's focal point to x. The movement is animated over the number of frames specified in NumberOfFlyFrames. The LOD desired frame rate is used.
- `obj.FlyToImage (vtkRenderer ren, double x)` - Set the number of frames to fly to when FlyTo is invoked.

- `obj.SetNumberOfFlyFrames (int )` - Set the number of frames to fly to when FlyTo is invoked.
- `int = obj.GetNumberOfFlyFramesMinValue ()` - Set the number of frames to fly to when FlyTo is invoked.
- `int = obj.GetNumberOfFlyFramesMaxValue ()` - Set the number of frames to fly to when FlyTo is invoked.
- `int = obj.GetNumberOfFlyFrames ()` - Set the number of frames to fly to when FlyTo is invoked.
- `obj.SetDolly (double )` - Set the total Dolly value to use when flying to (FlyTo()) a specified point. Negative values fly away from the point.
- `double = obj.GetDolly ()` - Set the total Dolly value to use when flying to (FlyTo()) a specified point. Negative values fly away from the point.
- `int = obj. GetEventPosition ()` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `int = obj. GetLastEventPosition ()` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetLastEventPosition (int , int )` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetLastEventPosition (int  a[2])` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetEventPosition (int x, int y)` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetEventPosition (int pos[2])` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetEventPositionFlipY (int x, int y)` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetEventPositionFlipY (int pos[2])` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetAltKey (int )` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.

- `int = obj.GetAltKey ()` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetControlKey (int )` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `int = obj.GetControlKey ()` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetShiftKey (int )` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `int = obj.GetShiftKey ()` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetKeyCode (char )` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `char = obj.GetKeyCode ()` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetRepeatCount (int )` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `int = obj.GetRepeatCount ()` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetKeySym (string )` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `string = obj.GetKeySym ()` - Set/Get information about the current event. The current x,y position is in the EventPosition, and the previous event position is in LastEventPosition, updated automatically each time EventPosition is set using its Set() method. Mouse positions are measured in pixels. The other information is about key board input.
- `obj.SetEventInformation (int x, int y, int ctrl, int shift, char keycode, int repeatcount, string k` - Calls SetEventInformation, but flips the Y based on the current Size[1] value (i.e. y = this-¿Size[1] - y - 1).

- `obj.SetEventInformationFlipY (int x, int y, int ctrl, int shift, char keycode, int repeatcount, str` - Set all the keyboard-related event information in one call.

- `obj.SetKeyEventInformation (int ctrl, int shift, char keycode, int repeatcount, string keysym)` - This methods sets the Size ivar of the interactor without actually changing the size of the window. Normally application programmers would use UpdateSize if anything. This is useful for letting someone else change the size of the rendering window and just letting the interactor know about the change. The current event width/height (if any) is in EventSize (Expose event, for example). Window size is measured in pixels.

- `obj.SetSize (int , int )` - This methods sets the Size ivar of the interactor without actually changing the size of the window. Normally application programmers would use UpdateSize if anything. This is useful for letting someone else change the size of the rendering window and just letting the interactor know about the change. The current event width/height (if any) is in EventSize (Expose event, for example). Window size is measured in pixels.

- `obj.SetSize (int a[2])` - This methods sets the Size ivar of the interactor without actually changing the size of the window. Normally application programmers would use UpdateSize if anything. This is useful for letting someone else change the size of the rendering window and just letting the interactor know about the change. The current event width/height (if any) is in EventSize (Expose event, for example). Window size is measured in pixels.

- `int = obj. GetSize ()` - This methods sets the Size ivar of the interactor without actually changing the size of the window. Normally application programmers would use UpdateSize if anything. This is useful for letting someone else change the size of the rendering window and just letting the interactor know about the change. The current event width/height (if any) is in EventSize (Expose event, for example). Window size is measured in pixels.

- `obj.SetEventSize (int , int )` - This methods sets the Size ivar of the interactor without actually changing the size of the window. Normally application programmers would use UpdateSize if anything. This is useful for letting someone else change the size of the rendering window and just letting the interactor know about the change. The current event width/height (if any) is in EventSize (Expose event, for example). Window size is measured in pixels.

- `obj.SetEventSize (int a[2])` - This methods sets the Size ivar of the interactor without actually changing the size of the window. Normally application programmers would use UpdateSize if anything. This is useful for letting someone else change the size of the rendering window and just letting the interactor know about the change. The current event width/height (if any) is in EventSize (Expose event, for example). Window size is measured in pixels.

- `int = obj. GetEventSize ()` - This methods sets the Size ivar of the interactor without actually changing the size of the window. Normally application programmers would use UpdateSize if anything. This is useful for letting someone else change the size of the rendering window and just letting the interactor know about the change. The current event width/height (if any) is in EventSize (Expose event, for example). Window size is measured in pixels.

- `vtkRenderer = obj.FindPokedRenderer (int , int )` - When an event occurs, we must determine which Renderer the event occurred within, since one RenderWindow may contain multiple renderers.

- `vtkObserverMediator = obj.GetObserverMediator ()` - Return the object used to mediate between vtkInteractorObservers contending for resources. Multiple interactor observers will often request different resources (e.g., cursor shape); the mediator uses a strategy to provide the resource based on priority of the observer plus the particular request (default versus non-default cursor shape).

- `obj.SetUseTDx (bool )` - Use a 3DConnexion device. Initial value is false. If VTK is not build with the TDx option, this is no-op. If VTK is build with the TDx option, and a device is not connected, a warning is emitted. It is must be called before the first Render to be effective, otherwise it is ignored.

- `bool = obj.GetUseTDx ()` - Use a 3DConnexion device. Initial value is false. If VTK is not build with the TDx option, this is no-op. If VTK is build with the TDx option, and a device is not connected, a warning is emitted. It is must be called before the first Render to be effective, otherwise it is ignored.

39.166 vtkRepresentationPainter

39.166.1 Usage

This painter merely defines the interface. Subclasses will change the polygon rendering mode dependent on the graphics library.

To create an instance of class vtkRepresentationPainter, simply invoke its constructor as follows

```
obj = vtkRepresentationPainter
```

39.166.2 Methods

The class vtkRepresentationPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRepresentationPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRepresentationPainter = obj.NewInstance ()`
- `vtkRepresentationPainter = obj.SafeDownCast (vtkObject o)`

39.167 vtkScalarBarActor

39.167.1 Usage

vtkScalarBarActor creates a scalar bar with annotation text. A scalar bar is a legend that indicates to the viewer the correspondence between color value and data value. The legend consists of a rectangular bar made of rectangular pieces each colored a constant value. Since vtkScalarBarActor is a subclass of vtkActor2D, it is drawn in the image plane (i.e., in the renderer's viewport) on top of the 3D graphics window.

To use vtkScalarBarActor you must associate a vtkScalarsToColors (or subclass) with it. The lookup table defines the colors and the range of scalar values used to map scalar data. Typically, the number of colors shown in the scalar bar is not equal to the number of colors in the lookup table, in which case sampling of the lookup table is performed.

Other optional capabilities include specifying the fraction of the viewport size (both x and y directions) which will control the size of the scalar bar and the number of annotation labels. The actual position of the scalar bar on the screen is controlled by using the vtkActor2D::SetPosition() method (by default the scalar bar is centered in the viewport). Other features include the ability to orient the scalar bar horizontally of vertically and controlling the format (printf style) with which to print the labels on the scalar bar. Also, the vtkScalarBarActor's property is applied to the scalar bar and annotation (including layer, and compositing operator).

Set the text property/attributes of the title and the labels through the vtkTextProperty objects associated to this actor.

To create an instance of class vtkScalarBarActor, simply invoke its constructor as follows

```
obj = vtkScalarBarActor
```

39.167.2 Methods

The class vtkScalarBarActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkScalarBarActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkScalarBarActor = obj.NewInstance ()`
- `vtkScalarBarActor = obj.SafeDownCast (vtkObject o)`
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Draw the scalar bar and annotation text to the screen.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Draw the scalar bar and annotation text to the screen.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Draw the scalar bar and annotation text to the screen.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Does this prop have some translucent polygonal geometry?
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.SetLookupTable (vtkScalarsToColors )` - Set/Get the vtkLookupTable to use. The lookup table specifies the number of colors to use in the table (if not overridden), as well as the scalar range.
- `vtkScalarsToColors = obj.GetLookupTable ()` - Set/Get the vtkLookupTable to use. The lookup table specifies the number of colors to use in the table (if not overridden), as well as the scalar range.
- `obj.SetUseOpacity (int )` - Should be display the opacity as well. This is displayed by changing the opacity of the scalar bar in accordance with the opacity of the given color. For clarity, a texture grid is placed in the background if Opacity is ON. You might also want to play with SetTextureGridWith in that case. [Default: off]
- `int = obj.GetUseOpacity ()` - Should be display the opacity as well. This is displayed by changing the opacity of the scalar bar in accordance with the opacity of the given color. For clarity, a texture grid is placed in the background if Opacity is ON. You might also want to play with SetTextureGridWith in that case. [Default: off]
- `obj.UseOpacityOn ()` - Should be display the opacity as well. This is displayed by changing the opacity of the scalar bar in accordance with the opacity of the given color. For clarity, a texture grid is placed in the background if Opacity is ON. You might also want to play with SetTextureGridWith in that case. [Default: off]
- `obj.UseOpacityOff ()` - Should be display the opacity as well. This is displayed by changing the opacity of the scalar bar in accordance with the opacity of the given color. For clarity, a texture grid is placed in the background if Opacity is ON. You might also want to play with SetTextureGridWith in that case. [Default: off]
- `obj.SetMaximumNumberOfColors (int )` - Set/Get the maximum number of scalar bar segments to show. This may differ from the number of colors in the lookup table, in which case the colors are samples from the lookup table.

- `int = obj.GetMaximumNumberOfColorsMinValue ()` - Set/Get the maximum number of scalar bar segments to show. This may differ from the number of colors in the lookup table, in which case the colors are samples from the lookup table.
- `int = obj.GetMaximumNumberOfColorsMaxValue ()` - Set/Get the maximum number of scalar bar segments to show. This may differ from the number of colors in the lookup table, in which case the colors are samples from the lookup table.
- `int = obj.GetMaximumNumberOfColors ()` - Set/Get the maximum number of scalar bar segments to show. This may differ from the number of colors in the lookup table, in which case the colors are samples from the lookup table.
- `obj.SetNumberOfLabels (int )` - Set/Get the number of annotation labels to show.
- `int = obj.GetNumberOfLabelsMinValue ()` - Set/Get the number of annotation labels to show.
- `int = obj.GetNumberOfLabelsMaxValue ()` - Set/Get the number of annotation labels to show.
- `int = obj.GetNumberOfLabels ()` - Set/Get the number of annotation labels to show.
- `obj.SetOrientation (int )` - Control the orientation of the scalar bar.
- `int = obj.GetOrientationMinValue ()` - Control the orientation of the scalar bar.
- `int = obj.GetOrientationMaxValue ()` - Control the orientation of the scalar bar.
- `int = obj.GetOrientation ()` - Control the orientation of the scalar bar.
- `obj.SetOrientationToHorizontal ()` - Control the orientation of the scalar bar.
- `obj.SetOrientationToVertical ()` - Control the orientation of the scalar bar.
- `obj.SetTitleTextProperty (vtkTextProperty p)` - Set/Get the title text property.
- `vtkTextProperty = obj.GetTitleTextProperty ()` - Set/Get the title text property.
- `obj.SetLabelTextProperty (vtkTextProperty p)` - Set/Get the labels text property.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - Set/Get the labels text property.
- `obj.SetLabelFormat (string )` - Set/Get the format with which to print the labels on the scalar bar.
- `string = obj.GetLabelFormat ()` - Set/Get the format with which to print the labels on the scalar bar.
- `obj.SetTitle (string )` - Set/Get the title of the scalar bar actor,
- `string = obj.GetTitle ()` - Set/Get the title of the scalar bar actor,
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of a scalar bar actor. Overloads the virtual vtkProp method.
- `obj.SetTextureGridWidth (double )` - Set the width of the texture grid. Used only if UseOpacity is ON.
- `double = obj.GetTextureGridWidth ()` - Set the width of the texture grid. Used only if UseOpacity is ON.
- `vtkActor2D = obj.GetTextureActor ()` - Get the texture actor.. you may want to change some properties on it

- `obj.SetTextPosition (int )` - Have the text preceding the scalar bar or suceeding it ? Succeed implies the that the text is Above scalar bar if orientation is horizontal or Right of scalar bar if orientation is vertical. Precede is the opposite

- `int = obj.GetTextPositionMinValue ()` - Have the text preceding the scalar bar or suceeding it ? Succeed implies the that the text is Above scalar bar if orientation is horizontal or Right of scalar bar if orientation is vertical. Precede is the opposite

- `int = obj.GetTextPositionMaxValue ()` - Have the text preceding the scalar bar or suceeding it ? Succeed implies the that the text is Above scalar bar if orientation is horizontal or Right of scalar bar if orientation is vertical. Precede is the opposite

- `int = obj.GetTextPosition ()` - Have the text preceding the scalar bar or suceeding it ? Succeed implies the that the text is Above scalar bar if orientation is horizontal or Right of scalar bar if orientation is vertical. Precede is the opposite

- `obj.SetTextPositionToPrecedeScalarBar ()` - Have the text preceding the scalar bar or suceeding it ? Succeed implies the that the text is Above scalar bar if orientation is horizontal or Right of scalar bar if orientation is vertical. Precede is the opposite

- `obj.SetTextPositionToSucceedScalarBar ()` - Set/Get the maximum width and height in pixels. Specifying the size as a relative fraction of the viewport can sometimes undersirably strech the size of the actor too much. These methods allow the user to set bounds on the maximum size of the scalar bar in pixels along any direction. Defaults to unbounded.

- `obj.SetMaximumWidthInPixels (int )` - Set/Get the maximum width and height in pixels. Specifying the size as a relative fraction of the viewport can sometimes undersirably strech the size of the actor too much. These methods allow the user to set bounds on the maximum size of the scalar bar in pixels along any direction. Defaults to unbounded.

- `int = obj.GetMaximumWidthInPixels ()` - Set/Get the maximum width and height in pixels. Specifying the size as a relative fraction of the viewport can sometimes undersirably strech the size of the actor too much. These methods allow the user to set bounds on the maximum size of the scalar bar in pixels along any direction. Defaults to unbounded.

- `obj.SetMaximumHeightInPixels (int )` - Set/Get the maximum width and height in pixels. Specifying the size as a relative fraction of the viewport can sometimes undersirably strech the size of the actor too much. These methods allow the user to set bounds on the maximum size of the scalar bar in pixels along any direction. Defaults to unbounded.

- `int = obj.GetMaximumHeightInPixels ()` - Set/Get the maximum width and height in pixels. Specifying the size as a relative fraction of the viewport can sometimes undersirably strech the size of the actor too much. These methods allow the user to set bounds on the maximum size of the scalar bar in pixels along any direction. Defaults to unbounded.

39.168 vtkScalarsToColorsPainter

39.168.1 Usage

This is a painter that converts scalars to colors. It enable/disables coloring state depending on the ScalarMode. This painter is composite dataset enabled.

To create an instance of class vtkScalarsToColorsPainter, simply invoke its constructor as follows

```
obj = vtkScalarsToColorsPainter
```

39.168.2 Methods

The class vtkScalarsToColorsPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkScalarsToColorsPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkScalarsToColorsPainter = obj.NewInstance ()`
- `vtkScalarsToColorsPainter = obj.SafeDownCast (vtkObject o)`
- `obj.SetLookupTable (vtkScalarsToColors lut)` - Specify a lookup table for the mapper to use.
- `vtkScalarsToColors = obj.GetLookupTable ()` - Specify a lookup table for the mapper to use.
- `obj.CreateDefaultLookupTable ()` - Create default lookup table. Generally used to create one when none is available with the scalar data.
- `int = obj.GetPremultiplyColorsWithAlpha (vtkActor actor)` - For alpha blending, we sometime premultiply the colors with alpha and change the alpha blending function. This call returns whether we are premultiplying or using the default blending function. Currently this checks if the actor has a texture, if not it returns true. TODO: It is possible to make this decision dependent on key passed down from a painter upstream that makes a more informed decision for alpha blending depending on extensions available, for example.
- `vtkDataObject = obj.GetOutput ()` - Subclasses need to override this to return the output of the pipeline.

39.169 vtkScaledTextActor

39.169.1 Usage

vtkScaledTextActor is deprecated. New code should use vtkTextActor with the Scaled = true option.

To create an instance of class vtkScaledTextActor, simply invoke its constructor as follows

```
obj = vtkScaledTextActor
```

39.169.2 Methods

The class vtkScaledTextActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkScaledTextActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkScaledTextActor = obj.NewInstance ()`
- `vtkScaledTextActor = obj.SafeDownCast (vtkObject o)`

39.170 vtkScenePicker

39.170.1 Usage

The Scene picker, unline conventional pickers picks an entire viewport at one shot and caches the result, which can be retrieved later. The utility of the class arises during ¡b¿Actor Selection¡/b¿. Let's say you have a couple of polygonal objects in your scene and you wish to have a status bar that indicates the object your mouse is over. Picking repeatedly every time your mouse moves would be very slow. The scene picker automatically picks your viewport every time the camera is changed and caches the information. Additionally, it observes the vtkRenderWindowInteractor to avoid picking during interaction, so that you still maintain your interactivity. In effect, the picker does an additional pick-render of your scene every time you stop interacting with your scene. As an example, see Rendering/TestScenePicker.

To create an instance of class vtkScenePicker, simply invoke its constructor as follows

```
obj = vtkScenePicker
```

39.170.2 Methods

The class vtkScenePicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkScenePicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkScenePicker = obj.NewInstance ()`
- `vtkScenePicker = obj.SafeDownCast (vtkObject o)`
- `obj.SetRenderer (vtkRenderer )` - Set the renderer. Scene picks are restricted to the viewport.
- `vtkRenderer = obj.GetRenderer ()` - Set the renderer. Scene picks are restricted to the viewport.
- `vtkIdType = obj.GetCellId (int displayPos[2])` - Get cell id at the pick position. Returns -1 if no cell was picked. Makes sense only after Pick has been called.
- `vtkIdType = obj.GetVertexId (int displayPos[2])` - Get cell id at the pick position. Returns -1 if no cell was picked. Makes sense only after Pick has been called.
- `vtkProp = obj.GetViewProp (int displayPos[2])` - Get actor at the pick position. Returns NULL if none. Makes sense only after Pick has been called.
- `obj.SetEnableVertexPicking (int )` - Vertex picking (using the method GetVertexId()), required additional resources and can slow down still render time by 5-10
- `int = obj.GetEnableVertexPicking ()` - Vertex picking (using the method GetVertexId()), required additional resources and can slow down still render time by 5-10
- `obj.EnableVertexPickingOn ()` - Vertex picking (using the method GetVertexId()), required additional resources and can slow down still render time by 5-10
- `obj.EnableVertexPickingOff ()` - Vertex picking (using the method GetVertexId()), required additional resources and can slow down still render time by 5-10

39.171 vtkSelectVisiblePoints

39.171.1 Usage

vtkSelectVisiblePoints is a filter that selects points based on whether they are visible or not. Visibility is determined by accessing the z-buffer of a rendering window. (The position of each input point is converted into display coordinates, and then the z-value at that point is obtained. If within the user-specified tolerance, the point is considered visible.)

Points that are visible (or if the ivar SelectInvisible is on, invisible points) are passed to the output. Associated data attributes are passed to the output as well.

This filter also allows you to specify a rectangular window in display (pixel) coordinates in which the visible points must lie. This can be used as a sort of local "brushing" operation to select just data within a window.

To create an instance of class vtkSelectVisiblePoints, simply invoke its constructor as follows

```
obj = vtkSelectVisiblePoints
```

39.171.2 Methods

The class vtkSelectVisiblePoints has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSelectVisiblePoints class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSelectVisiblePoints = obj.NewInstance ()`
- `vtkSelectVisiblePoints = obj.SafeDownCast (vtkObject o)`
- `obj.SetRenderer (vtkRenderer ren)` - Specify the renderer in which the visibility computation is to be performed.
- `vtkRenderer = obj.GetRenderer ()` - Set/Get the flag which enables selection in a rectangular display region.
- `obj.SetSelectionWindow (int )` - Set/Get the flag which enables selection in a rectangular display region.
- `int = obj.GetSelectionWindow ()` - Set/Get the flag which enables selection in a rectangular display region.
- `obj.SelectionWindowOn ()` - Set/Get the flag which enables selection in a rectangular display region.
- `obj.SelectionWindowOff ()` - Set/Get the flag which enables selection in a rectangular display region.
- `obj.SetSelection (int , int , int , int )` - Specify the selection window in display coordinates. You must specify a rectangular region using (xmin,xmax,ymin,ymax).
- `obj.SetSelection (int  a[4])` - Specify the selection window in display coordinates. You must specify a rectangular region using (xmin,xmax,ymin,ymax).
- `int = obj. GetSelection ()` - Specify the selection window in display coordinates. You must specify a rectangular region using (xmin,xmax,ymin,ymax).
- `obj.SetSelectInvisible (int )` - Set/Get the flag which enables inverse selection; i.e., invisible points are selected.

- `int = obj.GetSelectInvisible ()` - Set/Get the flag which enables inverse selection; i.e., invisible points are selected.
- `obj.SelectInvisibleOn ()` - Set/Get the flag which enables inverse selection; i.e., invisible points are selected.
- `obj.SelectInvisibleOff ()` - Set/Get the flag which enables inverse selection; i.e., invisible points are selected.
- `obj.SetTolerance (double )` - Set/Get a tolerance to use to determine whether a point is visible. A tolerance is usually required because the conversion from world space to display space during rendering introduces numerical round-off.
- `double = obj.GetToleranceMinValue ()` - Set/Get a tolerance to use to determine whether a point is visible. A tolerance is usually required because the conversion from world space to display space during rendering introduces numerical round-off.
- `double = obj.GetToleranceMaxValue ()` - Set/Get a tolerance to use to determine whether a point is visible. A tolerance is usually required because the conversion from world space to display space during rendering introduces numerical round-off.
- `double = obj.GetTolerance ()` - Set/Get a tolerance to use to determine whether a point is visible. A tolerance is usually required because the conversion from world space to display space during rendering introduces numerical round-off.
- `bool = obj.IsPointOccluded (double x[], float zPtr)` - Tests if a point x is being occluded or not against the Z-Buffer array passed in by zPtr. Call Initialize before calling this method.
- `long = obj.GetMTime ()` - Return MTime also considering the renderer.

39.172 vtkSequencePass

39.172.1 Usage

vtkSequencePass executes a list of render passes sequentially. This class allows to define a sequence of render passes at run time. The other solution to write a sequence of render passes is to write an effective subclass of vtkRenderPass.

As vtkSequencePass is a vtkRenderPass itself, it is possible to have a hierarchy of render passes built at runtime.

To create an instance of class vtkSequencePass, simply invoke its constructor as follows

```
obj = vtkSequencePass
```

39.172.2 Methods

The class vtkSequencePass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSequencePass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSequencePass = obj.NewInstance ()`
- `vtkSequencePass = obj.SafeDownCast (vtkObject o)`

- `obj.ReleaseGraphicsResources (vtkWindow w)` - Release graphics resources and ask components to release their own resources.
- `vtkRenderPassCollection = obj.GetPasses ()` - The ordered list of render passes to execute sequentially. If the pointer is NULL or the list is empty, it is silently ignored. There is no warning. Initial value is a NULL pointer.
- `obj.SetPasses (vtkRenderPassCollection passes)` - The ordered list of render passes to execute sequentially. If the pointer is NULL or the list is empty, it is silently ignored. There is no warning. Initial value is a NULL pointer.

39.173 vtkShader

39.173.1 Usage

vtkShader is a base class for interfacing VTK to hardware shader libraries. vtkShader interprets a vtkXMLDataElement that describes a particular shader. Descendants of this class inherit this functionality and additionally interface to specific shader libraries like NVidia's Cg and OpenGL2.0 (GLSL) to perform operations, on individual shaders.

During each render, the vtkShaderProgram calls Compile(), PassShaderVariables(), Bind() and after the actor has been rendered, calls Unbind(), in that order.

To create an instance of class vtkShader, simply invoke its constructor as follows

```
obj = vtkShader
```

39.173.2 Methods

The class vtkShader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkShader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkShader = obj.NewInstance ()`
- `vtkShader = obj.SafeDownCast (vtkObject o)`
- `int = obj.Compile ()` - Called to compile the shader code. The subclasses must only compile the code in this method. Returns if the compile was successful. Subclasses should compile the code only if it was not already compiled.
- `obj.PassShaderVariables (vtkActor actor, vtkRenderer ren)` - Called to pass VTK actor/property/light values and other Shader variables over to the shader. This is called by the ShaderProgram during each render.
- `obj.Bind ()` - Called to unbind the shader. As with Bind(), this is only applicable to Cg.
- `obj.Unbind ()` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Get/Set the XMLShader representation for this shader. A shader is not valid without a XMLShader.
- `obj.SetXMLShader (vtkXMLShader )` - Get/Set the XMLShader representation for this shader. A shader is not valid without a XMLShader.

- `vtkXMLShader = obj.GetXMLShader ()` - Get/Set the XMLShader representation for this shader. A shader is not valid without a XMLShader.
- `int = obj.HasShaderVariable (string name)` - Indicates if a variable by the given name exists.
- `obj.AddShaderVariable (string name, int num\_of\_elements, int values)` - Methods to add shader variables to this shader. The shader variable type must match with that declared in the Material xml, otherwise, the variable is not made available to the shader.
- `obj.AddShaderVariable (string name, int num\_of\_elements, float values)` - Methods to add shader variables to this shader. The shader variable type must match with that declared in the Material xml, otherwise, the variable is not made available to the shader.
- `obj.AddShaderVariable (string name, int num\_of\_elements, double values)` - Methods to add shader variables to this shader. The shader variable type must match with that declared in the Material xml, otherwise, the variable is not made available to the shader.
- `int = obj.GetShaderVariableSize (string name)` - Get number of elements in a Shader variable. Return 0 if failed to find the shader variable.
- `int = obj.GetShaderVariableType (string name)` - Returns the type of a Shader variable with the given name. Return 0 on error.
- `int = obj.GetShaderVariable (string name, int values)` - Methods to get the value of shader variables with the given name. Values must be at least the size of the shader variable (obtained by GetShaderVariableSize(). Returns if the operation was successful.
- `int = obj.GetShaderVariable (string name, float values)` - Methods to get the value of shader variables with the given name. Values must be at least the size of the shader variable (obtained by GetShaderVariableSize(). Returns if the operation was successful.
- `int = obj.GetShaderVariable (string name, double values)` - Methods to get the value of shader variables with the given name. Values must be at least the size of the shader variable (obtained by GetShaderVariableSize(). Returns if the operation was successful.
- `int = obj.GetScope ()` - Returns the scope of the shader i.e. if it's a vertex or fragment shader. (vtkXMLShader::SCOPE_VERTEX or vtkXMLShader::SCOPE_FRAGMENT).

39.174 vtkShaderProgram

39.174.1 Usage

vtkShaderProgram is a superclass for managing Hardware Shaders defined in the XML Material file and interfacing VTK to those shaders. It's concrete descendants are responsible for installing vertex and fragment programs to the graphics hardware.

.SECTION Shader Operations are shader library operations that are performed on individual shaders, that is, without consideration of the partner shader.

.SECTION Program Operations are shader library operations that treat the vertex and fragment shader as a single unit.

.SECTION Design This class is a Strategy pattern for 'Program' operations, which treat vertex/fragment shader pairs as a single 'Program', as required by some shader libraries (GLSL). Typically, 'Shader' operations are delegated to instances of vtkShader (managed by descendants of this class) while 'Program' operations are handled by descendants of this class, vtkCgShaderProgram, vtkGLSLShaderProgram.

To create an instance of class vtkShaderProgram, simply invoke its constructor as follows

```
obj = vtkShaderProgram
```

39.174.2 Methods

The class vtkShaderProgram has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkShaderProgram class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkShaderProgram = obj.NewInstance ()`
- `vtkShaderProgram = obj.SafeDownCast (vtkObject o)`
- `vtkXMLMaterial = obj.GetMaterial ()`
- `obj.SetMaterial (vtkXMLMaterial )`
- `int = obj.AddShader (vtkShader shader)`
- `obj.RemoveShader (int index)` - Remove a shader at the given index.
- `obj.RemoveShader (vtkShader shader)` - Removes the given shader.
- `vtkCollectionIterator = obj.NewShaderIterator ()` - Returns a new iterator to iterate over the shaders.
- `int = obj.GetNumberOfShaders ()` - Returns the number of shaders available in this shader program.
- `obj.ReadMaterial ()`
- `obj.Render (vtkActor , vtkRenderer )`
- `obj.AddShaderVariable (string name, int numVars, int x)`
- `obj.AddShaderVariable (string name, int numVars, float x)`
- `obj.AddShaderVariable (string name, int numVars, double x)`
- `obj.PostRender (vtkActor , vtkRenderer )` - Called to unload the shaders after the actor has been rendered.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `vtkShaderDeviceAdapter = obj.GetShaderDeviceAdapter ()` - Get the vtkShaderDeviceAdapter which can be used to execute this shader program.

39.175 vtkShadowMapPass

39.175.1 Usage

Render the opaque polygonal geometry of a scene with shadow maps (a technique to render hard shadows in hardware).

This pass expects an initialized depth buffer and color buffer. Initialized buffers means they have been cleared with farest z-value and background color/gradient/transparent color. An opaque pass may have been performed right after the initialization.

Its delegate is usually set to a vtkOpaquePass.

.SECTION Implementation The first pass of the algorithm is to generate a shadow map per light (depth map from the light point of view) by rendering the opaque objects with the OCCLUDER property keys. The second pass is to render the opaque objects with the RECEIVER keys.

To create an instance of class vtkShadowMapPass, simply invoke its constructor as follows

```
obj = vtkShadowMapPass
```

39.175.2 Methods

The class vtkShadowMapPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkShadowMapPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkShadowMapPass = obj.NewInstance ()`
- `vtkShadowMapPass = obj.SafeDownCast (vtkObject o)`
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Release graphics resources and ask components to release their own resources.
- `vtkRenderPass = obj.GetOpaquePass ()` - Delegate for rendering the opaque polygonal geometry. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkTranslucentPass. Initial value is a NULL pointer.
- `obj.SetOpaquePass (vtkRenderPass opaquePass)` - Delegate for rendering the opaque polygonal geometry. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkTranslucentPass. Initial value is a NULL pointer.
- `vtkRenderPass = obj.GetCompositeZPass ()` - Delegate for rendering the opaque polygonal geometry. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkTranslucentPass. Initial value is a NULL pointer.
- `obj.SetCompositeZPass (vtkRenderPass opaquePass)` - Delegate for rendering the opaque polygonal geometry. If it is NULL, nothing will be rendered and a warning will be emitted. It is usually set to a vtkTranslucentPass. Initial value is a NULL pointer.
- `obj.SetResolution (int )` - Set/Get the number of pixels in each dimension of the shadow maps (shadow maps are square). Initial value is 256. The greater the better. Resolution does not have to be a power-of-two value.
- `int = obj.GetResolution ()` - Set/Get the number of pixels in each dimension of the shadow maps (shadow maps are square). Initial value is 256. The greater the better. Resolution does not have to be a power-of-two value.
- `obj.SetPolygonOffsetFactor (float )` - Factor used to scale the maximum depth slope of a polygon (definition from OpenGL 2.1 spec section 3.5.5 "Depth Offset" page 112). This is used during the creation the shadow maps (not during mapping of the shadow maps onto the geometry) Play with this value and PolygonOffsetUnits to solve self-shadowing. Valid values can be either positive or negative. Initial value is 1.1f (recommended by the nVidia presentation about Shadow Mapping by Cass Everitt). 3.1f works well with the regression test.

- `float = obj.GetPolygonOffsetFactor ()` - Factor used to scale the maximum depth slope of a polygon (definition from OpenGL 2.1 spec section 3.5.5 "Depth Offset" page 112). This is used during the creation the shadow maps (not during mapping of the shadow maps onto the geometry) Play with this value and PolygonOffsetUnits to solve self-shadowing. Valid values can be either positive or negative. Initial value is 1.1f (recommended by the nVidia presentation about Shadow Mapping by Cass Everitt). 3.1f works well with the regression test.

- `obj.SetPolygonOffsetUnits (float )` - Factor used to scale an implementation dependent constant that relates to the usable resolution of the depth buffer (definition from OpenGL 2.1 spec section 3.5.5 "Depth Offset" page 112). This is used during the creation the shadow maps (not during mapping of the shadow maps onto the geometry) Play with this value and PolygonOffsetFactor to solve self-shadowing. Valid values can be either positive or negative. Initial value is 4.0f (recommended by the nVidia presentation about Shadow Mapping by Cass Everitt). 10.0f works well with the regression test.

- `float = obj.GetPolygonOffsetUnits ()` - Factor used to scale an implementation dependent constant that relates to the usable resolution of the depth buffer (definition from OpenGL 2.1 spec section 3.5.5 "Depth Offset" page 112). This is used during the creation the shadow maps (not during mapping of the shadow maps onto the geometry) Play with this value and PolygonOffsetFactor to solve self-shadowing. Valid values can be either positive or negative. Initial value is 4.0f (recommended by the nVidia presentation about Shadow Mapping by Cass Everitt). 10.0f works well with the regression test.

39.176 vtkSobelGradientMagnitudePass

39.176.1 Usage

Detect the edges of the image renderered by its delegate. Edge-detection uses a Sobel high-pass filter (3x3 kernel).

This pass expects an initialized depth buffer and color buffer. Initialized buffers means they have been cleared with farest z-value and background color/gradient/transparent color. An opaque pass may have been performed right after the initialization.

The delegate is used once.

Its delegate is usually set to a vtkCameraPass or to a post-processing pass.

This pass requires a OpenGL context that supports texture objects (TO), framebuffer objects (FBO) and GLSL. If not, it will emit an error message and will render its delegate and return.

.SECTION Implementation To compute the gradient magnitude, the x and y components of the gradient (Gx and Gy) have to be computed first. Each computation of Gx and Gy uses a separable filter. The first pass takes the image from the delegate as the single input texture. The first pass has two outputs, one for the first part of Gx, Gx1, result of a convolution with (-1 0 1), one for the first part of Gy, Gy1, result of a convolution with (1 2 1). The second pass has two inputs, Gx1 and Gy1. Kernel (1 2 1)$\hat{\text{T}}$ is applied to Gx1 and kernel (-1 0 1)$\hat{\text{T}}$ is applied to Gx2. It gives the values for Gx and Gy. Thoses values are then used to compute the magnitude of the gradient which is stored in the render target. The gradient computation happens per component (R,G,B). A is arbitrarly set to 1 (full opacity).

To create an instance of class vtkSobelGradientMagnitudePass, simply invoke its constructor as follows

```
obj = vtkSobelGradientMagnitudePass
```

39.176.2 Methods

The class vtkSobelGradientMagnitudePass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSobelGradientMagnitudePass class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkSobelGradientMagnitudePass = obj.NewInstance ()`
- `vtkSobelGradientMagnitudePass = obj.SafeDownCast (vtkObject o)`
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Release graphics resources and ask components to release their own resources.

39.177 vtkStandardPolyDataPainter

39.177.1 Usage

vtkStandardPolyDataPainter is a catch-all painter. It should work with pretty much any vtkPolyData, and attributes, and vtkPolyDataPainterDeviceAdapter. On the flip side, the vtkStandardPolyDataPainter will be slower than the more special purpose painters.

To create an instance of class vtkStandardPolyDataPainter, simply invoke its constructor as follows

```
obj = vtkStandardPolyDataPainter
```

39.177.2 Methods

The class vtkStandardPolyDataPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkStandardPolyDataPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkStandardPolyDataPainter = obj.NewInstance ()`
- `vtkStandardPolyDataPainter = obj.SafeDownCast (vtkObject o)`
- `obj.AddMultiTextureCoordsArray (vtkDataArray array)`

39.178 vtkSurfaceLICDefaultPainter

39.178.1 Usage

vtkSurfaceLICDefaultPainter is a vtkDefaultPainter replacement that inserts the vtkSurfaceLICPainter at the correct position in the painter chain.

To create an instance of class vtkSurfaceLICDefaultPainter, simply invoke its constructor as follows

```
obj = vtkSurfaceLICDefaultPainter
```

39.178.2 Methods

The class vtkSurfaceLICDefaultPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSurfaceLICDefaultPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkSurfaceLICDefaultPainter = obj.NewInstance ()`
- `vtkSurfaceLICDefaultPainter = obj.SafeDownCast (vtkObject o)`
- `obj.SetSurfaceLICPainter (vtkSurfaceLICPainter )` - Get/Set the Surface LIC painter.
- `vtkSurfaceLICPainter = obj.GetSurfaceLICPainter ()` - Get/Set the Surface LIC painter.

39.179 vtkSurfaceLICPainter

39.179.1 Usage

vtkSurfaceLICPainter painter performs LIC on the surface of arbitrary geometry. Point vectors are used as the vector field for generating the LIC. The implementation is based on "Image Space Based Visualization on Unstread Flow on Surfaces" by Laramee, Jobard and Hauser appered in proceedings of IEEE Visualization '03, pages 131-138.

To create an instance of class vtkSurfaceLICPainter, simply invoke its constructor as follows

```
obj = vtkSurfaceLICPainter
```

39.179.2 Methods

The class vtkSurfaceLICPainter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSurfaceLICPainter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSurfaceLICPainter = obj.NewInstance ()`
- `vtkSurfaceLICPainter = obj.SafeDownCast (vtkObject o)`
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this mapper. The parameter window could be used to determine which graphic resources to release. In this case, releases the display lists.
- `vtkDataObject = obj.GetOutput ()` - Get the output data object from this painter. Overridden to pass the input points (or cells) vectors as the tcoords to the deletage painters. This is required by the internal GLSL shader programs used for generating LIC.
- `obj.SetEnable (int )` - Enable/Disable this painter.
- `int = obj.GetEnable ()` - Enable/Disable this painter.
- `obj.EnableOn ()` - Enable/Disable this painter.
- `obj.EnableOff ()` - Enable/Disable this painter.
- `obj.SetInputArrayToProcess (int fieldAssociation, string name)` - Set the vectors to used for applying LIC. By default point vectors are used. Arguments are same as those passed to vtkAlgorithm::SetInputArrayToProcess except the first 3 arguments i.e. idx, port, connection.
- `obj.SetInputArrayToProcess (int fieldAssociation, int fieldAttributeType)` - Set the vectors to used for applying LIC. By default point vectors are used. Arguments are same as those passed to vtkAlgorithm::SetInputArrayToProcess except the first 3 arguments i.e. idx, port, connection.

- `obj.SetEnhancedLIC (int )` - Enable/Disable enhanced LIC that improves image quality by increasing inter-streamline contrast while suppressing artifacts. Enhanced LIC performs two passes of LIC, with a 3x3 Laplacian high-pass filter in between that processes the output of pass #1 LIC and forwards the result as the input 'noise' to pass #2 LIC. This flag is automatically turned off during user interaction.
- `int = obj.GetEnhancedLIC ()` - Enable/Disable enhanced LIC that improves image quality by increasing inter-streamline contrast while suppressing artifacts. Enhanced LIC performs two passes of LIC, with a 3x3 Laplacian high-pass filter in between that processes the output of pass #1 LIC and forwards the result as the input 'noise' to pass #2 LIC. This flag is automatically turned off during user interaction.
- `obj.EnhancedLICOn ()` - Enable/Disable enhanced LIC that improves image quality by increasing inter-streamline contrast while suppressing artifacts. Enhanced LIC performs two passes of LIC, with a 3x3 Laplacian high-pass filter in between that processes the output of pass #1 LIC and forwards the result as the input 'noise' to pass #2 LIC. This flag is automatically turned off during user interaction.
- `obj.EnhancedLICOff ()` - Enable/Disable enhanced LIC that improves image quality by increasing inter-streamline contrast while suppressing artifacts. Enhanced LIC performs two passes of LIC, with a 3x3 Laplacian high-pass filter in between that processes the output of pass #1 LIC and forwards the result as the input 'noise' to pass #2 LIC. This flag is automatically turned off during user interaction.
- `obj.SetNumberOfSteps (int )` - Get/Set the number of integration steps in each direction.
- `int = obj.GetNumberOfSteps ()` - Get/Set the number of integration steps in each direction.
- `obj.SetStepSize (double )` - Get/Set the step size (in pixels).
- `double = obj.GetStepSize ()` - Get/Set the step size (in pixels).
- `obj.SetLICIntensity (double )` - Control the contribution of the LIC in the final output image. 0.0 produces same result as disabling LIC alltogether, while 1.0 implies show LIC result alone.
- `double = obj.GetLICIntensityMinValue ()` - Control the contribution of the LIC in the final output image. 0.0 produces same result as disabling LIC alltogether, while 1.0 implies show LIC result alone.
- `double = obj.GetLICIntensityMaxValue ()` - Control the contribution of the LIC in the final output image. 0.0 produces same result as disabling LIC alltogether, while 1.0 implies show LIC result alone.
- `double = obj.GetLICIntensity ()` - Control the contribution of the LIC in the final output image. 0.0 produces same result as disabling LIC alltogether, while 1.0 implies show LIC result alone.
- `int = obj.GetRenderingPreparationSuccess ()` - Check if the LIC process runs properly.
- `int = obj.GetLICSuccess ()` - Returns true is the rendering context supports extensions needed by this painter.

39.180 vtkTDxInteractorStyle

39.180.1 Usage

vtkTDxInteractorStyle is an abstract class defining an event-driven interface to support 3DConnexion device events send by vtkRenderWindowInteractor. vtkRenderWindowInteractor forwards events in a platform independent form to vtkInteractorStyle which can then delegate some processing to vtkTDxInteractorStyle.

To create an instance of class vtkTDxInteractorStyle, simply invoke its constructor as follows

```
obj = vtkTDxInteractorStyle
```

39.180.2 Methods

The class vtkTDxInteractorStyle has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTDxInteractorStyle class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTDxInteractorStyle = obj.NewInstance ()`
- `vtkTDxInteractorStyle = obj.SafeDownCast (vtkObject o)`
- `vtkTDxInteractorStyleSettings = obj.GetSettings ()` - 3Dconnexion device settings. (sensitivity, individual axis filters). Initial object is not null.
- `obj.SetSettings (vtkTDxInteractorStyleSettings settings)` - 3Dconnexion device settings. (sensitivity, individual axis filters). Initial object is not null.

39.181 vtkTDxInteractorStyleCamera

39.181.1 Usage

vtkTDxInteractorStyleCamera allows the end-user to manipulate tha camera with a 3DConnexion device.
To create an instance of class vtkTDxInteractorStyleCamera, simply invoke its constructor as follows

```
obj = vtkTDxInteractorStyleCamera
```

39.181.2 Methods

The class vtkTDxInteractorStyleCamera has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTDxInteractorStyleCamera class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTDxInteractorStyleCamera = obj.NewInstance ()`
- `vtkTDxInteractorStyleCamera = obj.SafeDownCast (vtkObject o)`

39.182 vtkTDxInteractorStyleSettings

39.182.1 Usage

vtkTDxInteractorStyleSettings defines settings for 3DConnexion device such as sensitivity, axis filters
To create an instance of class vtkTDxInteractorStyleSettings, simply invoke its constructor as follows

```
obj = vtkTDxInteractorStyleSettings
```

39.182.2 Methods

The class vtkTDxInteractorStyleSettings has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTDxInteractorStyleSettings class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTDxInteractorStyleSettings = obj.NewInstance ()`
- `vtkTDxInteractorStyleSettings = obj.SafeDownCast (vtkObject o)`
- `obj.SetAngleSensitivity (double )` - Sensitivity of the rotation angle. This can be any value: positive, negative, null. - x¡-1.0: faster reversed - x=-1.0: reversed neutral - -1.0¡x¡0.0: reversed slower - x=0.0: no rotation - 0.0¡x¡1.0: slower - x=1.0: neutral - x¿1.0: faster
- `double = obj.GetAngleSensitivity ()` - Sensitivity of the rotation angle. This can be any value: positive, negative, null. - x¡-1.0: faster reversed - x=-1.0: reversed neutral - -1.0¡x¡0.0: reversed slower - x=0.0: no rotation - 0.0¡x¡1.0: slower - x=1.0: neutral - x¿1.0: faster
- `obj.SetUseRotationX (bool )` - Use or mask the rotation component around the X-axis. Initial value is true.
- `bool = obj.GetUseRotationX ()` - Use or mask the rotation component around the X-axis. Initial value is true.
- `obj.SetUseRotationY (bool )` - Use or mask the rotation component around the Y-axis. Initial value is true.
- `bool = obj.GetUseRotationY ()` - Use or mask the rotation component around the Y-axis. Initial value is true.
- `obj.SetUseRotationZ (bool )` - Use or mask the rotation component around the Z-axis. Initial value is true.
- `bool = obj.GetUseRotationZ ()` - Use or mask the rotation component around the Z-axis. Initial value is true.
- `obj.SetTranslationXSensitivity (double )` - Sensitivity of the translation along the X-axis. This can be any value: positive, negative, null. - x¡-1.0: faster reversed - x=-1.0: reversed neutral - -1.0¡x¡0.0: reversed slower - x=0.0: no translation - 0.0¡x¡1.0: slower - x=1.0: neutral - x¿1.0: faster Initial value is 1.0
- `double = obj.GetTranslationXSensitivity ()` - Sensitivity of the translation along the X-axis. This can be any value: positive, negative, null. - x¡-1.0: faster reversed - x=-1.0: reversed neutral - -1.0¡x¡0.0: reversed slower - x=0.0: no translation - 0.0¡x¡1.0: slower - x=1.0: neutral - x¿1.0: faster Initial value is 1.0
- `obj.SetTranslationYSensitivity (double )` - Sensitivity of the translation along the Y-axis. See comment of SetTranslationXSensitivity().
- `double = obj.GetTranslationYSensitivity ()` - Sensitivity of the translation along the Y-axis. See comment of SetTranslationXSensitivity().
- `obj.SetTranslationZSensitivity (double )` - Sensitivity of the translation along the Z-axis. See comment of SetTranslationXSensitivity().
- `double = obj.GetTranslationZSensitivity ()` - Sensitivity of the translation along the Z-axis. See comment of SetTranslationXSensitivity().

39.183 vtkTesting

39.183.1 Usage

This is a VTK regression testing framework. Looks like this:
vtkTesting* t = vtkTesting::New();
Two options for setting arguments
Option 1: for (cc = 1; cc ¡ argc; cc ++) t-¿AddArgument(argv[cc]);
Option 2: t-¿AddArgument("-D"); t-¿AddArgument(my_data_dir); t-¿AddArgument("-V"); t-¿AddArgument(my_valid_in
...
Two options of doing testing:
Option 1: t-¿SetRenderWindow(renWin); int res = t-¿RegressionTest(threshold);
Option 2: int res = t-¿RegressionTest(test_image, threshold);
...
if (res == vtkTesting::PASSED) Test passed else Test failed
To create an instance of class vtkTesting, simply invoke its constructor as follows

```
obj = vtkTesting
```

39.183.2 Methods

The class vtkTesting has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTesting class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTesting = obj.NewInstance ()`
- `vtkTesting = obj.SafeDownCast (vtkObject o)`
- `obj.SetFrontBuffer (int )` - Use front buffer for tests. By default use back buffer.
- `int = obj.GetFrontBufferMinValue ()` - Use front buffer for tests. By default use back buffer.
- `int = obj.GetFrontBufferMaxValue ()` - Use front buffer for tests. By default use back buffer.
- `obj.FrontBufferOn ()` - Use front buffer for tests. By default use back buffer.
- `obj.FrontBufferOff ()` - Use front buffer for tests. By default use back buffer.
- `int = obj.GetFrontBuffer ()` - Use front buffer for tests. By default use back buffer.
- `int = obj.RegressionTest (double thresh)` - Perform the test and return result. At the same time the output will be written cout
- `int = obj.RegressionTest (vtkImageData image, double thresh)` - Compare the image with the valid image.
- `int = obj.CompareAverageOfL2Norm (vtkDataSet pdA, vtkDataSet pdB, double tol)` - Compute the average L2 norm between all point data data arrays of types float and double present in the data sets "dsA" and "dsB" (this includes instances of vtkPoints) Compare the result of each L2 comutation to "tol".
- `int = obj.CompareAverageOfL2Norm (vtkDataArray daA, vtkDataArray daB, double tol)` - Compute the average L2 norm between two data arrays "daA" and "daB" and compare against "tol".
- `obj.SetRenderWindow (vtkRenderWindow rw)` - Set and get the render window that will be used for regression testing.

- `vtkRenderWindow = obj.GetRenderWindow ()` - Set and get the render window that will be used for regression testing.
- `obj.SetValidImageFileName (string )` - Set/Get the name of the valid image file
- `string = obj.GetValidImageFileName ()` - Set/Get the name of the valid image file
- `double = obj.GetImageDifference ()` - Get the image difference.
- `obj.AddArgument (string argv)` - Pass the command line arguments into this class to be processed. Many of the Get methods such as GetValidImage and GetBaselineRoot rely on the arguments to be passed in prior to retrieving these values. Just call AddArgument for each argument that was passed into the command line
- `obj.CleanArguments ()`
- `string = obj.GetDataRoot ()` - Get some paramters from the command line arguments, env, or defaults
- `obj.SetDataRoot (string )` - Get some paramters from the command line arguments, env, or defaults
- `string = obj.GetTempDirectory ()` - Get some paramters from the command line arguments, env, or defaults
- `obj.SetTempDirectory (string )` - Get some paramters from the command line arguments, env, or defaults
- `int = obj.IsValidImageSpecified ()` - Is a valid image specified on the command line areguments?
- `int = obj.IsInteractiveModeSpecified ()` - Is the interactive mode specified?
- `int = obj.IsFlagSpecified (string flag)` - Is some arbitrary user flag ("-X", "-Z" etc) specified
- `obj.SetBorderOffset (int )` - Number of pixels added as borders to avoid problems with window decorations added by some window managers.
- `int = obj.GetBorderOffset ()` - Number of pixels added as borders to avoid problems with window decorations added by some window managers.
- `obj.SetVerbose (int )` - Get/Set verbosity level. A level of 0 is quiet.
- `int = obj.GetVerbose ()` - Get/Set verbosity level. A level of 0 is quiet.

39.184 vtkTextActor

39.184.1 Usage

vtkTextActor can be used to place text annotation into a window. When TextScaleMode is NONE, the text is fixed font and operation is the same as a vtkPolyDataMapper2D/vtkActor2D pair. When TextScaleMode is VIEWPORT, the font resizes such that it maintains a consistent size relative to the viewport in which it is rendered. When TextScaleMode is PROP, the font resizes such that the text fits inside the box defined by the position 1 & 2 coordinates. This class replaces the deprecated vtkScaledTextActor and acts as a convenient wrapper for a vtkTextMapper/vtkActor2D pair. Set the text property/attributes through the vtkTextProperty associated to this actor.

To create an instance of class vtkTextActor, simply invoke its constructor as follows

```
obj = vtkTextActor
```

39.184.2 Methods

The class vtkTextActor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextActor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTextActor = obj.NewInstance ()`
- `vtkTextActor = obj.SafeDownCast (vtkObject o)`
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this text actor. Overloads the virtual vtkProp method.
- `obj.SetMapper (vtkPolyDataMapper2D mapper)` - Override the vtkPolyDataMapper2D that defines the text to be drawn. One will be created by default if none is supplied
- `obj.SetInput (string inputString)` - Set the text string to be displayed. "
n" is recognized as a carriage return/linefeed (line separator). Only 7-bit ASCII is allowed (anything else, such as Latin1 or UTF8, is not acceptable). Convenience method to the underlying mapper
- `string = obj.GetInput ()` - Set the text string to be displayed. "
n" is recognized as a carriage return/linefeed (line separator). Only 7-bit ASCII is allowed (anything else, such as Latin1 or UTF8, is not acceptable). Convenience method to the underlying mapper
- `obj.SetMinimumSize (int , int )` - Set/Get the minimum size in pixels for this actor. Defaults to 10,10. Only valid when TextScaleMode is PROP.
- `obj.SetMinimumSize (int  a[2])` - Set/Get the minimum size in pixels for this actor. Defaults to 10,10. Only valid when TextScaleMode is PROP.
- `int = obj. GetMinimumSize ()` - Set/Get the minimum size in pixels for this actor. Defaults to 10,10. Only valid when TextScaleMode is PROP.
- `obj.SetMaximumLineHeight (float )` - Set/Get the maximum height of a line of text as a percentage of the vertical area allocated to this scaled text actor. Defaults to 1.0. Only valid when TextScaleMode is PROP.
- `float = obj.GetMaximumLineHeight ()` - Set/Get the maximum height of a line of text as a percentage of the vertical area allocated to this scaled text actor. Defaults to 1.0. Only valid when TextScaleMode is PROP.
- `obj.SetTextScaleMode (int )` - Set how text should be scaled. If set to vtkTextActor::TEXT_SCALE_MODE_NONE, the the font size will be fixed by the size given in TextProperty. If set to vtkTextActor::TEXT_SCALE_MODE_PROP, the text will be scaled to fit exactly in the prop as specified by the position 1 & 2 coordinates. If set to vtkTextActor::TEXT_SCALE_MODE_VIEWPORT, the text will be scaled based on the size of the viewport it is displayed in.
- `int = obj.GetTextScaleModeMinValue ()` - Set how text should be scaled. If set to vtkTextActor::TEXT_SCALE_MODE_NONE, the the font size will be fixed by the size given in TextProperty. If set to vtkTextActor::TEXT_SCALE_MODE_PROP, the text will be scaled to fit exactly in the prop as specified by the position 1 & 2 coordinates. If set to vtkTextActor::TEXT_SCALE_MODE_VIEWPORT, the text will be scaled based on the size of the viewport it is displayed in.

- `int = obj.GetTextScaleModeMaxValue ()` - Set how text should be scaled. If set to vtkTextActor::TEXT_SCALE_MODE_NONE, the the font size will be fixed by the size given in TextProperty. If set to vtkTextActor::TEXT_SCALE_MODE_PROP, the text will be scaled to fit exactly in the prop as specified by the position 1 & 2 coordinates. If set to vtkTextActor::TEXT_SCALE_MODE_VIEWPORT, the text will be scaled based on the size of the viewport it is displayed in.
- `int = obj.GetTextScaleMode ()` - Set how text should be scaled. If set to vtkTextActor::TEXT_SCALE_MODE_NON the the font size will be fixed by the size given in TextProperty. If set to vtkTextActor::TEXT_SCALE_MODE_PROP, the text will be scaled to fit exactly in the prop as specified by the position 1 & 2 coordinates. If set to vtkTextActor::TEXT_SCALE_MODE_VIEWPORT, the text will be scaled based on the size of the viewport it is displayed in.
- `obj.SetTextScaleModeToNone ()` - Set how text should be scaled. If set to vtkTextActor::TEXT_SCALE_MODE_NON the the font size will be fixed by the size given in TextProperty. If set to vtkTextActor::TEXT_SCALE_MODE_PROP, the text will be scaled to fit exactly in the prop as specified by the position 1 & 2 coordinates. If set to vtkTextActor::TEXT_SCALE_MODE_VIEWPORT, the text will be scaled based on the size of the viewport it is displayed in.
- `obj.SetTextScaleModeToProp ()` - Set how text should be scaled. If set to vtkTextActor::TEXT_SCALE_MODE_NON the the font size will be fixed by the size given in TextProperty. If set to vtkTextActor::TEXT_SCALE_MODE_PROP, the text will be scaled to fit exactly in the prop as specified by the position 1 & 2 coordinates. If set to vtkTextActor::TEXT_SCALE_MODE_VIEWPORT, the text will be scaled based on the size of the viewport it is displayed in.
- `obj.SetTextScaleModeToViewport ()` - DO NOT CALL. Deprecated in VTK 5.4. Use SetTextScaleMode or GetTextScaleMode instead.
- `obj.SetScaledText (int )` - DO NOT CALL. Deprecated in VTK 5.4. Use SetTextScaleMode or GetTextScaleMode instead.
- `int = obj.GetScaledText ()` - DO NOT CALL. Deprecated in VTK 5.4. Use SetTextScaleMode or GetTextScaleMode instead.
- `obj.ScaledTextOn ()` - DO NOT CALL. Deprecated in VTK 5.4. Use SetTextScaleMode or GetTextScaleMode instead.
- `obj.ScaledTextOff ()` - DO NOT CALL. Deprecated in VTK 5.4. Use SetTextScaleMode or GetTextScaleMode instead.
- `obj.SetUseBorderAlign (int )` - Turn on or off the UseBorderAlign option. When UseBorderAlign is on, the bounding rectangle is used to align the text, which is the proper behavior when using vtkTextRepresentation
- `int = obj.GetUseBorderAlign ()` - Turn on or off the UseBorderAlign option. When UseBorderAlign is on, the bounding rectangle is used to align the text, which is the proper behavior when using vtkTextRepresentation
- `obj.UseBorderAlignOn ()` - Turn on or off the UseBorderAlign option. When UseBorderAlign is on, the bounding rectangle is used to align the text, which is the proper behavior when using vtkTextRepresentation
- `obj.UseBorderAlignOff ()` - Turn on or off the UseBorderAlign option. When UseBorderAlign is on, the bounding rectangle is used to align the text, which is the proper behavior when using vtkTextRepresentation
- `obj.SetAlignmentPoint (int point)` - This method is being depricated. Use SetJustification and SetVerticalJustification in text property instead. Set/Get the Alignment point if zero (default), the text aligns itself to the bottom left corner (which is defined by the PositionCoordinate) otherwise the text aligns itself to corner/midpoint or centre @verbatim 6 7 8 3 4 5 0 1 2 @endverbatim This is

the same as setting the TextProperty's justification. Currently TextActor is not oriented around its AlignmentPoint.

- `int = obj.GetAlignmentPoint ()` - This method is being depricated. Use SetJustification and SetVerticalJustification in text property instead. Set/Get the Alignment point if zero (default), the text aligns itself to the bottom left corner (which is defined by the PositionCoordinate) otherwise the text aligns itself to corner/midpoint or centre @verbatim 6 7 8 3 4 5 0 1 2 @endverbatim This is the same as setting the TextProperty's justification. Currently TextActor is not oriented around its AlignmentPoint.

- `obj.SetOrientation (float orientation)` - Counterclockwise rotation around the Alignment point. Units are in degrees and defaults to 0. The orientation in the text property rotates the text in the texture map. It will proba ly not give you the effect you desire.

- `float = obj.GetOrientation ()` - Counterclockwise rotation around the Alignment point. Units are in degrees and defaults to 0. The orientation in the text property rotates the text in the texture map. It will proba ly not give you the effect you desire.

- `obj.SetTextProperty (vtkTextProperty p)` - Set/Get the text property.

- `vtkTextProperty = obj.GetTextProperty ()` - Set/Get the text property.

- `obj.SetNonLinearFontScale (double exponent, int target)` - Enable non-linear scaling of font sizes. This is useful in combination with scaled text. With small windows you want to use the entire scaled text area. With larger windows you want to reduce the font size some so that the entire area is not used. These values modify the computed font size as follows: newFontSize = pow(FontSize,exponent)*pow(target,1.0 - exponent) typically exponent should be around 0.7 and target should be around 10

- `obj.SpecifiedToDisplay (double pos, vtkViewport vport, int specified)` - This is just a simple coordinate conversion method used in the render process.

- `obj.DisplayToSpecified (double pos, vtkViewport vport, int specified)` - This is just a simple coordinate conversion method used in the render process.

- `obj.ComputeScaledFont (vtkViewport viewport)` - Compute the scale the font should be given the viewport. The result is placed in the ScaledTextProperty ivar.

- `vtkTextProperty = obj.GetScaledTextProperty ()` - Get the scaled font. Use ComputeScaledFont to set the scale for a given viewport.

39.185 vtkTextActor3D

39.185.1 Usage

The input text is rendered into a buffer, which in turn is used as a texture applied onto a quad (a vtkImageActor is used under the hood). .SECTION Warning This class is experimental at the moment. - The orientation is not optimized, the quad should be oriented, not the text itself when it is rendered in the buffer (we end up with excessively big textures for 45 degrees angles). This will be fixed first. - No checking is done at the moment regarding hardware texture size limits. - Alignment is not supported (soon). - Multiline is not supported. - Need to fix angle out of 0¡-¿360

To create an instance of class vtkTextActor3D, simply invoke its constructor as follows

```
obj = vtkTextActor3D
```

39.185.2 Methods

The class vtkTextActor3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextActor3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTextActor3D = obj.NewInstance ()`
- `vtkTextActor3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (string )` - Set the text string to be displayed.
- `string = obj.GetInput ()` - Set the text string to be displayed.
- `obj.SetTextProperty (vtkTextProperty p)` - Set/Get the text property.
- `vtkTextProperty = obj.GetTextProperty ()` - Set/Get the text property.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this text actor. Overloads the virtual vtkProp method.
- `double = obj.GetBounds ()` - Get the bounds for this Prop3D as (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax). These are the padded-to-power-of-two texture bounds.
- `int = obj.GetBoundingBox (int bbox[4])` - Get the Freetype-derived real bounding box for the given vtkTextProperty and text string str. Results are returned in the four element bbox int array. This call can be used for sizing other elements.

39.186 vtkTextMapper

39.186.1 Usage

vtkTextMapper provides 2D text annotation support for VTK. It is a vtkMapper2D that can be associated with a vtkActor2D and placed into a vtkRenderer.

To use vtkTextMapper, specify an input text string.

To create an instance of class vtkTextMapper, simply invoke its constructor as follows

```
obj = vtkTextMapper
```

39.186.2 Methods

The class vtkTextMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTextMapper = obj.NewInstance ()`
- `vtkTextMapper = obj.SafeDownCast (vtkObject o)`
- `obj.GetSize (vtkViewport , int size[2])` - Return the size[2]/width/height of the rectangle required to draw this mapper (in pixels).

- `int = obj.GetWidth (vtkViewport v)` - Return the size[2]/width/height of the rectangle required to draw this mapper (in pixels).
- `int = obj.GetHeight (vtkViewport v)` - Return the size[2]/width/height of the rectangle required to draw this mapper (in pixels).
- `obj.SetInput (string inputString)` - Set the input text string to the mapper. The mapper recognizes ”
n” as a carriage return/linefeed (line separator).
- `string = obj.GetInput ()` - Set the input text string to the mapper. The mapper recognizes ”
n” as a carriage return/linefeed (line separator).
- `obj.SetTextProperty (vtkTextProperty p)` - Set/Get the text property.
- `vtkTextProperty = obj.GetTextProperty ()` - Set/Get the text property.
- `obj.ShallowCopy (vtkTextMapper tm)` - Shallow copy of an actor.
- `int = obj.GetNumberOfLines (string input)` - Determine the number of lines in the input string (delimited by ”
n”).
- `int = obj.GetNumberOfLines ()` - Get the number of lines in the input string (the method GetNumberOfLines(char*) must have been previously called for the return value to be valid).
- `int = obj.SetConstrainedFontSize (vtkViewport , int targetWidth, int targetHeight)` - Set and return the font size required to make this mapper fit in a given target rectangle (width x height, in pixels). A static version of the method is also available for convenience to other classes (e.g., widgets).
- `int = obj.GetSystemFontSize (int size)`

39.187 vtkTextProperty

39.187.1 Usage

vtkTextProperty is an object that represents text properties. The primary properties that can be set are color, opacity, font size, font family horizontal and vertical justification, bold/italic/shadow styles.

To create an instance of class vtkTextProperty, simply invoke its constructor as follows

```
obj = vtkTextProperty
```

39.187.2 Methods

The class vtkTextProperty has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextProperty class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTextProperty = obj.NewInstance ()`
- `vtkTextProperty = obj.SafeDownCast (vtkObject o)`
- `obj.SetColor (double , double , double )` - Set the color of the text.
- `obj.SetColor (double  a[3])` - Set the color of the text.

- `double = obj. GetColor ()` - Set the color of the text.
- `obj.SetOpacity (double )` - Set/Get the text's opacity. 1.0 is totally opaque and 0.0 is completely transparent.
- `double = obj.GetOpacity ()` - Set/Get the text's opacity. 1.0 is totally opaque and 0.0 is completely transparent.
- `string = obj.GetFontFamilyAsString ()` - Set/Get the font family. Supports legacy three font family system.
- `obj.SetFontFamilyAsString (string )` - Set/Get the font family. Supports legacy three font family system.
- `obj.SetFontFamily (int t)` - Set/Get the font family. Supports legacy three font family system.
- `int = obj.GetFontFamily ()` - Set/Get the font family. Supports legacy three font family system.
- `int = obj.GetFontFamilyMinValue ()` - Set/Get the font family. Supports legacy three font family system.
- `obj.SetFontFamilyToArial ()` - Set/Get the font family. Supports legacy three font family system.
- `obj.SetFontFamilyToCourier ()` - Set/Get the font family. Supports legacy three font family system.
- `obj.SetFontFamilyToTimes ()` - Set/Get the font family. Supports legacy three font family system.
- `obj.SetFontSize (int )` - Set/Get the font size (in points).
- `int = obj.GetFontSizeMinValue ()` - Set/Get the font size (in points).
- `int = obj.GetFontSizeMaxValue ()` - Set/Get the font size (in points).
- `int = obj.GetFontSize ()` - Set/Get the font size (in points).
- `obj.SetBold (int )` - Enable/disable text bolding.
- `int = obj.GetBold ()` - Enable/disable text bolding.
- `obj.BoldOn ()` - Enable/disable text bolding.
- `obj.BoldOff ()` - Enable/disable text bolding.
- `obj.SetItalic (int )` - Enable/disable text italic.
- `int = obj.GetItalic ()` - Enable/disable text italic.
- `obj.ItalicOn ()` - Enable/disable text italic.
- `obj.ItalicOff ()` - Enable/disable text italic.
- `obj.SetShadow (int )` - Enable/disable text shadow.
- `int = obj.GetShadow ()` - Enable/disable text shadow.
- `obj.ShadowOn ()` - Enable/disable text shadow.
- `obj.ShadowOff ()` - Enable/disable text shadow.
- `obj.SetShadowOffset (int , int )` - Set/Get the shadow offset, i.e. the distance from the text to its shadow, in the same unit as FontSize.
- `obj.SetShadowOffset (int  a[2])` - Set/Get the shadow offset, i.e. the distance from the text to its shadow, in the same unit as FontSize.

- `int = obj. GetShadowOffset ()` - Set/Get the shadow offset, i.e. the distance from the text to its shadow, in the same unit as FontSize.
- `obj.GetShadowColor (double color[3])` - Get the shadow color. It is computed from the Color ivar
- `obj.SetJustification (int )` - Set/Get the horizontal justification to left (default), centered, or right.
- `int = obj.GetJustificationMinValue ()` - Set/Get the horizontal justification to left (default), centered, or right.
- `int = obj.GetJustificationMaxValue ()` - Set/Get the horizontal justification to left (default), centered, or right.
- `int = obj.GetJustification ()` - Set/Get the horizontal justification to left (default), centered, or right.
- `obj.SetJustificationToLeft ()` - Set/Get the horizontal justification to left (default), centered, or right.
- `obj.SetJustificationToCentered ()` - Set/Get the horizontal justification to left (default), centered, or right.
- `obj.SetJustificationToRight ()` - Set/Get the horizontal justification to left (default), centered, or right.
- `string = obj.GetJustificationAsString ()` - Set/Get the horizontal justification to left (default), centered, or right.
- `obj.SetVerticalJustification (int )` - Set/Get the vertical justification to bottom (default), middle, or top.
- `int = obj.GetVerticalJustificationMinValue ()` - Set/Get the vertical justification to bottom (default), middle, or top.
- `int = obj.GetVerticalJustificationMaxValue ()` - Set/Get the vertical justification to bottom (default), middle, or top.
- `int = obj.GetVerticalJustification ()` - Set/Get the vertical justification to bottom (default), middle, or top.
- `obj.SetVerticalJustificationToBottom ()` - Set/Get the vertical justification to bottom (default), middle, or top.
- `obj.SetVerticalJustificationToCentered ()` - Set/Get the vertical justification to bottom (default), middle, or top.
- `obj.SetVerticalJustificationToTop ()` - Set/Get the vertical justification to bottom (default), middle, or top.
- `string = obj.GetVerticalJustificationAsString ()` - Set/Get the vertical justification to bottom (default), middle, or top.
- `obj.SetOrientation (double )` - Set/Get the text's orientation (in degrees).
- `double = obj.GetOrientation ()` - Set/Get the text's orientation (in degrees).
- `obj.SetLineSpacing (double )` - Set/Get the (extra) spacing between lines, expressed as a text height multiplication factor.
- `double = obj.GetLineSpacing ()` - Set/Get the (extra) spacing between lines, expressed as a text height multiplication factor.

- `obj.SetLineOffset (double )` - Set/Get the vertical offset (measured in pixels).
- `double = obj.GetLineOffset ()` - Set/Get the vertical offset (measured in pixels).
- `obj.ShallowCopy (vtkTextProperty tprop)` - Shallow copy of a text property.

39.188 vtkTexture

39.188.1 Usage

vtkTexture is an object that handles loading and binding of texture maps. It obtains its data from an input image data dataset type. Thus you can create visualization pipelines to read, process, and construct textures. Note that textures will only work if texture coordinates are also defined, and if the rendering system supports texture.

Instances of vtkTexture are associated with actors via the actor's SetTexture() method. Actors can share texture maps (this is encouraged to save memory resources.)

To create an instance of class vtkTexture, simply invoke its constructor as follows

```
obj = vtkTexture
```

39.188.2 Methods

The class vtkTexture has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTexture class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTexture = obj.NewInstance ()`
- `vtkTexture = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer ren)` - Renders a texture map. It first checks the object's modified time to make sure the texture maps Input is valid, then it invokes the Load() method.
- `obj.PostRender (vtkRenderer )` - Cleans up after the texture rendering to restore the state of the graphics context.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this texture. The parameter window could be used to determine which graphic resources to release.
- `obj.Load (vtkRenderer )` - Abstract interface to renderer. Each concrete subclass of vtkTexture will load its data into graphics system in response to this method invocation.
- `int = obj.GetRepeat ()` - Turn on/off the repetition of the texture map when the texture coords extend beyond the [0,1] range.
- `obj.SetRepeat (int )` - Turn on/off the repetition of the texture map when the texture coords extend beyond the [0,1] range.
- `obj.RepeatOn ()` - Turn on/off the repetition of the texture map when the texture coords extend beyond the [0,1] range.
- `obj.RepeatOff ()` - Turn on/off the repetition of the texture map when the texture coords extend beyond the [0,1] range.

- `int = obj.GetEdgeClamp ()` - Turn on/off the clamping of the texture map when the texture coords extend beyond the [0,1] range. Only used when Repeat is off, and edge clamping is supported by the graphics card.

- `obj.SetEdgeClamp (int )` - Turn on/off the clamping of the texture map when the texture coords extend beyond the [0,1] range. Only used when Repeat is off, and edge clamping is supported by the graphics card.

- `obj.EdgeClampOn ()` - Turn on/off the clamping of the texture map when the texture coords extend beyond the [0,1] range. Only used when Repeat is off, and edge clamping is supported by the graphics card.

- `obj.EdgeClampOff ()` - Turn on/off the clamping of the texture map when the texture coords extend beyond the [0,1] range. Only used when Repeat is off, and edge clamping is supported by the graphics card.

- `int = obj.GetInterpolate ()` - Turn on/off linear interpolation of the texture map when rendering.

- `obj.SetInterpolate (int )` - Turn on/off linear interpolation of the texture map when rendering.

- `obj.InterpolateOn ()` - Turn on/off linear interpolation of the texture map when rendering.

- `obj.InterpolateOff ()` - Turn on/off linear interpolation of the texture map when rendering.

- `obj.SetQuality (int )` - Force texture quality to 16-bit or 32-bit. This might not be supported on all machines.

- `int = obj.GetQuality ()` - Force texture quality to 16-bit or 32-bit. This might not be supported on all machines.

- `obj.SetQualityToDefault ()` - Force texture quality to 16-bit or 32-bit. This might not be supported on all machines.

- `obj.SetQualityTo16Bit ()` - Force texture quality to 16-bit or 32-bit. This might not be supported on all machines.

- `obj.SetQualityTo32Bit ()` - Force texture quality to 16-bit or 32-bit. This might not be supported on all machines.

- `int = obj.GetMapColorScalarsThroughLookupTable ()` - Turn on/off the mapping of color scalars through the lookup table. The default is Off. If Off, unsigned char scalars will be used directly as texture. If On, scalars will be mapped through the lookup table to generate 4-component unsigned char scalars. This ivar does not affect other scalars like unsigned short, float, etc. These scalars are always mapped through lookup tables.

- `obj.SetMapColorScalarsThroughLookupTable (int )` - Turn on/off the mapping of color scalars through the lookup table. The default is Off. If Off, unsigned char scalars will be used directly as texture. If On, scalars will be mapped through the lookup table to generate 4-component unsigned char scalars. This ivar does not affect other scalars like unsigned short, float, etc. These scalars are always mapped through lookup tables.

- `obj.MapColorScalarsThroughLookupTableOn ()` - Turn on/off the mapping of color scalars through the lookup table. The default is Off. If Off, unsigned char scalars will be used directly as texture. If On, scalars will be mapped through the lookup table to generate 4-component unsigned char scalars. This ivar does not affect other scalars like unsigned short, float, etc. These scalars are always mapped through lookup tables.

- `obj.MapColorScalarsThroughLookupTableOff ()` - Turn on/off the mapping of color scalars through the lookup table. The default is Off. If Off, unsigned char scalars will be used directly as texture. If On, scalars will be mapped through the lookup table to generate 4-component unsigned char scalars. This ivar does not affect other scalars like unsigned short, float, etc. These scalars are always mapped through lookup tables.

- `obj.SetLookupTable (vtkScalarsToColors )` - Specify the lookup table to convert scalars if necessary

- `vtkScalarsToColors = obj.GetLookupTable ()` - Specify the lookup table to convert scalars if necessary

- `vtkUnsignedCharArray = obj.GetMappedScalars ()` - Get Mapped Scalars

- `obj.SetTransform (vtkTransform transform)` - Set a transform on the texture which allows one to scale, rotate and translate the texture.

- `vtkTransform = obj.GetTransform ()` - Set a transform on the texture which allows one to scale, rotate and translate the texture.

- `int = obj.GetBlendingMode ()` - Used to specify how the texture will blend its RGB and Alpha values with other textures and the fragment the texture is rendered upon.

- `obj.SetBlendingMode (int )` - Used to specify how the texture will blend its RGB and Alpha values with other textures and the fragment the texture is rendered upon.

- `bool = obj.GetPremultipliedAlpha ()` - Whether the texture colors are premultiplied by alpha. Initial value is false.

- `obj.SetPremultipliedAlpha (bool )` - Whether the texture colors are premultiplied by alpha. Initial value is false.

- `obj.PremultipliedAlphaOn ()` - Whether the texture colors are premultiplied by alpha. Initial value is false.

- `obj.PremultipliedAlphaOff ()` - Whether the texture colors are premultiplied by alpha. Initial value is false.

- `int = obj.GetRestrictPowerOf2ImageSmaller ()` - When the texture is forced to be a power of 2, the default behavior is for the "new" image's dimensions to be greater than or equal to with respects to the original. Setting RestrictPowerOf2ImageSmaller to be 1 (or ON) with force the new image's dimensions to be less than or equal to with respects to the original.

- `obj.SetRestrictPowerOf2ImageSmaller (int )` - When the texture is forced to be a power of 2, the default behavior is for the "new" image's dimensions to be greater than or equal to with respects to the original. Setting RestrictPowerOf2ImageSmaller to be 1 (or ON) with force the new image's dimensions to be less than or equal to with respects to the original.

- `obj.RestrictPowerOf2ImageSmallerOn ()` - When the texture is forced to be a power of 2, the default behavior is for the "new" image's dimensions to be greater than or equal to with respects to the original. Setting RestrictPowerOf2ImageSmaller to be 1 (or ON) with force the new image's dimensions to be less than or equal to with respects to the original.

- `obj.RestrictPowerOf2ImageSmallerOff ()` - When the texture is forced to be a power of 2, the default behavior is for the "new" image's dimensions to be greater than or equal to with respects to the original. Setting RestrictPowerOf2ImageSmaller to be 1 (or ON) with force the new image's dimensions to be less than or equal to with respects to the original.

39.189 vtkTexturedActor2D

39.189.1 Usage

vtkTexturedActor2D is an Actor2D which has additional support for textures, just like vtkActor. To use textures, the geometry must have texture coordinates, and the texture must be set with SetTexture().

To create an instance of class vtkTexturedActor2D, simply invoke its constructor as follows

```
obj = vtkTexturedActor2D
```

39.189.2 Methods

The class vtkTexturedActor2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTexturedActor2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTexturedActor2D = obj.NewInstance ()`
- `vtkTexturedActor2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetTexture (vtkTexture texture)` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. An actor does not need to have an associated texture map and multiple actors can share one texture.
- `vtkTexture = obj.GetTexture ()` - Set/Get the texture object to control rendering texture maps. This will be a vtkTexture object. An actor does not need to have an associated texture map and multiple actors can share one texture.
- `obj.ReleaseGraphicsResources (vtkWindow win)` - Release any graphics resources that are being consumed by this actor. The parameter window could be used to determine which graphic resources to release.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Support the standard render methods.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Support the standard render methods.
- `long = obj.GetMTime ()` - Return this object's modified time.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this vtkTexturedActor2D. Overrides vtkActor2D method.

39.190 vtkTextureObject

39.190.1 Usage

vtkTextureObject represents an OpenGL texture object. It provides API to create textures using data already loaded into pixel buffer objects. It can also be used to create textures without uploading any data.

To create an instance of class vtkTextureObject, simply invoke its constructor as follows

```
obj = vtkTextureObject
```

39.190.2 Methods

The class vtkTextureObject has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextureObject class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTextureObject = obj.NewInstance ()`
- `vtkTextureObject = obj.SafeDownCast (vtkObject o)`
- `obj.SetContext (vtkRenderWindow )` - Get/Set the context. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions.
- `vtkRenderWindow = obj.GetContext ()` - Get/Set the context. This does not increase the reference count of the context to avoid reference loops. SetContext() may raise an error is the OpenGL context does not support the required OpenGL extensions.
- `int = obj.GetWidth ()` - Get the texture dimensions. These are the properties of the OpenGL texture this instance represents.
- `int = obj.GetHeight ()` - Get the texture dimensions. These are the properties of the OpenGL texture this instance represents.
- `int = obj.GetDepth ()` - Get the texture dimensions. These are the properties of the OpenGL texture this instance represents.
- `int = obj.GetComponents ()` - Get the texture dimensions. These are the properties of the OpenGL texture this instance represents.
- `int = obj.GetNumberOfDimensions ()`
- `int = obj.GetTarget ()` - Returns OpenGL texture target to which the texture is/can be bound.
- `int = obj.GetHandle ()` - Returns the OpenGL handle.
- `obj.Bind ()` - Activate the texture. The texture must have been created using Create(). RenderWindow must be set before calling this.
- `obj.UnBind ()` - Activate the texture. The texture must have been created using Create(). RenderWindow must be set before calling this.
- `bool = obj.IsBound ()` - Tells if the texture object is bound to the active texture image unit. (a texture object can be bound to multiple texture image unit).
- `obj.SendParameters ()` - Send all the texture object parameters to the hardware if not done yet.
- `bool = obj.Create1D (int numComps, vtkPixelBufferObject pbo, bool shaderSupportsTextureInt)` - Create a 1D texture using the PBO. Eventually we may start supporting creating a texture from subset of data in the PBO, but for simplicity we'll begin with entire PBO data. numComps must be in [1-4]. shaderSupportsTextureInt is true if the shader has an alternate implementation supporting sampler with integer values. Even if the card supports texture int, it does not mean that the implementor of the shader made a version that supports texture int.

- `bool = obj.Create2D (int width, int height, int numComps, vtkPixelBufferObject pbo, bool shaderSupp` - Create a 2D texture using the PBO. Eventually we may start supporting creating a texture from subset of data in the PBO, but for simplicity we'll begin with entire PBO data. numComps must be in [1-4].

- `bool = obj.CreateDepth (int width, int height, int internalFormat, vtkPixelBufferObject pbo)` - Create a 2D depth texture using a PBO. re: valid_internalFormat: internalFormat¿=0 && internalFormat¡NumberOfDepthFormats

- `bool = obj.AllocateDepth (int width, int height, int internalFormat)` - Create a 2D depth texture but does not initialize its values.

- `bool = obj.Allocate1D (int width, int numComps, int vtkType)` - Create a 1D color texture but does not initialize its values. Internal format is deduced from numComps and vtkType.

- `bool = obj.Allocate2D (int width, int height, int numComps, int vtkType)` - Create a 2D color texture but does not initialize its values. Internal format is deduced from numComps and vtkType.

- `bool = obj.Allocate3D (int width, int height, int depth, int numComps, int vtkType)` - Create a 3D color texture but does not initialize its values. Internal format is deduced from numComps and vtkType.

- `bool = obj.Create3D (int width, int height, int depth, int numComps, vtkPixelBufferObject pbo, bool` - Create a 3D texture using the PBO. Eventually we may start supporting creating a texture from subset of data in the PBO, but for simplicity we'll begin with entire PBO data. numComps must be in [1-4].

- `bool = obj.Create2D (int width, int height, int numComps, int vtktype, bool shaderSupportsTextureIn` - Create texture without uploading any data. To create a DEPTH_COMPONENT texture, vtktype must be set to VTK_VOID and numComps must be 1.

- `bool = obj.Create3D (int width, int height, int depth, int numComps, int vtktype, bool shaderSuppor` - Create texture without uploading any data. To create a DEPTH_COMPONENT texture, vtktype must be set to VTK_VOID and numComps must be 1.

- `vtkPixelBufferObject = obj.Download ()` - This is used to download raw data from the texture into a pixel bufer. The pixel buffer API can then be used to download the pixel buffer data to CPU arrays. The caller takes on the responsibility of deleting the returns vtkPixelBufferObject once it done with it.

- `int = obj.GetDataType ()` - Get the data type for the texture as a vtk type int i.e. VTK_INT etc.

- `int = obj.GetInternalFormat (int vtktype, int numComps, bool shaderSupportsTextureInt)`

- `int = obj.GetFormat (int vtktype, int numComps, bool shaderSupportsTextureInt)`

- `int = obj.GetWrapS ()` - Wrap mode for the first texture coordinate "s" Valid values are: - Clamp - ClampToEdge - Repeat - ClampToBorder - MirroredRepeat Initial value is Repeat (as in OpenGL spec)

- `obj.SetWrapS (int )` - Wrap mode for the first texture coordinate "s" Valid values are: - Clamp - ClampToEdge - Repeat - ClampToBorder - MirroredRepeat Initial value is Repeat (as in OpenGL spec)

- `int = obj.GetWrapT ()` - Wrap mode for the first texture coordinate "t" Valid values are: - Clamp - ClampToEdge - Repeat - ClampToBorder - MirroredRepeat Initial value is Repeat (as in OpenGL spec)

- `obj.SetWrapT (int )` - Wrap mode for the first texture coordinate "t" Valid values are: - Clamp - ClampToEdge - Repeat - ClampToBorder - MirroredRepeat Initial value is Repeat (as in OpenGL spec)
- `int = obj.GetWrapR ()` - Wrap mode for the first texture coordinate "r" Valid values are: - Clamp - ClampToEdge - Repeat - ClampToBorder - MirroredRepeat Initial value is Repeat (as in OpenGL spec)
- `obj.SetWrapR (int )` - Wrap mode for the first texture coordinate "r" Valid values are: - Clamp - ClampToEdge - Repeat - ClampToBorder - MirroredRepeat Initial value is Repeat (as in OpenGL spec)
- `int = obj.GetMinificationFilter ()` - Minification filter mode. Valid values are: - Nearest - Linear - NearestMipmapNearest - NearestMipmapLinear - LinearMipmapNearest - LinearMipmapLinear Initial value is Nearest (note initial value in OpenGL spec is NearestMipMapLinear but this is error-prone because it makes the texture object incomplete.).
- `obj.SetMinificationFilter (int )` - Minification filter mode. Valid values are: - Nearest - Linear - NearestMipmapNearest - NearestMipmapLinear - LinearMipmapNearest - LinearMipmapLinear Initial value is Nearest (note initial value in OpenGL spec is NearestMipMapLinear but this is error-prone because it makes the texture object incomplete.).
- `bool = obj.GetLinearMagnification ()` - Tells if the magnification mode is linear (true) or nearest (false). Initial value is false (initial value in OpenGL spec is true).
- `obj.SetLinearMagnification (bool )` - Tells if the magnification mode is linear (true) or nearest (false). Initial value is false (initial value in OpenGL spec is true).
- `obj.SetBorderColor (float , float , float , float )` - Border Color (RGBA). Each component is in [0.0f,1.0f]. Initial value is (0.0f,0.0f,0.0f,0.0f), as in OpenGL spec.
- `obj.SetBorderColor (float a[4])` - Border Color (RGBA). Each component is in [0.0f,1.0f]. Initial value is (0.0f,0.0f,0.0f,0.0f), as in OpenGL spec.
- `float = obj. GetBorderColor ()` - Border Color (RGBA). Each component is in [0.0f,1.0f]. Initial value is (0.0f,0.0f,0.0f,0.0f), as in OpenGL spec.
- `obj.SetPriority (float )` - Priority of the texture object to be resident on the card for higher performance in the range [0.0f,1.0f]. Initial value is 1.0f, as in OpenGL spec.
- `float = obj.GetPriority ()` - Priority of the texture object to be resident on the card for higher performance in the range [0.0f,1.0f]. Initial value is 1.0f, as in OpenGL spec.
- `obj.SetMinLOD (float )` - Lower-clamp the computed LOD against this value. Any float value is valid. Initial value is -1000.0f, as in OpenGL spec.
- `float = obj.GetMinLOD ()` - Lower-clamp the computed LOD against this value. Any float value is valid. Initial value is -1000.0f, as in OpenGL spec.
- `obj.SetMaxLOD (float )` - Upper-clamp the computed LOD against this value. Any float value is valid. Initial value is 1000.0f, as in OpenGL spec.
- `float = obj.GetMaxLOD ()` - Upper-clamp the computed LOD against this value. Any float value is valid. Initial value is 1000.0f, as in OpenGL spec.
- `obj.SetBaseLevel (int )` - Level of detail of the first texture image. A texture object is a list of texture images. It is a non-negative integer value. Initial value is 0, as in OpenGL spec.
- `int = obj.GetBaseLevel ()` - Level of detail of the first texture image. A texture object is a list of texture images. It is a non-negative integer value. Initial value is 0, as in OpenGL spec.

- `obj.SetMaxLevel (int )` - Level of detail of the first texture image. A texture object is a list of texture images. It is a non-negative integer value. Initial value is 1000, as in OpenGL spec.

- `int = obj.GetMaxLevel ()` - Level of detail of the first texture image. A texture object is a list of texture images. It is a non-negative integer value. Initial value is 1000, as in OpenGL spec.

- `bool = obj.GetDepthTextureCompare ()` - Tells if the output of a texture unit with a depth texture uses comparison or not. Comparison happens between D_t the depth texture value in the range [0,1] and with R the interpolated third texture coordinate clamped to range [0,1]. The result of the comparison is noted 'r'. If this flag is false, r=D_t. Initial value is false, as in OpenGL spec. Ignored if the texture object is not a depth texture.

- `obj.SetDepthTextureCompare (bool )` - Tells if the output of a texture unit with a depth texture uses comparison or not. Comparison happens between D_t the depth texture value in the range [0,1] and with R the interpolated third texture coordinate clamped to range [0,1]. The result of the comparison is noted 'r'. If this flag is false, r=D_t. Initial value is false, as in OpenGL spec. Ignored if the texture object is not a depth texture.

- `int = obj.GetDepthTextureCompareFunction ()` - In case DepthTextureCompare is true, specify the comparison function in use. The result of the comparison is noted 'r'. Valid values are: - Value - Lequal: r=R¡=Dt ? 1.0 : 0.0 - Gequal: r=R¿=Dt ? 1.0 : 0.0 - Less: r=R¡D_t ? 1.0 : 0.0 - Greater: r=R¿Dt ? 1.0 : 0.0 - Equal: r=R==Dt ? 1.0 : 0.0 - NotEqual: r=R!=Dt ? 1.0 : 0.0 - AlwaysTrue: r=1.0 - Never: r=0.0 If the magnification of minification factor are not nearest, percentage closer filtering (PCF) is used: R is compared to several D_t and r is the average of the comparisons (it is NOT the average of D_t compared once to R). Initial value is Lequal, as in OpenGL spec. Ignored if the texture object is not a depth texture.

- `obj.SetDepthTextureCompareFunction (int )` - In case DepthTextureCompare is true, specify the comparison function in use. The result of the comparison is noted 'r'. Valid values are: - Value - Lequal: r=R¡=Dt ? 1.0 : 0.0 - Gequal: r=R¿=Dt ? 1.0 : 0.0 - Less: r=R¡D_t ? 1.0 : 0.0 - Greater: r=R¿Dt ? 1.0 : 0.0 - Equal: r=R==Dt ? 1.0 : 0.0 - NotEqual: r=R!=Dt ? 1.0 : 0.0 - AlwaysTrue: r=1.0 - Never: r=0.0 If the magnification of minification factor are not nearest, percentage closer filtering (PCF) is used: R is compared to several D_t and r is the average of the comparisons (it is NOT the average of D_t compared once to R). Initial value is Lequal, as in OpenGL spec. Ignored if the texture object is not a depth texture.

- `int = obj.GetDepthTextureMode ()` - Defines the mapping from depth component 'r' to RGBA components. Ignored if the texture object is not a depth texture. Valid modes are: - Luminance: (R,G,B,A)=(r,r,r,1) - Intensity: (R,G,B,A)=(r,r,r,r) - Alpha: (R.G.B.A)=(0,0,0,r) Initial value is Luminance, as in OpenGL spec.

- `obj.SetDepthTextureMode (int )` - Defines the mapping from depth component 'r' to RGBA components. Ignored if the texture object is not a depth texture. Valid modes are: - Luminance: (R,G,B,A)=(r,r,r,1) - Intensity: (R,G,B,A)=(r,r,r,r) - Alpha: (R.G.B.A)=(0,0,0,r) Initial value is Luminance, as in OpenGL spec.

- `bool = obj.GetGenerateMipmap ()` - Tells the hardware to generate mipmap textures from the first texture image at BaseLevel. Initial value is false, as in OpenGL spec.

- `obj.SetGenerateMipmap (bool )` - Tells the hardware to generate mipmap textures from the first texture image at BaseLevel. Initial value is false, as in OpenGL spec.

- `obj.CopyToFrameBuffer (int srcXmin, int srcYmin, int srcXmax, int srcYmax, int dstXmin, int dstYmin` - Copy a sub-part of the texture (src) in the current framebuffer at location (dstXmin,dstYmin). (dstXmin,dstYmin) is the location of the lower left corner of the rectangle. width and height are the dimensions of the framebuffer. - texture coordinates are sent on texture coordinate processing unit 0. - if the fixed-pipeline fragment shader is used, texturing has to be set on texture image unit 0 and the texture object has to be bound on texture image unit 0. - if a customized fragment shader is used, you

are free to pick the texture image unit you want. You can even have multiple texture objects attached on multiple texture image units. In this case, you call this method only on one of them.

- `obj.CopyFromFrameBuffer (int srcXmin, int srcYmin, int dstXmin, int dstYmin, int width, int height)` - Copy a sub-part of a logical buffer of the framebuffer (color or depth) to the texture object. src is the framebuffer, dst is the texture. (srcXmin,srcYmin) is the location of the lower left corner of the rectangle in the framebuffer. (dstXmin,dstYmin) is the location of the lower left corner of the rectangle in the texture. width and height specifies the size of the rectangle in pixels. If the logical buffer is a color buffer, it has to be selected first with glReadBuffer().

39.191 vtkTransformInterpolator

39.191.1 Usage

This class is used to interpolate a series of 4x4 transformation matrices. Position, scale and orientation (i.e., rotations) are interpolated separately, and can be interpolated linearly or with a spline function. Note that orientation is interpolated using quaternions via SLERP (spherical linear interpolation) or the special vtkQuaternionSpline class.

To use this class, specify at least two pairs of (t,transformation matrix) with the AddTransform() method. Then interpolated the transforms with the InterpolateTransform(t,transform) method, where "t" must be in the range of (min,max) times specified by the AddTransform() method.

By default, spline interpolation is used for the interpolation of the transformation matrices. The position, scale and orientation of the matrices are interpolated with instances of the classes vtkTupleInterpolator (position,scale) and vtkQuaternionInterpolator (rotation). The user can override the interpolation behavior by gaining access to these separate interpolation classes. These interpolator classes (vtkTupleInterpolator and vtkQuaternionInterpolator) can be modified to perform linear versus spline interpolation, and/or different spline basis functions can be specified.

To create an instance of class vtkTransformInterpolator, simply invoke its constructor as follows

```
obj = vtkTransformInterpolator
```

39.191.2 Methods

The class vtkTransformInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTransformInterpolator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTransformInterpolator = obj.NewInstance ()`
- `vtkTransformInterpolator = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetNumberOfTransforms ()` - Return the number of transforms in the list of transforms.
- `double = obj.GetMinimumT ()` - Obtain some information about the interpolation range. The numbers returned (corresponding to parameter t, usually thought of as time) are undefined if the list of transforms is empty.
- `double = obj.GetMaximumT ()` - Obtain some information about the interpolation range. The numbers returned (corresponding to parameter t, usually thought of as time) are undefined if the list of transforms is empty.
- `obj.Initialize ()` - Clear the list of transforms.

- `obj.AddTransform (double t, vtkTransform xform)` - Add another transform to the list of transformations defining the transform function. Note that using the same time t value more than once replaces the previous transform value at t. At least two transforms must be added to define a function. There are variants to this method depending on whether you are adding a vtkTransform, vtkMaxtirx4x4, and/or vtkProp3D.

- `obj.AddTransform (double t, vtkMatrix4x4 matrix)` - Add another transform to the list of transformations defining the transform function. Note that using the same time t value more than once replaces the previous transform value at t. At least two transforms must be added to define a function. There are variants to this method depending on whether you are adding a vtkTransform, vtkMaxtirx4x4, and/or vtkProp3D.

- `obj.AddTransform (double t, vtkProp3D prop3D)` - Add another transform to the list of transformations defining the transform function. Note that using the same time t value more than once replaces the previous transform value at t. At least two transforms must be added to define a function. There are variants to this method depending on whether you are adding a vtkTransform, vtkMaxtirx4x4, and/or vtkProp3D.

- `obj.RemoveTransform (double t)` - Delete the transform at a particular parameter t. If there is no transform defined at location t, then the method does nothing.

- `obj.InterpolateTransform (double t, vtkTransform xform)` - Interpolate the list of transforms and determine a new transform (i.e., fill in the transformation provided). If t is outside the range of (min,max) values, then t is clamped.

- `obj.SetInterpolationType (int )` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the position, scale and orientation interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `int = obj.GetInterpolationTypeMinValue ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the position, scale and orientation interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `int = obj.GetInterpolationTypeMaxValue ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the position, scale and orientation interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `int = obj.GetInterpolationType ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the position, scale and orientation interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `obj.SetInterpolationTypeToLinear ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the position, scale and orientation interpolators. Note that if the InterpolationType is set to "Manual", then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `obj.SetInterpolationTypeToSpline ()` - These are convenience methods to switch between linear and spline interpolation. The methods simply forward the request for linear or spline interpolation to the position, scale and orientation interpolators. Note that if the InterpolationType is set to "Manual",

then the interpolators are expected to be directly manipulated and this class does not forward the request for interpolation type to its interpolators.

- `obj.SetInterpolationTypeToManual ()` - Set/Get the tuple interpolator used to interpolate the position portion of the transformation matrix. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances.
- `obj.SetPositionInterpolator (vtkTupleInterpolator )` - Set/Get the tuple interpolator used to interpolate the position portion of the transformation matrix. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances.
- `vtkTupleInterpolator = obj.GetPositionInterpolator ()` - Set/Get the tuple interpolator used to interpolate the position portion of the transformation matrix. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances.
- `obj.SetScaleInterpolator (vtkTupleInterpolator )` - Set/Get the tuple interpolator used to interpolate the scale portion of the transformation matrix. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances.
- `vtkTupleInterpolator = obj.GetScaleInterpolator ()` - Set/Get the tuple interpolator used to interpolate the scale portion of the transformation matrix. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances.
- `obj.SetRotationInterpolator (vtkQuaternionInterpolator )` - Set/Get the tuple interpolator used to interpolate the orientation portion of the transformation matrix. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances.
- `vtkQuaternionInterpolator = obj.GetRotationInterpolator ()` - Set/Get the tuple interpolator used to interpolate the orientation portion of the transformation matrix. Note that you can modify the behavior of the interpolator (linear vs spline interpolation; change spline basis) by manipulating the interpolator instances.
- `long = obj.GetMTime ()` - Override GetMTime() because we depend on the interpolators which may be modified outside of this class.

39.192 vtkTranslucentPass

39.192.1 Usage

vtkTranslucentPass renders the translucent polygonal geometry of all the props that have the keys contained in vtkRenderState.

This pass expects an initialized depth buffer and color buffer. Initialized buffers means they have been cleared with farest z-value and background color/gradient/transparent color.

To create an instance of class vtkTranslucentPass, simply invoke its constructor as follows

```
obj = vtkTranslucentPass
```

39.192.2 Methods

The class vtkTranslucentPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTranslucentPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTranslucentPass = obj.NewInstance ()`
- `vtkTranslucentPass = obj.SafeDownCast (vtkObject o)`

39.193 vtkTupleInterpolator

39.193.1 Usage

This class is used to interpolate a tuple which may have an arbitrary number of components (but at least one component). The interpolation may be linear in form, or via a subclasses of vtkSpline.

To use this class, begin by specifying the number of components of the tuple and the interpolation function to use. Then specify at least one pair of (t,tuple) with the AddTuple() method. Next interpolate the tuples with the InterpolateTuple(t,tuple) method, where "t" must be in the range of (t_min,t_max) parameter values specified by the AddTuple() method (if not then t is clamped), and tuple[] is filled in by the method (make sure that tuple [] is long enough to hold the interpolated data).

You can control the type of interpolation to use. By default, the interpolation is based on a Kochanek spline. However, other types of splines can be specified. You can also set the interpolation method to linear, in which case the specified spline has no effect on the interpolation.

To create an instance of class vtkTupleInterpolator, simply invoke its constructor as follows

```
obj = vtkTupleInterpolator
```

39.193.2 Methods

The class vtkTupleInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTupleInterpolator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTupleInterpolator = obj.NewInstance ()`
- `vtkTupleInterpolator = obj.SafeDownCast (vtkObject o)`
- `obj.SetNumberOfComponents (int numComp)` - Specify the number of tuple components to interpolate. Note that setting this value discards any previously inserted data.
- `int = obj.GetNumberOfComponents ()` - Specify the number of tuple components to interpolate. Note that setting this value discards any previously inserted data.
- `int = obj.GetNumberOfTuples ()` - Return the number of tuples in the list of tuples to be interpolated.
- `double = obj.GetMinimumT ()` - Obtain some information about the interpolation range. The numbers returned (corresponding to parameter t, usually thought of as time) are undefined if the list of transforms is empty. This is a convenience method for interpolation.
- `double = obj.GetMaximumT ()` - Obtain some information about the interpolation range. The numbers returned (corresponding to parameter t, usually thought of as time) are undefined if the list of transforms is empty. This is a convenience method for interpolation.
- `obj.Initialize ()` - Reset the class so that it contains no (t,tuple) information.

- `obj.AddTuple (double t, double tuple[])` - Add another tuple to the list of tuples to be interpolated. Note that using the same time t value more than once replaces the previous tuple value at t. At least two tuples must be added to define an interpolation function.

- `obj.RemoveTuple (double t)` - Delete the tuple at a particular parameter t. If there is no tuple defined at t, then the method does nothing.

- `obj.InterpolateTuple (double t, double tuple[])` - Interpolate the list of tuples and determine a new tuple (i.e., fill in the tuple provided). If t is outside the range of (min,max) values, then t is clamped. Note that each component of tuple[] is interpolated independently.

- `obj.SetInterpolationType (int type)` - Specify which type of function to use for interpolation. By default spline interpolation (SetInterpolationFunctionToSpline()) is used (i.e., a Kochanek spline) and the InterpolatingSpline instance variable is used to birth the actual interpolation splines via a combination of NewInstance() and DeepCopy(). You may also choose to use linear interpolation by invoking SetInterpolationFunctionToLinear(). Note that changing the type of interpolation causes previously inserted data to be discarded.

- `int = obj.GetInterpolationType ()` - Specify which type of function to use for interpolation. By default spline interpolation (SetInterpolationFunctionToSpline()) is used (i.e., a Kochanek spline) and the InterpolatingSpline instance variable is used to birth the actual interpolation splines via a combination of NewInstance() and DeepCopy(). You may also choose to use linear interpolation by invoking SetInterpolationFunctionToLinear(). Note that changing the type of interpolation causes previously inserted data to be discarded.

- `obj.SetInterpolationTypeToLinear ()` - Specify which type of function to use for interpolation. By default spline interpolation (SetInterpolationFunctionToSpline()) is used (i.e., a Kochanek spline) and the InterpolatingSpline instance variable is used to birth the actual interpolation splines via a combination of NewInstance() and DeepCopy(). You may also choose to use linear interpolation by invoking SetInterpolationFunctionToLinear(). Note that changing the type of interpolation causes previously inserted data to be discarded.

- `obj.SetInterpolationTypeToSpline ()` - If the InterpolationType is set to spline, then this method applies. By default Kochanek interpolation is used, but you can specify any instance of vtkSpline to use. Note that the actual interpolating splines are created by invoking NewInstance() followed by DeepCopy() on the interpolating spline specified here, for each tuple component to interpolate.

- `obj.SetInterpolatingSpline (vtkSpline )` - If the InterpolationType is set to spline, then this method applies. By default Kochanek interpolation is used, but you can specify any instance of vtkSpline to use. Note that the actual interpolating splines are created by invoking NewInstance() followed by DeepCopy() on the interpolating spline specified here, for each tuple component to interpolate.

- `vtkSpline = obj.GetInterpolatingSpline ()` - If the InterpolationType is set to spline, then this method applies. By default Kochanek interpolation is used, but you can specify any instance of vtkSpline to use. Note that the actual interpolating splines are created by invoking NewInstance() followed by DeepCopy() on the interpolating spline specified here, for each tuple component to interpolate.

39.194 vtkUniformVariables

39.194.1 Usage

vtkUniformVariables is a list of uniform variables attached to either a vtkShader2 object or to a vtkShaderProgram2. Uniform variables on a vtkShaderProgram2 override values of uniform variables on a vtkShader2.

To create an instance of class vtkUniformVariables, simply invoke its constructor as follows

```
obj = vtkUniformVariables
```

39.194.2 Methods

The class vtkUniformVariables has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUniformVariables class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUniformVariables = obj.NewInstance ()`
- `vtkUniformVariables = obj.SafeDownCast (vtkObject o)`
- `obj.SetUniformi (string name, int numberOfComponents, int value)` - Set an integer uniform variable.
- `obj.SetUniformf (string name, int numberOfComponents, float value)` - Set an float uniform variable.
- `obj.SetUniformiv (string name, int numberOfComponents, int numberOfElements, int value)` - Set an array of integer uniform variables. The array 'value' is of size 'numberOfElements'*'numberOfComponents.'.
- `obj.SetUniformfv (string name, int numberOfComponents, int numberOfElements, float value)` - Set an array of float uniform variables. The array 'value' is of size 'numberOfElements'*'numberOfComponents.'.
- `obj.SetUniformMatrix (string name, int rows, int columns, float value)` - Set a matrix uniform variable.
- `obj.RemoveUniform (string name)` - Remove uniform 'name' from the list.
- `obj.RemoveAllUniforms ()` - Remove all uniforms from the list.
- `obj.Send (string name, int uniformIndex)` -
- `obj.Start ()` - Place the internal cursor on the first uniform.
- `bool = obj.IsAtEnd ()` - Is the iteration done?
- `string = obj.GetCurrentName ()` - Name of the uniform at the current cursor position.
- `obj.SendCurrentUniform (int uniformIndex)` -
- `obj.Next ()` - Move the cursor to the next uniform.
- `obj.DeepCopy (vtkUniformVariables other)` - Copy all the variables from 'other'. Any existing variable will be deleted first.
- `obj.Merge (vtkUniformVariables other)` - Copy all the variables from 'other'. Any existing variable will be overwritten.

39.195 vtkViewTheme

39.195.1 Usage

This may be set on any subclass of vtkView. The view class will attempt to use the values set in the theme to customize the view. Views will not generally use every aspect of the theme. NOTICE: This class will be deprecated in favor of a more robust solution based on style sheets. Do not become overly-dependent on the functionality of themes.

To create an instance of class vtkViewTheme, simply invoke its constructor as follows

```
obj = vtkViewTheme
```

39.195.2 Methods

The class vtkViewTheme has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkViewTheme class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkViewTheme = obj.NewInstance ()`
- `vtkViewTheme = obj.SafeDownCast (vtkObject o)`
- `obj.SetPointSize (double )` - The size of points or vertices
- `double = obj.GetPointSize ()` - The size of points or vertices
- `obj.SetLineWidth (double )` - The width of lines or edges
- `double = obj.GetLineWidth ()` - The width of lines or edges
- `obj.SetPointColor (double , double , double )` - The color and opacity of points or vertices when not mapped through a lookup table.
- `obj.SetPointColor (double  a[3])` - The color and opacity of points or vertices when not mapped through a lookup table.
- `double = obj. GetPointColor ()` - The color and opacity of points or vertices when not mapped through a lookup table.
- `obj.SetPointOpacity (double )` - The color and opacity of points or vertices when not mapped through a lookup table.
- `double = obj.GetPointOpacity ()` - The color and opacity of points or vertices when not mapped through a lookup table.
- `obj.SetPointHueRange (double mn, double mx)` - The ranges to use in the point lookup table. You may also do this by accessing the point lookup table directly with GetPointLookupTable() and calling these methods.
- `obj.SetPointHueRange (double rng[2])` - The ranges to use in the point lookup table. You may also do this by accessing the point lookup table directly with GetPointLookupTable() and calling these methods.
- `obj.GetPointHueRange (double rng[2])` - The ranges to use in the point lookup table. You may also do this by accessing the point lookup table directly with GetPointLookupTable() and calling these methods.
- `obj.SetPointSaturationRange (double mn, double mx)`
- `obj.SetPointSaturationRange (double rng[2])`
- `obj.GetPointSaturationRange (double rng[2])`
- `obj.SetPointValueRange (double mn, double mx)`
- `obj.SetPointValueRange (double rng[2])`
- `obj.GetPointValueRange (double rng[2])`
- `obj.SetPointAlphaRange (double mn, double mx)`

- `obj.SetPointAlphaRange (double rng[2])`
- `obj.GetPointAlphaRange (double rng[2])`
- `vtkScalarsToColors = obj.GetPointLookupTable ()` - Set/Get the point lookup table.
- `obj.SetPointLookupTable (vtkScalarsToColors lut)` - Set/Get the point lookup table.
- `obj.SetScalePointLookupTable (bool )` - Whether to scale the lookup table to fit the range of the data.
- `bool = obj.GetScalePointLookupTable ()` - Whether to scale the lookup table to fit the range of the data.
- `obj.ScalePointLookupTableOn ()` - Whether to scale the lookup table to fit the range of the data.
- `obj.ScalePointLookupTableOff ()` - Whether to scale the lookup table to fit the range of the data.
- `obj.SetCellColor (double , double , double )` - The color and opacity of cells or edges when not mapped through a lookup table.
- `obj.SetCellColor (double  a[3])` - The color and opacity of cells or edges when not mapped through a lookup table.
- `double = obj. GetCellColor ()` - The color and opacity of cells or edges when not mapped through a lookup table.
- `obj.SetCellOpacity (double )` - The color and opacity of cells or edges when not mapped through a lookup table.
- `double = obj.GetCellOpacity ()` - The color and opacity of cells or edges when not mapped through a lookup table.
- `obj.SetCellHueRange (double mn, double mx)` - The ranges to use in the cell lookup table. You may also do this by accessing the cell lookup table directly with GetCellLookupTable() and calling these methods.
- `obj.SetCellHueRange (double rng[2])` - The ranges to use in the cell lookup table. You may also do this by accessing the cell lookup table directly with GetCellLookupTable() and calling these methods.
- `obj.GetCellHueRange (double rng[2])` - The ranges to use in the cell lookup table. You may also do this by accessing the cell lookup table directly with GetCellLookupTable() and calling these methods.
- `obj.SetCellSaturationRange (double mn, double mx)`
- `obj.SetCellSaturationRange (double rng[2])`
- `obj.GetCellSaturationRange (double rng[2])`
- `obj.SetCellValueRange (double mn, double mx)`
- `obj.SetCellValueRange (double rng[2])`
- `obj.GetCellValueRange (double rng[2])`
- `obj.SetCellAlphaRange (double mn, double mx)`
- `obj.SetCellAlphaRange (double rng[2])`
- `obj.GetCellAlphaRange (double rng[2])`
- `vtkScalarsToColors = obj.GetCellLookupTable ()` - Set/Get the cell lookup table.

- `obj.SetCellLookupTable (vtkScalarsToColors lut)` - Set/Get the cell lookup table.
- `obj.SetScaleCellLookupTable (bool )` - Whether to scale the lookup table to fit the range of the data.
- `bool = obj.GetScaleCellLookupTable ()` - Whether to scale the lookup table to fit the range of the data.
- `obj.ScaleCellLookupTableOn ()` - Whether to scale the lookup table to fit the range of the data.
- `obj.ScaleCellLookupTableOff ()` - Whether to scale the lookup table to fit the range of the data.
- `obj.SetOutlineColor (double , double , double )` - The color of any outlines in the view.
- `obj.SetOutlineColor (double a[3])` - The color of any outlines in the view.
- `double = obj. GetOutlineColor ()` - The color of any outlines in the view.
- `obj.SetSelectedPointColor (double , double , double )` - The color of selected points or vertices.
- `obj.SetSelectedPointColor (double a[3])` - The color of selected points or vertices.
- `double = obj. GetSelectedPointColor ()` - The color of selected points or vertices.
- `obj.SetSelectedPointOpacity (double )` - The color of selected points or vertices.
- `double = obj.GetSelectedPointOpacity ()` - The color of selected points or vertices.
- `obj.SetSelectedCellColor (double , double , double )` - The color of selected cells or edges.
- `obj.SetSelectedCellColor (double a[3])` - The color of selected cells or edges.
- `double = obj. GetSelectedCellColor ()` - The color of selected cells or edges.
- `obj.SetSelectedCellOpacity (double )` - The color of selected cells or edges.
- `double = obj.GetSelectedCellOpacity ()` - The color of selected cells or edges.
- `obj.SetBackgroundColor (double , double , double )` - The view background color.
- `obj.SetBackgroundColor (double a[3])` - The view background color.
- `double = obj. GetBackgroundColor ()` - The view background color.
- `obj.SetBackgroundColor2 (double , double , double )` - The second background color (for gradients).
- `obj.SetBackgroundColor2 (double a[3])` - The second background color (for gradients).
- `double = obj. GetBackgroundColor2 ()` - The second background color (for gradients).
- `obj.SetPointTextProperty (vtkTextProperty tprop)` - The text property to use for labelling points/vertices.
- `vtkTextProperty = obj.GetPointTextProperty ()` - The text property to use for labelling points/vertices.
- `obj.SetCellTextProperty (vtkTextProperty tprop)` - The text property to use for labelling edges/cells.
- `vtkTextProperty = obj.GetCellTextProperty ()` - The text property to use for labelling edges/cells.
- `obj.SetVertexLabelColor (double r, double g, double b)` - The color to use for labelling graph vertices. This is deprecated. Use GetPointTextProperty()-¿SetColor() instead.
- `obj.SetVertexLabelColor (double c[3])` - The color to use for labelling graph vertices. This is deprecated. Use GetPointTextProperty()-¿SetColor() instead.

- `obj.GetVertexLabelColor (double c[3])` - The color to use for labelling graph edges. This is deprecated. Use GetCellTextProperty()-¿SetColor() instead.

- `obj.SetEdgeLabelColor (double r, double g, double b)` - The color to use for labelling graph edges. This is deprecated. Use GetCellTextProperty()-¿SetColor() instead.

- `obj.SetEdgeLabelColor (double c[3])` - The color to use for labelling graph edges. This is deprecated. Use GetCellTextProperty()-¿SetColor() instead.

- `obj.GetEdgeLabelColor (double c[3])` - Convenience methods for creating some default view themes. The return reference is reference-counted, so you will have to call Delete() on the reference when you are finished with it.

- `bool = obj.LookupMatchesPointTheme (vtkScalarsToColors s2c)` - Whether a given lookup table matches the point or cell theme of this theme.

- `bool = obj.LookupMatchesCellTheme (vtkScalarsToColors s2c)` - Whether a given lookup table matches the point or cell theme of this theme.

39.196 vtkVisibilitySort

39.196.1 Usage

vtkVisibilitySort encapsulates a method for depth sorting the cells of a vtkDataSet for a given viewpoint. It should be noted that subclasses are not required to give an absolutely correct sorting. Many types of unstructured grids may have sorting cycles, meaning that there is no possible correct sorting. Some subclasses also only give an approximate sorting in the interest of speed.

.SECTION Note The Input field of this class tends to causes reference cycles. To help break these cycles, garbage collection is enabled on this object and the input parameter is traced. For this to work, though, an object in the loop holding the visibility sort should also report that to the garbage collector.

To create an instance of class vtkVisibilitySort, simply invoke its constructor as follows

```
obj = vtkVisibilitySort
```

39.196.2 Methods

The class vtkVisibilitySort has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVisibilitySort class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkVisibilitySort = obj.NewInstance ()`

- `vtkVisibilitySort = obj.SafeDownCast (vtkObject o)`

- `obj.InitTraversal ()` - To facilitate incremental sorting algorithms, the cells are retrieved in an iteration process. That is, call InitTraversal to start the iteration and call GetNextCells to get the cell IDs in order. However, for efficiencies sake, GetNextCells returns an ordered list of several id's in once call (but not necessarily all). GetNextCells will return NULL once the entire sorted list is output. The vtkIdTypeArray returned from GetNextCells is a cached array, so do not delete it. At the same note, do not expect the array to be valid after subsequent calls to GetNextCells.

- `vtkIdTypeArray = obj.GetNextCells ()` - To facilitate incremental sorting algorithms, the cells are retrieved in an iteration process. That is, call InitTraversal to start the iteration and call GetNextCells to get the cell IDs in order. However, for efficiencies sake, GetNextCells returns an ordered list of several id's in once call (but not necessarily all). GetNextCells will return NULL once the entire sorted list is output. The vtkIdTypeArray returned from GetNextCells is a cached array, so do not delete it. At the same note, do not expect the array to be valid after subsequent calls to GetNextCells.
- `obj.SetMaxCellsReturned (int )` - Set/Get the maximum number of cells that GetNextCells will return in one invocation.
- `int = obj.GetMaxCellsReturnedMinValue ()` - Set/Get the maximum number of cells that GetNextCells will return in one invocation.
- `int = obj.GetMaxCellsReturnedMaxValue ()` - Set/Get the maximum number of cells that GetNextCells will return in one invocation.
- `int = obj.GetMaxCellsReturned ()` - Set/Get the maximum number of cells that GetNextCells will return in one invocation.
- `obj.SetModelTransform (vtkMatrix4x4 mat)` - Set/Get the matrix that transforms from object space to world space. Generally, you get this matrix from a call to GetMatrix of a vtkProp3D (vtkActor).
- `vtkMatrix4x4 = obj.GetModelTransform ()` - Set/Get the matrix that transforms from object space to world space. Generally, you get this matrix from a call to GetMatrix of a vtkProp3D (vtkActor).
- `vtkMatrix4x4 = obj.GetInverseModelTransform ()`
- `obj.SetCamera (vtkCamera camera)` - Set/Get the camera that specifies the viewing parameters.
- `vtkCamera = obj.GetCamera ()` - Set/Get the camera that specifies the viewing parameters.
- `obj.SetInput (vtkDataSet data)` - Set/Get the data set containing the cells to sort.
- `vtkDataSet = obj.GetInput ()` - Set/Get the data set containing the cells to sort.
- `int = obj.GetDirection ()` - Set/Get the sorting direction. Be default, the direction is set to back to front.
- `obj.SetDirection (int )` - Set/Get the sorting direction. Be default, the direction is set to back to front.
- `obj.SetDirectionToBackToFront ()` - Set/Get the sorting direction. Be default, the direction is set to back to front.
- `obj.SetDirectionToFrontToBack ()` - Overwritten to enable garbage collection.
- `obj.Register (vtkObjectBase o)` - Overwritten to enable garbage collection.
- `obj.UnRegister (vtkObjectBase o)` - Overwritten to enable garbage collection.

39.197 vtkVisibleCellSelector

39.197.1 Usage

DEPRECATED: Please refer to vtkHardwareSelector instead. This class can be used to determine what cells are visible within a given rectangle of the RenderWindow. To use it, call in order, SetRenderer(), SetArea(), SetProcessorId(), SetRenderPasses(), and then Select(). Select will cause the attached vtkRenderer to render in a special color mode, where each cell is given it own color so that later inspection of the Rendered Pixels can determine what cells are visible. In practice up to five different rendering passes may occur depending

on your choices in SetRenderPasses. After Select(), a list of the visible cells can be obtained by calling GetSelectedIds().

Limitations: Antialiasing will break this class. If your graphics card settings force their use this class will return invalid results.

Currently only cells from PolyDataMappers can be selected from. When vtkRenderer is put into a SelectMode, it temporarily swaps in a new vtkIdentColoredPainter to do the color index rendering of each cell in each vtkProp that it renders. Until alternatives to vtkIdentColoredPainter exist that can do a similar coloration of other vtkDataSet types, only polygonal data can be selected. If you need to select other data types, consider using vtkDataSetMapper and turning on it's PassThroughCellIds feature, or using vtkFrustumExtractor.

Only Opaque geometry in Actors is selected from. Assemblies and LODMappers are not currently supported.

During selection, visible datasets that can not be selected from are temporarily hidden so as not to produce invalid indices from their colors.

To create an instance of class vtkVisibleCellSelector, simply invoke its constructor as follows

```
obj = vtkVisibleCellSelector
```

39.197.2 Methods

The class vtkVisibleCellSelector has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVisibleCellSelector class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVisibleCellSelector = obj.NewInstance ()`
- `vtkVisibleCellSelector = obj.SafeDownCast (vtkObject o)`
- `obj.SetRenderer (vtkRenderer )` - Call to let this know where to select within.
- `obj.SetArea (int x0, int y0, int x1, int y1)` - Call to set the selection area region. This crops the selected area to the renderers pixel limits.
- `obj.SetProcessorId (int pid)` - Call to let this know what processor number to render as in the processor select pass. Internally this adds 1 to pid because 0 is reserved for miss.
- `int = obj.GetProcessorId ()` - Call to let this know what processor number to render as in the processor select pass. Internally this adds 1 to pid because 0 is reserved for miss.
- `obj.SetRenderPasses (int DoProcessor, int DoActor, int DoCellIdHi, int DoCellIdMid, int DoCellIdLo,` - Call to let this know what selection render passes to do. If you have only one processor or one actor, you can leave DoProcessor and DoActor as false (the default). If you have less than $2\hat{4}8$ cells in any actor, you do not need the CellIdHi pass, or similarly if you have less than $2\hat{2}4$ cells, you do not need DoCellIdMid. The DoPointId will enable another render pass for determining visible vertices.
- `obj.Select ()` - Execute the selection algorithm.
- `obj.GetSelectedIds (vtkIdTypeArray ToCopyInto)` - After Select(), this will return the list of selected Ids. The ProcessorId and Actor Id are returned in the first two components. The CellId is returned in the last two components (only 64 bits total).
- `obj.GetSelectedIds (vtkSelection ToCopyInto)` - After Select(), this will return the list of selected Ids.

- `obj.GetSelectedVertices (vtkIdTypeArray VertexPointers, vtkIdTypeArray VertexIds)` - After Select(), (assuming DoVertexId is on), the will return arrays that describe which cell vertices are visible. The VertexPointers array contains one index into the VertexIds array for every visible cell. Any index may be -1 in which case no vertices were visible for that cell. The VertexIds array contains a set of integers for each cell that has visible vertices. The first entry in the set is the number of visible vertices. The rest are visible vertex ranks. A set such at 2,0,4, means that a particular polygon's first and fifth vertices were visible.
- `vtkProp = obj.GetActorFromId (vtkIdType id)` - After a select, this will return a pointer to the actor corresponding to a particular id. This will return NULL if id is out of range.
- `obj.PrintSelectedIds (vtkIdTypeArray IdsToPrint)` - For debugging - prints out the list of selected ids.

39.198 vtkVolume

39.198.1 Usage

vtkVolume is used to represent a volumetric entity in a rendering scene. It inherits functions related to the volume's position, orientation and origin from vtkProp3D. The volume maintains a reference to the volumetric data (i.e., the volume mapper). The volume also contains a reference to a volume property which contains all common volume rendering parameters.

To create an instance of class vtkVolume, simply invoke its constructor as follows

```
obj = vtkVolume
```

39.198.2 Methods

The class vtkVolume has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolume class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolume = obj.NewInstance ()`
- `vtkVolume = obj.SafeDownCast (vtkObject o)`
- `obj.SetMapper (vtkAbstractVolumeMapper mapper)` - Set/Get the volume mapper.
- `vtkAbstractVolumeMapper = obj.GetMapper ()` - Set/Get the volume mapper.
- `obj.SetProperty (vtkVolumeProperty property)` - Set/Get the volume property.
- `vtkVolumeProperty = obj.GetProperty ()` - Set/Get the volume property.
- `obj.GetVolumes (vtkPropCollection vc)` - For some exporters and other other operations we must be able to collect all the actors or volumes. This method is used in that process.
- `obj.Update ()` - Update the volume rendering pipeline by updating the volume mapper
- `double = obj.GetBounds ()` - Get the bounds - either all six at once (xmin, xmax, ymin, ymax, zmin, zmax) or one at a time.
- `obj.GetBounds (double bounds[6])` - Get the bounds - either all six at once (xmin, xmax, ymin, ymax, zmin, zmax) or one at a time.

- `double = obj.GetMinXBound ()` - Get the bounds - either all six at once (xmin, xmax, ymin, ymax, zmin, zmax) or one at a time.
- `double = obj.GetMaxXBound ()` - Get the bounds - either all six at once (xmin, xmax, ymin, ymax, zmin, zmax) or one at a time.
- `double = obj.GetMinYBound ()` - Get the bounds - either all six at once (xmin, xmax, ymin, ymax, zmin, zmax) or one at a time.
- `double = obj.GetMaxYBound ()` - Get the bounds - either all six at once (xmin, xmax, ymin, ymax, zmin, zmax) or one at a time.
- `double = obj.GetMinZBound ()` - Get the bounds - either all six at once (xmin, xmax, ymin, ymax, zmin, zmax) or one at a time.
- `double = obj.GetMaxZBound ()` - Get the bounds - either all six at once (xmin, xmax, ymin, ymax, zmin, zmax) or one at a time.
- `long = obj.GetMTime ()` - Return the MTime also considering the property etc.
- `long = obj.GetRedrawMTime ()` - Return the mtime of anything that would cause the rendered image to appear differently. Usually this involves checking the mtime of the prop plus anything else it depends on such as properties, mappers, etc.
- `obj.ShallowCopy (vtkProp prop)` - Shallow copy of this vtkVolume. Overloads the virtual vtkProp method.

39.199 vtkVolumeCollection

39.199.1 Usage

vtkVolumeCollection represents and provides methods to manipulate a list of volumes (i.e., vtkVolume and subclasses). The list is unsorted and duplicate entries are not prevented.

To create an instance of class vtkVolumeCollection, simply invoke its constructor as follows

```
obj = vtkVolumeCollection
```

39.199.2 Methods

The class vtkVolumeCollection has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeCollection class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeCollection = obj.NewInstance ()`
- `vtkVolumeCollection = obj.SafeDownCast (vtkObject o)`
- `obj.AddItem (vtkVolume a)` - Get the next Volume in the list. Return NULL when at the end of the list.
- `vtkVolume = obj.GetNextVolume ()` - Get the next Volume in the list. Return NULL when at the end of the list.
- `vtkVolume = obj.GetNextItem ()` - Access routine provided for compatibility with previous versions of VTK. Please use the GetNextVolume() variant where possible.

39.200 vtkVolumeProperty

39.200.1 Usage

vtkVolumeProperty is used to represent common properties associated with volume rendering. This includes properties for determining the type of interpolation to use when sampling a volume, the color of a volume, the scalar opacity of a volume, the gradient opacity of a volume, and the shading parameters of a volume.

When the scalar opacity or the gradient opacity of a volume is not set, then the function is defined to be a constant value of 1.0. When a scalar and gradient opacity are both set simultaneously, then the opacity is defined to be the product of the scalar opacity and gradient opacity transfer functions.

Most properties can be set per "component" for volume mappers that support multiple independent components. If you are using 2 component data as LV or 4 component data as RGBV (as specified in the mapper) only the first scalar opacity and gradient opacity transfer functions will be used (and all color functions will be ignored). Omitting the index parameter on the Set/Get methods will access index = 0.

To create an instance of class vtkVolumeProperty, simply invoke its constructor as follows

```
obj = vtkVolumeProperty
```

39.200.2 Methods

The class vtkVolumeProperty has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeProperty class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeProperty = obj.NewInstance ()`
- `vtkVolumeProperty = obj.SafeDownCast (vtkObject o)`
- `obj.DeepCopy (vtkVolumeProperty p)`
- `long = obj.GetMTime ()` - Get the modified time for this object (or the properties registered with this object).
- `obj.SetIndependentComponents (int )` - Does the data have independent components, or do some define color only? If IndependentComponents is On (the default) then each component will be independently passed through a lookup table to determine RGBA, shaded. Some volume Mappers can handle 1 to 4 component unsigned char or unsigned short data (see each mapper header file to determine functionality). If IndependentComponents is Off, then you must have either 2 or 4 component data. For 2 component data, the first is passed through the first color transfer function and the second component is passed through the first opacity transfer function. Normals will be generated off of the second component. For 4 component data, the first three will directly represent RGB (no lookup table). The fourth component will be passed through the first scalar opacity transfer function for opacity. Normals will be generated from the fourth component.
- `int = obj.GetIndependentComponentsMinValue ()` - Does the data have independent components, or do some define color only? If IndependentComponents is On (the default) then each component will be independently passed through a lookup table to determine RGBA, shaded. Some volume Mappers can handle 1 to 4 component unsigned char or unsigned short data (see each mapper header file to determine functionality). If IndependentComponents is Off, then you must have either 2 or 4 component data. For 2 component data, the first is passed through the first color transfer function and the second component is passed through the first opacity transfer function. Normals will be generated off of the second component. For 4 component data, the first three will directly represent RGB (no lookup table). The fourth component will be passed through the first scalar opacity transfer function for opacity. Normals will be generated from the fourth component.

- `int = obj.GetIndependentComponentsMaxValue ()` - Does the data have independent components, or do some define color only? If IndependentComponents is On (the default) then each component will be independently passed through a lookup table to determine RGBA, shaded. Some volume Mappers can handle 1 to 4 component unsigned char or unsigned short data (see each mapper header file to determine functionality). If IndependentComponents is Off, then you must have either 2 or 4 component data. For 2 component data, the first is passed through the first color transfer function and the second component is passed through the first opacity transfer function. Normals will be generated off of the second component. For 4 component data, the first three will directly represent RGB (no lookup table). The fourth component will be passed through the first scalar opacity transfer function for opacity. Normals will be generated from the fourth component.

- `int = obj.GetIndependentComponents ()` - Does the data have independent components, or do some define color only? If IndependentComponents is On (the default) then each component will be independently passed through a lookup table to determine RGBA, shaded. Some volume Mappers can handle 1 to 4 component unsigned char or unsigned short data (see each mapper header file to determine functionality). If IndependentComponents is Off, then you must have either 2 or 4 component data. For 2 component data, the first is passed through the first color transfer function and the second component is passed through the first opacity transfer function. Normals will be generated off of the second component. For 4 component data, the first three will directly represent RGB (no lookup table). The fourth component will be passed through the first scalar opacity transfer function for opacity. Normals will be generated from the fourth component.

- `obj.IndependentComponentsOn ()` - Does the data have independent components, or do some define color only? If IndependentComponents is On (the default) then each component will be independently passed through a lookup table to determine RGBA, shaded. Some volume Mappers can handle 1 to 4 component unsigned char or unsigned short data (see each mapper header file to determine functionality). If IndependentComponents is Off, then you must have either 2 or 4 component data. For 2 component data, the first is passed through the first color transfer function and the second component is passed through the first opacity transfer function. Normals will be generated off of the second component. For 4 component data, the first three will directly represent RGB (no lookup table). The fourth component will be passed through the first scalar opacity transfer function for opacity. Normals will be generated from the fourth component.

- `obj.IndependentComponentsOff ()` - Does the data have independent components, or do some define color only? If IndependentComponents is On (the default) then each component will be independently passed through a lookup table to determine RGBA, shaded. Some volume Mappers can handle 1 to 4 component unsigned char or unsigned short data (see each mapper header file to determine functionality). If IndependentComponents is Off, then you must have either 2 or 4 component data. For 2 component data, the first is passed through the first color transfer function and the second component is passed through the first opacity transfer function. Normals will be generated off of the second component. For 4 component data, the first three will directly represent RGB (no lookup table). The fourth component will be passed through the first scalar opacity transfer function for opacity. Normals will be generated from the fourth component.

- `obj.SetInterpolationType (int )` - Set the interpolation type for sampling a volume. Initial value is VTK_NEAREST_INTERPOLATION.

- `int = obj.GetInterpolationTypeMinValue ()` - Set the interpolation type for sampling a volume. Initial value is VTK_NEAREST_INTERPOLATION.

- `int = obj.GetInterpolationTypeMaxValue ()` - Set the interpolation type for sampling a volume. Initial value is VTK_NEAREST_INTERPOLATION.

- `int = obj.GetInterpolationType ()` - Set the interpolation type for sampling a volume. Initial value is VTK_NEAREST_INTERPOLATION.

- `obj.SetInterpolationTypeToNearest ()` - Set the interpolation type for sampling a volume. Initial value is VTK_NEAREST_INTERPOLATION.

- `obj.SetInterpolationTypeToLinear ()` - Set the interpolation type for sampling a volume. Initial value is VTK_NEAREST_INTERPOLATION.
- `string = obj.GetInterpolationTypeAsString (void )` - Set the interpolation type for sampling a volume. Initial value is VTK_NEAREST_INTERPOLATION.
- `obj.SetComponentWeight (int index, double value)` - Set/Get the scalar component weights
- `double = obj.GetComponentWeight (int index)` - Set/Get the scalar component weights
- `obj.SetColor (int index, vtkPiecewiseFunction function)` - Set the color of a volume to a gray level transfer function for the component indicated by index. This will set the color channels for this component to 1.
- `obj.SetColor (vtkPiecewiseFunction f)` - Set the color of a volume to a gray level transfer function for the component indicated by index. This will set the color channels for this component to 1.
- `obj.SetColor (int index, vtkColorTransferFunction function)` - Set the color of a volume to an RGB transfer function for the component indicated by index. This will set the color channels for this component to 3. This will also recompute the color channels
- `obj.SetColor (vtkColorTransferFunction f)` - Set the color of a volume to an RGB transfer function for the component indicated by index. This will set the color channels for this component to 3. This will also recompute the color channels
- `int = obj.GetColorChannels (int index)` - Get the number of color channels in the transfer function for the given component.
- `int = obj.GetColorChannels ()` - Get the number of color channels in the transfer function for the given component.
- `vtkPiecewiseFunction = obj.GetGrayTransferFunction (int index)` - Get the gray transfer function. If no transfer function has been set for this component, a default one is created and returned.
- `vtkPiecewiseFunction = obj.GetGrayTransferFunction ()` - Get the gray transfer function. If no transfer function has been set for this component, a default one is created and returned.
- `vtkColorTransferFunction = obj.GetRGBTransferFunction (int index)` - Get the RGB transfer function for the given component. If no transfer function has been set for this component, a default one is created and returned.
- `vtkColorTransferFunction = obj.GetRGBTransferFunction ()` - Get the RGB transfer function for the given component. If no transfer function has been set for this component, a default one is created and returned.
- `obj.SetScalarOpacity (int index, vtkPiecewiseFunction function)` - Set the opacity of a volume to an opacity transfer function based on scalar value for the component indicated by index.
- `obj.SetScalarOpacity (vtkPiecewiseFunction f)` - Set the opacity of a volume to an opacity transfer function based on scalar value for the component indicated by index.
- `vtkPiecewiseFunction = obj.GetScalarOpacity (int index)` - Get the scalar opacity transfer function for the given component. If no transfer function has been set for this component, a default one is created and returned.
- `vtkPiecewiseFunction = obj.GetScalarOpacity ()` - Get the scalar opacity transfer function for the given component. If no transfer function has been set for this component, a default one is created and returned.

- `obj.SetScalarOpacityUnitDistance (int index, double distance)` - Set/Get the unit distance on which the scalar opacity transfer function is defined. By default this is 1.0, meaning that over a distance of 1.0 units, a given opacity (from the transfer function) is accumulated. This is adjusted for the actual sampling distance during rendering.

- `obj.SetScalarOpacityUnitDistance (double distance)` - Set/Get the unit distance on which the scalar opacity transfer function is defined. By default this is 1.0, meaning that over a distance of 1.0 units, a given opacity (from the transfer function) is accumulated. This is adjusted for the actual sampling distance during rendering.

- `double = obj.GetScalarOpacityUnitDistance (int index)` - Set/Get the unit distance on which the scalar opacity transfer function is defined. By default this is 1.0, meaning that over a distance of 1.0 units, a given opacity (from the transfer function) is accumulated. This is adjusted for the actual sampling distance during rendering.

- `double = obj.GetScalarOpacityUnitDistance ()` - Set the opacity of a volume to an opacity transfer function based on gradient magnitude for the given component.

- `obj.SetGradientOpacity (int index, vtkPiecewiseFunction function)` - Set the opacity of a volume to an opacity transfer function based on gradient magnitude for the given component.

- `obj.SetGradientOpacity (vtkPiecewiseFunction function)` - Get the gradient magnitude opacity transfer function for the given component. If no transfer function has been set for this component, a default one is created and returned. This default function is always returned if DisableGradientOpacity is On for that component.

- `vtkPiecewiseFunction = obj.GetGradientOpacity (int index)` - Get the gradient magnitude opacity transfer function for the given component. If no transfer function has been set for this component, a default one is created and returned. This default function is always returned if DisableGradientOpacity is On for that component.

- `vtkPiecewiseFunction = obj.GetGradientOpacity ()` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `obj.SetDisableGradientOpacity (int index, int value)` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `obj.SetDisableGradientOpacity (int value)` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `obj.DisableGradientOpacityOn (int index)` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `obj.DisableGradientOpacityOn ()` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `obj.DisableGradientOpacityOff (int index)` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function

for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `obj.DisableGradientOpacityOff ()` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `int = obj.GetDisableGradientOpacity (int index)` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `int = obj.GetDisableGradientOpacity ()` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `vtkPiecewiseFunction = obj.GetStoredGradientOpacity (int index)` - Enable/Disable the gradient opacity function for the given component. If set to true, any call to GetGradientOpacity() will return a default function for this component. Note that the gradient opacity function is still stored, it is not set or reset and can be retrieved using GetStoredGradientOpacity().

- `vtkPiecewiseFunction = obj.GetStoredGradientOpacity ()` - Set/Get the shading of a volume. If shading is turned off, then the mapper for the volume will not perform shading calculations. If shading is turned on, the mapper may perform shading calculations - in some cases shading does not apply (for example, in a maximum intensity projection) and therefore shading will not be performed even if this flag is on. For a compositing type of mapper, turning shading off is generally the same as setting ambient=1, diffuse=0, specular=0. Shading can be independently turned on/off per component.

- `obj.SetShade (int index, int value)` - Set/Get the shading of a volume. If shading is turned off, then the mapper for the volume will not perform shading calculations. If shading is turned on, the mapper may perform shading calculations - in some cases shading does not apply (for example, in a maximum intensity projection) and therefore shading will not be performed even if this flag is on. For a compositing type of mapper, turning shading off is generally the same as setting ambient=1, diffuse=0, specular=0. Shading can be independently turned on/off per component.

- `obj.SetShade (int value)` - Set/Get the shading of a volume. If shading is turned off, then the mapper for the volume will not perform shading calculations. If shading is turned on, the mapper may perform shading calculations - in some cases shading does not apply (for example, in a maximum intensity projection) and therefore shading will not be performed even if this flag is on. For a compositing type of mapper, turning shading off is generally the same as setting ambient=1, diffuse=0, specular=0. Shading can be independently turned on/off per component.

- `int = obj.GetShade (int index)` - Set/Get the shading of a volume. If shading is turned off, then the mapper for the volume will not perform shading calculations. If shading is turned on, the mapper may perform shading calculations - in some cases shading does not apply (for example, in a maximum intensity projection) and therefore shading will not be performed even if this flag is on. For a compositing type of mapper, turning shading off is generally the same as setting ambient=1, diffuse=0, specular=0. Shading can be independently turned on/off per component.

- `int = obj.GetShade ()` - Set/Get the shading of a volume. If shading is turned off, then the mapper for the volume will not perform shading calculations. If shading is turned on, the mapper may perform shading calculations - in some cases shading does not apply (for example, in a maximum intensity projection) and therefore shading will not be performed even if this flag is on. For a compositing type of mapper, turning shading off is generally the same as setting ambient=1, diffuse=0, specular=0. Shading can be independently turned on/off per component.

- `obj.ShadeOn (int index)` - Set/Get the shading of a volume. If shading is turned off, then the mapper for the volume will not perform shading calculations. If shading is turned on, the mapper may perform shading calculations - in some cases shading does not apply (for example, in a maximum intensity projection) and therefore shading will not be performed even if this flag is on. For a compositing type of mapper, turning shading off is generally the same as setting ambient=1, diffuse=0, specular=0. Shading can be independently turned on/off per component.

- `obj.ShadeOn ()` - Set/Get the shading of a volume. If shading is turned off, then the mapper for the volume will not perform shading calculations. If shading is turned on, the mapper may perform shading calculations - in some cases shading does not apply (for example, in a maximum intensity projection) and therefore shading will not be performed even if this flag is on. For a compositing type of mapper, turning shading off is generally the same as setting ambient=1, diffuse=0, specular=0. Shading can be independently turned on/off per component.

- `obj.ShadeOff (int index)` - Set/Get the shading of a volume. If shading is turned off, then the mapper for the volume will not perform shading calculations. If shading is turned on, the mapper may perform shading calculations - in some cases shading does not apply (for example, in a maximum intensity projection) and therefore shading will not be performed even if this flag is on. For a compositing type of mapper, turning shading off is generally the same as setting ambient=1, diffuse=0, specular=0. Shading can be independently turned on/off per component.

- `obj.ShadeOff ()` - Set/Get the ambient lighting coefficient.

- `obj.SetAmbient (int index, double value)` - Set/Get the ambient lighting coefficient.

- `obj.SetAmbient (double value)` - Set/Get the ambient lighting coefficient.

- `double = obj.GetAmbient (int index)` - Set/Get the ambient lighting coefficient.

- `double = obj.GetAmbient ()` - Set/Get the diffuse lighting coefficient.

- `obj.SetDiffuse (int index, double value)` - Set/Get the diffuse lighting coefficient.

- `obj.SetDiffuse (double value)` - Set/Get the diffuse lighting coefficient.

- `double = obj.GetDiffuse (int index)` - Set/Get the diffuse lighting coefficient.

- `double = obj.GetDiffuse ()` - Set/Get the specular lighting coefficient.

- `obj.SetSpecular (int index, double value)` - Set/Get the specular lighting coefficient.

- `obj.SetSpecular (double value)` - Set/Get the specular lighting coefficient.

- `double = obj.GetSpecular (int index)` - Set/Get the specular lighting coefficient.

- `double = obj.GetSpecular ()` - Set/Get the specular power.

- `obj.SetSpecularPower (int index, double value)` - Set/Get the specular power.

- `obj.SetSpecularPower (double value)` - Set/Get the specular power.

- `double = obj.GetSpecularPower (int index)` - Set/Get the specular power.

- `double = obj.GetSpecularPower ()`

39.201 vtkVolumetricPass

39.201.1 Usage

vtkVolumetricPass renders the volumetric geometry of all the props that have the keys contained in vtkRenderState.

This pass expects an initialized depth buffer and color buffer. Initialized buffers means they have been cleared with farest z-value and background color/gradient/transparent color.

To create an instance of class vtkVolumetricPass, simply invoke its constructor as follows

```
obj = vtkVolumetricPass
```

39.201.2 Methods

The class vtkVolumetricPass has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumetricPass class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumetricPass = obj.NewInstance ()`
- `vtkVolumetricPass = obj.SafeDownCast (vtkObject o)`

39.202 vtkVRMLExporter

39.202.1 Usage

vtkVRMLExporter is a concrete subclass of vtkExporter that writes VRML 2.0 files. This is based on the VRML 2.0 draft #3 but it should be pretty stable since we aren't using any of the newer features.

To create an instance of class vtkVRMLExporter, simply invoke its constructor as follows

```
obj = vtkVRMLExporter
```

39.202.2 Methods

The class vtkVRMLExporter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVRMLExporter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVRMLExporter = obj.NewInstance ()`
- `vtkVRMLExporter = obj.SafeDownCast (vtkObject o)`
- `obj.SetFileName (string )` - Specify the name of the VRML file to write.
- `string = obj.GetFileName ()` - Specify the name of the VRML file to write.
- `obj.SetSpeed (double )` - Specify the Speed of navigation. Default is 4.
- `double = obj.GetSpeed ()` - Specify the Speed of navigation. Default is 4.

39.203 vtkWindowToImageFilter

39.203.1 Usage

vtkWindowToImageFilter provides methods needed to read the data in a vtkWindow and use it as input to the imaging pipeline. This is useful for saving an image to a file for example. The window can be read as either RGB or RGBA pixels; in addition, the depth buffer can also be read. RGB and RGBA pixels are of type unsigned char, while Z-Buffer data is returned as floats. Use this filter to convert RenderWindows or ImageWindows to an image format.

To create an instance of class vtkWindowToImageFilter, simply invoke its constructor as follows

```
obj = vtkWindowToImageFilter
```

39.203.2 Methods

The class vtkWindowToImageFilter has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWindowToImageFilter class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWindowToImageFilter = obj.NewInstance ()`
- `vtkWindowToImageFilter = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkWindow input)` - Indicates what renderer to get the pixel data from. Initial value is 0.
- `vtkWindow = obj.GetInput ()` - Returns which renderer is being used as the source for the pixel data. Initial value is 0.
- `obj.SetMagnification (int )` - The magnification of the current render window. Initial value is 1.
- `int = obj.GetMagnificationMinValue ()` - The magnification of the current render window. Initial value is 1.
- `int = obj.GetMagnificationMaxValue ()` - The magnification of the current render window. Initial value is 1.
- `int = obj.GetMagnification ()` - The magnification of the current render window. Initial value is 1.
- `obj.ReadFrontBufferOn ()` - Set/Get the flag that determines which buffer to read from. The default is to read from the front buffer.
- `obj.ReadFrontBufferOff ()` - Set/Get the flag that determines which buffer to read from. The default is to read from the front buffer.
- `int = obj.GetReadFrontBuffer ()` - Set/Get the flag that determines which buffer to read from. The default is to read from the front buffer.
- `obj.SetReadFrontBuffer (int )` - Set/Get the flag that determines which buffer to read from. The default is to read from the front buffer.
- `obj.ShouldRerenderOn ()` - Set/get whether to re-render the input window. Initial value is true. (This option makes no difference if Magnification ¿ 1.)

- `obj.ShouldRerenderOff ()` - Set/get whether to re-render the input window. Initial value is true. (This option makes no difference if Magnification ¿ 1.)
- `obj.SetShouldRerender (int )` - Set/get whether to re-render the input window. Initial value is true. (This option makes no difference if Magnification ¿ 1.)
- `int = obj.GetShouldRerender ()` - Set/get whether to re-render the input window. Initial value is true. (This option makes no difference if Magnification ¿ 1.)
- `obj.SetViewport (double , double , double , double )` - Set/get the extents to be used to generate the image. Initial value is 0,0,1,1 (This option does not work if Magnification ¿ 1.)
- `obj.SetViewport (double a[4])` - Set/get the extents to be used to generate the image. Initial value is 0,0,1,1 (This option does not work if Magnification ¿ 1.)
- `double = obj. GetViewport ()` - Set/get the extents to be used to generate the image. Initial value is 0,0,1,1 (This option does not work if Magnification ¿ 1.)
- `obj.SetInputBufferType (int )` - Set/get the window buffer from which data will be read. Choices include VTK_RGB (read the color image from the window), VTK_RGBA (same, but include the alpha channel), and VTK_ZBUFFER (depth buffer, returned as a float array). Initial value is VTK_RGB.
- `int = obj.GetInputBufferType ()` - Set/get the window buffer from which data will be read. Choices include VTK_RGB (read the color image from the window), VTK_RGBA (same, but include the alpha channel), and VTK_ZBUFFER (depth buffer, returned as a float array). Initial value is VTK_RGB.
- `obj.SetInputBufferTypeToRGB ()` - Set/get the window buffer from which data will be read. Choices include VTK_RGB (read the color image from the window), VTK_RGBA (same, but include the alpha channel), and VTK_ZBUFFER (depth buffer, returned as a float array). Initial value is VTK_RGB.
- `obj.SetInputBufferTypeToRGBA ()` - Set/get the window buffer from which data will be read. Choices include VTK_RGB (read the color image from the window), VTK_RGBA (same, but include the alpha channel), and VTK_ZBUFFER (depth buffer, returned as a float array). Initial value is VTK_RGB.
- `obj.SetInputBufferTypeToZBuffer ()` - Set/get the window buffer from which data will be read. Choices include VTK_RGB (read the color image from the window), VTK_RGBA (same, but include the alpha channel), and VTK_ZBUFFER (depth buffer, returned as a float array). Initial value is VTK_RGB.
- `vtkImageData = obj.GetOutput ()` - Get the output data object for a port on this algorithm.

39.204 vtkWorldPointPicker

39.204.1 Usage

vtkWorldPointPicker is used to find the x,y,z world coordinate of a screen x,y,z. This picker cannot pick actors and/or mappers, it simply determines an x-y-z coordinate in world space. (It will always return a x-y-z, even if the selection point is not over a prop/actor.)

To create an instance of class vtkWorldPointPicker, simply invoke its constructor as follows

```
obj = vtkWorldPointPicker
```

39.204.2 Methods

The class vtkWorldPointPicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWorldPointPicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkWorldPointPicker = obj.NewInstance ()`
- `vtkWorldPointPicker = obj.SafeDownCast (vtkObject o)`
- `int = obj.Pick (double selectionX, double selectionY, double selectionZ, vtkRenderer renderer)` - Perform the pick. (This method overload's the superclass.)
- `int = obj.Pick (double selectionPt[3], vtkRenderer renderer)` - Perform the pick. (This method overload's the superclass.)

39.205 vtkXGPUInfoList

39.205.1 Usage

vtkXGPUInfoList implements Probe() method of vtkGPUInfoList through some X server extensions API. NV-CONTROL for Nvidia. ATIFGLEXTENSION for ATI is not supported yet. There is no support for other vendors.

To create an instance of class vtkXGPUInfoList, simply invoke its constructor as follows

```
obj = vtkXGPUInfoList
```

39.205.2 Methods

The class vtkXGPUInfoList has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXGPUInfoList class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXGPUInfoList = obj.NewInstance ()`
- `vtkXGPUInfoList = obj.SafeDownCast (vtkObject o)`
- `obj.Probe ()` - Build the list of vtkInfoGPU if not done yet.

39.206 vtkXOpenGLRenderWindow

39.206.1 Usage

vtkXOpenGLRenderWindow is a concrete implementation of the abstract class vtkRenderWindow. vtkOpenGLRenderer interfaces to the OpenGL graphics library. Application programmers should normally use vtkRenderWindow instead of the OpenGL specific version.

To create an instance of class vtkXOpenGLRenderWindow, simply invoke its constructor as follows

```
obj = vtkXOpenGLRenderWindow
```

39.206.2 Methods

The class vtkXOpenGLRenderWindow has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXOpenGLRenderWindow class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXOpenGLRenderWindow = obj.NewInstance ()`
- `vtkXOpenGLRenderWindow = obj.SafeDownCast (vtkObject o)`
- `obj.Start (void )` - Begin the rendering process.
- `obj.Frame (void )` - End the rendering process and display the image.
- `obj.WindowInitialize (void )` - Initialize the window for rendering.
- `obj.Initialize (void )` - Initialize the rendering window. This will setup all system-specific resources. This method and Finalize() must be symmetric and it should be possible to call them multiple times, even changing WindowId in-between. This is what WindowRemap does.
- `obj.Finalize (void )` - "Deinitialize" the rendering window. This will shutdown all system-specific resources. After having called this, it should be possible to destroy a window that was used for a SetWindowId() call without any ill effects.
- `obj.SetFullScreen (int )` - Change the window to fill the entire screen.
- `obj.WindowRemap (void )` - Resize the window.
- `obj.PrefFullScreen (void )` - Set the preferred window size to full screen.
- `obj.SetSize (int , int )` - Specify the size of the rendering window in pixels.
- `obj.SetSize (int a[2])` - Specify the size of the rendering window in pixels.
- `int = obj.GetDesiredDepth ()` - Get the X properties of an ideal rendering window.
- `obj.SetStereoCapableWindow (int capable)` - Prescribe that the window be created in a stereo-capable mode. This method must be called before the window is realized. This method overrides the superclass method since this class can actually check whether the window has been realized yet.
- `obj.MakeCurrent ()` - Make this window the current OpenGL context.
- `bool = obj.IsCurrent ()` - Tells if this window is the current OpenGL context for the calling thread.
- `obj.SetForceMakeCurrent ()` - If called, allow MakeCurrent() to skip cache-check when called. MakeCurrent() reverts to original behavior of cache-checking on the next render.
- `string = obj.ReportCapabilities ()` - Get report of capabilities for the render window
- `int = obj.SupportsOpenGL ()` - Does this render window support OpenGL? 0-false, 1-true
- `int = obj.IsDirect ()` - Is this render window using hardware acceleration? 0-false, 1-true
- `obj.SetWindowName (string )`
- `obj.SetPosition (int , int )` - Move the window to a new position on the display.
- `obj.SetPosition (int a[2])` - Move the window to a new position on the display.

- `obj.HideCursor ()` - Hide or Show the mouse cursor, it is nice to be able to hide the default cursor if you want VTK to display a 3D cursor instead.
- `obj.ShowCursor ()` - Hide or Show the mouse cursor, it is nice to be able to hide the default cursor if you want VTK to display a 3D cursor instead.
- `obj.SetCurrentCursor (int )` - Change the shape of the cursor
- `int = obj.GetEventPending ()` - Check to see if a mouse button has been pressed. All other events are ignored by this method. This is a useful check to abort a long render.
- `obj.SetWindowInfo (string info)` - Set this RenderWindow's X window id to a pre-existing window.
- `obj.SetNextWindowInfo (string info)` - Set the window info that will be used after WindowRemap()
- `obj.SetParentInfo (string info)` - Sets the X window id of the window that WILL BE created.
- `obj.Render ()` - This computes the size of the render window before calling the supper classes render
- `obj.SetOffScreenRendering (int i)` - Render without displaying the window.

39.207 vtkXRenderWindowInteractor

39.207.1 Usage

vtkXRenderWindowInteractor is a convenience object that provides event bindings to common graphics functions. For example, camera and actor functions such as zoom-in/zoom-out, azimuth, roll, and pan. IT is one of the window system specific subclasses of vtkRenderWindowInteractor. Please see vtkRenderWindowInteractor documentation for event bindings.

To create an instance of class vtkXRenderWindowInteractor, simply invoke its constructor as follows

```
obj = vtkXRenderWindowInteractor
```

39.207.2 Methods

The class vtkXRenderWindowInteractor has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXRenderWindowInteractor class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXRenderWindowInteractor = obj.NewInstance ()`
- `vtkXRenderWindowInteractor = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize ()` - Initializes the event handlers without an XtAppContext. This is good for when you don't have a user interface, but you still want to have mouse interaction.
- `obj.TerminateApp ()` - Break the event loop on 'q','e' keypress. Want more ???
- `int = obj.GetBreakLoopFlag ()` - The BreakLoopFlag is checked in the Start() method. Setting it to anything other than zero will cause the interactor loop to terminate and return to the calling function.
- `obj.SetBreakLoopFlag (int )` - The BreakLoopFlag is checked in the Start() method. Setting it to anything other than zero will cause the interactor loop to terminate and return to the calling function.

- `obj.BreakLoopFlagOff ()` - The BreakLoopFlag is checked in the Start() method. Setting it to anything other than zero will cause the interactor loop to terminate and return to the calling function.
- `obj.BreakLoopFlagOn ()` - The BreakLoopFlag is checked in the Start() method. Setting it to anything other than zero will cause the interactor loop to terminate and return to the calling function.
- `obj.Enable ()` - Enable/Disable interactions. By default interactors are enabled when initialized. Initialize() must be called prior to enabling/disabling interaction. These methods are used when a window/widget is being shared by multiple renderers and interactors. This allows a "modal" display where one interactor is active when its data is to be displayed and all other interactors associated with the widget are disabled when their data is not displayed.
- `obj.Disable ()` - Enable/Disable interactions. By default interactors are enabled when initialized. Initialize() must be called prior to enabling/disabling interaction. These methods are used when a window/widget is being shared by multiple renderers and interactors. This allows a "modal" display where one interactor is active when its data is to be displayed and all other interactors associated with the widget are disabled when their data is not displayed.
- `obj.Start ()` - This will start up the X event loop and never return. If you call this method it will loop processing X events until the application is exited.
- `obj.UpdateSize (int , int )` - Update the Size data member and set the associated RenderWindow's size.
- `obj.GetMousePosition (int x, int y)` - Re-defines virtual function to get mouse position by querying X-server.

Chapter 40

Visualization Toolkit View Classes

40.1 vtkConvertSelectionDomain

40.1.1 Usage

vtkConvertSelectionDomain converts a selection from one domain to another using known domain mappings. The domain mappings are described by a vtkMultiBlockDataSet containing one or more vtkTables.

The first input port is for the input selection (or collection of annotations in a vtkAnnotationLayers object), while the second port is for the multi-block of mappings, and the third port is for the data that is being selected on.

If the second or third port is not set, this filter will pass the selection/annotation to the output unchanged.

The second output is the selection associated with the "current annotation" normally representing the current interactive selection.

To create an instance of class vtkConvertSelectionDomain, simply invoke its constructor as follows

```
obj = vtkConvertSelectionDomain
```

40.1.2 Methods

The class vtkConvertSelectionDomain has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkConvertSelectionDomain class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkConvertSelectionDomain = obj.NewInstance ()`
- `vtkConvertSelectionDomain = obj.SafeDownCast (vtkObject o)`

40.2 vtkDataRepresentation

40.2.1 Usage

vtkDataRepresentation the superclass for representations of data objects. This class itself may be instantiated and used as a representation that simply holds a connection to a pipeline.

If there are multiple representations present in a view, you should use a subclass of vtkDataRepresentation. The representation is responsible for taking the input pipeline connection and converting it to an object usable by a view. In the most common case, the representation will contain the pipeline necessary to convert a data object into an actor or set of actors.

The representation has a concept of a selection. If the user performs a selection operation on the view, the view forwards this on to its representations. The representation is responsible for displaying that selection in an appropriate way.

Representation selections may also be linked. The representation shares the selection by converting it into a view-independent format, then setting the selection on its vtkAnnotationLink. Other representations sharing the same selection link instance will get the same selection from the selection link when the view is updated. The application is responsible for linking representations as appropriate by setting the same vtkAnnotationLink on each linked representation.

To create an instance of class vtkDataRepresentation, simply invoke its constructor as follows

```
obj = vtkDataRepresentation
```

40.2.2 Methods

The class vtkDataRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDataRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkDataRepresentation = obj.NewInstance ()`
- `vtkDataRepresentation = obj.SafeDownCast (vtkObject o)`
- `vtkAlgorithmOutput = obj.GetInputConnection (int port, int index)` - The annotation link for this representation. To link annotations, set the same vtkAnnotationLink object in multiple representations.
- `vtkAnnotationLink = obj.GetAnnotationLink ()` - The annotation link for this representation. To link annotations, set the same vtkAnnotationLink object in multiple representations.
- `obj.SetAnnotationLink (vtkAnnotationLink link)` - The annotation link for this representation. To link annotations, set the same vtkAnnotationLink object in multiple representations.
- `obj.ApplyViewTheme (vtkViewTheme )` - The view calls this method when a selection occurs. The representation takes this selection and converts it into a selection on its data by calling ConvertSelection, then calls UpdateSelection with the converted selection. Subclasses should not overrride this method, but should instead override ConvertSelection. The optional third argument specifies whether the selection should be added to the previous selection on this representation.
- `obj.Select (vtkView view, vtkSelection selection)` - The view calls this method when a selection occurs. The representation takes this selection and converts it into a selection on its data by calling ConvertSelection, then calls UpdateSelection with the converted selection. Subclasses should not overrride this method, but should instead override ConvertSelection. The optional third argument specifies whether the selection should be added to the previous selection on this representation.
- `obj.Select (vtkView view, vtkSelection selection, bool extend)` - The view calls this method when a selection occurs. The representation takes this selection and converts it into a selection on its data by calling ConvertSelection, then calls UpdateSelection with the converted selection. Subclasses should not overrride this method, but should instead override ConvertSelection. The optional third argument specifies whether the selection should be added to the previous selection on this representation.
- `obj.SetSelectable (bool )` - Whether this representation is able to handle a selection. Default is true.

- `bool = obj.GetSelectable ()` - Whether this representation is able to handle a selection. Default is true.
- `obj.SelectableOn ()` - Whether this representation is able to handle a selection. Default is true.
- `obj.SelectableOff ()` - Whether this representation is able to handle a selection. Default is true.
- `obj.UpdateSelection (vtkSelection selection)` - Updates the selection in the selection link and fires a selection change event. Subclasses should not overrride this method, but should instead override ConvertSelection. The optional second argument specifies whether the selection should be added to the previous selection on this representation.
- `obj.UpdateSelection (vtkSelection selection, bool extend)` - Updates the selection in the selection link and fires a selection change event. Subclasses should not overrride this method, but should instead override ConvertSelection. The optional second argument specifies whether the selection should be added to the previous selection on this representation.
- `vtkAlgorithmOutput = obj.GetInternalAnnotationOutputPort ()` - The output port that contains the annotations whose selections are localized for a particular input data object. This should be used when connecting the internal pipelines.
- `vtkAlgorithmOutput = obj.GetInternalAnnotationOutputPort (int port)` - The output port that contains the annotations whose selections are localized for a particular input data object. This should be used when connecting the internal pipelines.
- `vtkAlgorithmOutput = obj.GetInternalAnnotationOutputPort (int port, int conn)` - The output port that contains the annotations whose selections are localized for a particular input data object. This should be used when connecting the internal pipelines.
- `vtkAlgorithmOutput = obj.GetInternalSelectionOutputPort ()` - The output port that contains the selection associated with the current annotation (normally the interactive selection). This should be used when connecting the internal pipelines.
- `vtkAlgorithmOutput = obj.GetInternalSelectionOutputPort (int port)` - The output port that contains the selection associated with the current annotation (normally the interactive selection). This should be used when connecting the internal pipelines.
- `vtkAlgorithmOutput = obj.GetInternalSelectionOutputPort (int port, int conn)` - The output port that contains the selection associated with the current annotation (normally the interactive selection). This should be used when connecting the internal pipelines.
- `vtkAlgorithmOutput = obj.GetInternalOutputPort ()` - Retrieves an output port for the input data object at the specified port and connection index. This may be connected to the representation's internal pipeline.
- `vtkAlgorithmOutput = obj.GetInternalOutputPort (int port)` - Retrieves an output port for the input data object at the specified port and connection index. This may be connected to the representation's internal pipeline.
- `vtkAlgorithmOutput = obj.GetInternalOutputPort (int port, int conn)` - Retrieves an output port for the input data object at the specified port and connection index. This may be connected to the representation's internal pipeline.
- `obj.SetSelectionType (int )` - Set the selection type produced by this view. This should be one of the content type constants defined in vtkSelectionNode.h. Common values are vtkSelectionNode::INDICES vtkSelectionNode::PEDIGREEIDS vtkSelectionNode::VALUES
- `int = obj.GetSelectionType ()` - Set the selection type produced by this view. This should be one of the content type constants defined in vtkSelectionNode.h. Common values are vtkSelectionNode::INDICES vtkSelectionNode::PEDIGREEIDS vtkSelectionNode::VALUES

- `obj.SetSelectionArrayNames (vtkStringArray names)` - If a VALUES selection, the arrays used to produce a selection.
- `vtkStringArray = obj.GetSelectionArrayNames ()` - If a VALUES selection, the arrays used to produce a selection.
- `obj.SetSelectionArrayName (string name)` - If a VALUES selection, the array used to produce a selection.
- `string = obj.GetSelectionArrayName ()` - If a VALUES selection, the array used to produce a selection.

40.3 vtkEmptyRepresentation

40.3.1 Usage

To create an instance of class vtkEmptyRepresentation, simply invoke its constructor as follows

```
obj = vtkEmptyRepresentation
```

40.3.2 Methods

The class vtkEmptyRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEmptyRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEmptyRepresentation = obj.NewInstance ()`
- `vtkEmptyRepresentation = obj.SafeDownCast (vtkObject o)`
- `vtkAlgorithmOutput = obj.GetInternalAnnotationOutputPort ()` - Since this representation has no inputs, override superclass implementation with one that ignores "port" and "conn" and still allows it to have an annotation output.
- `vtkAlgorithmOutput = obj.GetInternalAnnotationOutputPort (int port)` - Since this representation has no inputs, override superclass implementation with one that ignores "port" and "conn" and still allows it to have an annotation output.
- `vtkAlgorithmOutput = obj.GetInternalAnnotationOutputPort (int port, int conn)` - Since this representation has no inputs, override superclass implementation with one that ignores "port" and "conn" and still allows it to have an annotation output.

40.4 vtkGraphLayoutView

40.4.1 Usage

vtkGraphLayoutView performs graph layout and displays a vtkGraph. You may color and label the vertices and edges using fields in the graph. If coordinates are already assigned to the graph vertices in your graph, set the layout strategy to PassThrough in this view. The default layout is Fast2D which is fast but not that good, for better layout set the layout to Simple2D or ForceDirected. There are also tree and circle layout strategies. :)

.SEE ALSO vtkFast2DLayoutStrategy vtkSimple2DLayoutStrategy vtkForceDirectedLayoutStrategy

.SECTION Thanks Thanks a bunch to the holographic unfolding pattern.

To create an instance of class vtkGraphLayoutView, simply invoke its constructor as follows

```
obj = vtkGraphLayoutView
```

40.4.2 Methods

The class vtkGraphLayoutView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGraphLayoutView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGraphLayoutView = obj.NewInstance ()`
- `vtkGraphLayoutView = obj.SafeDownCast (vtkObject o)`
- `obj.SetVertexLabelArrayName (string name)` - The array to use for vertex labeling. Default is "label".
- `string = obj.GetVertexLabelArrayName ()` - The array to use for vertex labeling. Default is "label".
- `obj.SetEdgeLabelArrayName (string name)` - The array to use for edge labeling. Default is "label".
- `string = obj.GetEdgeLabelArrayName ()` - The array to use for edge labeling. Default is "label".
- `obj.SetVertexLabelVisibility (bool vis)` - Whether to show vertex labels. Default is off.
- `bool = obj.GetVertexLabelVisibility ()` - Whether to show vertex labels. Default is off.
- `obj.VertexLabelVisibilityOn ()` - Whether to show vertex labels. Default is off.
- `obj.VertexLabelVisibilityOff ()` - Whether to show vertex labels. Default is off.
- `obj.SetHideVertexLabelsOnInteraction (bool vis)` - Whether to hide vertex labels during mouse interactions. Default is off.
- `bool = obj.GetHideVertexLabelsOnInteraction ()` - Whether to hide vertex labels during mouse interactions. Default is off.
- `obj.HideVertexLabelsOnInteractionOn ()` - Whether to hide vertex labels during mouse interactions. Default is off.
- `obj.HideVertexLabelsOnInteractionOff ()` - Whether to hide vertex labels during mouse interactions. Default is off.
- `obj.SetEdgeVisibility (bool vis)` - Whether to show the edges at all. Default is on
- `bool = obj.GetEdgeVisibility ()` - Whether to show the edges at all. Default is on
- `obj.EdgeVisibilityOn ()` - Whether to show the edges at all. Default is on
- `obj.EdgeVisibilityOff ()` - Whether to show the edges at all. Default is on
- `obj.SetEdgeLabelVisibility (bool vis)` - Whether to show edge labels. Default is off.
- `bool = obj.GetEdgeLabelVisibility ()` - Whether to show edge labels. Default is off.
- `obj.EdgeLabelVisibilityOn ()` - Whether to show edge labels. Default is off.
- `obj.EdgeLabelVisibilityOff ()` - Whether to show edge labels. Default is off.

- `obj.SetHideEdgeLabelsOnInteraction (bool vis)` - Whether to hide edge labels during mouse interactions. Default is off.
- `bool = obj.GetHideEdgeLabelsOnInteraction ()` - Whether to hide edge labels during mouse interactions. Default is off.
- `obj.HideEdgeLabelsOnInteractionOn ()` - Whether to hide edge labels during mouse interactions. Default is off.
- `obj.HideEdgeLabelsOnInteractionOff ()` - Whether to hide edge labels during mouse interactions. Default is off.
- `obj.SetVertexColorArrayName (string name)` - The array to use for coloring vertices. Default is "color".
- `string = obj.GetVertexColorArrayName ()` - The array to use for coloring vertices. Default is "color".
- `obj.SetColorVertices (bool vis)` - Whether to color vertices. Default is off.
- `bool = obj.GetColorVertices ()` - Whether to color vertices. Default is off.
- `obj.ColorVerticesOn ()` - Whether to color vertices. Default is off.
- `obj.ColorVerticesOff ()` - Whether to color vertices. Default is off.
- `obj.SetEdgeColorArrayName (string name)` - The array to use for coloring edges. Default is "color".
- `string = obj.GetEdgeColorArrayName ()` - The array to use for coloring edges. Default is "color".
- `obj.SetColorEdges (bool vis)` - Whether to color edges. Default is off.
- `bool = obj.GetColorEdges ()` - Whether to color edges. Default is off.
- `obj.ColorEdgesOn ()` - Whether to color edges. Default is off.
- `obj.ColorEdgesOff ()` - Whether to color edges. Default is off.
- `obj.SetEnabledEdgesArrayName (string name)` - The array to use for coloring edges.
- `string = obj.GetEnabledEdgesArrayName ()` - The array to use for coloring edges.
- `obj.SetEnableEdgesByArray (bool vis)` - Whether to color edges. Default is off.
- `int = obj.GetEnableEdgesByArray ()` - Whether to color edges. Default is off.
- `obj.SetEnabledVerticesArrayName (string name)` - The array to use for coloring vertices.
- `string = obj.GetEnabledVerticesArrayName ()` - The array to use for coloring vertices.
- `obj.SetEnableVerticesByArray (bool vis)` - Whether to color vertices. Default is off.
- `int = obj.GetEnableVerticesByArray ()` - Whether to color vertices. Default is off.
- `obj.SetScalingArrayName (string name)` - The array used for scaling (if ScaledGlyphs is ON)
- `string = obj.GetScalingArrayName ()` - The array used for scaling (if ScaledGlyphs is ON)
- `obj.SetScaledGlyphs (bool arg)` - Whether to use scaled glyphs or not. Default is off.
- `bool = obj.GetScaledGlyphs ()` - Whether to use scaled glyphs or not. Default is off.
- `obj.ScaledGlyphsOn ()` - Whether to use scaled glyphs or not. Default is off.

- `obj.ScaledGlyphsOff ()` - Whether to use scaled glyphs or not. Default is off.

- `obj.SetLayoutStrategy (string name)` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToRandom ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToForceDirected ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToSimple2D ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToClustering2D ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToCommunity2D ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToFast2D ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToPassThrough ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToCircular ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToTree ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToCosmicTree ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToCone ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `obj.SetLayoutStrategyToSpanTree ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A

layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `string = obj.GetLayoutStrategyName ()` - The layout strategy to use when performing the graph layout. The possible strings are: - "Random" Randomly places vertices in a box. - "Force Directed" A layout in 3D or 2D simulating forces on edges. - "Simple 2D" A simple 2D force directed layout. - "Clustering 2D" A 2D force directed layout that's just like simple 2D but uses some techniques to cluster better. - "Community 2D" A linear-time 2D layout that's just like Fast 2D but looks for and uses a community array to 'accentuate' clusters. - "Fast 2D" A linear-time 2D layout. - "Pass Through" Use locations assigned to the input. - "Circular" Places vertices uniformly on a circle. - "Cone" Cone tree layout. - "Span Tree" Span Tree Layout. Default is "Simple 2D".

- `vtkGraphLayoutStrategy = obj.GetLayoutStrategy ()` - The layout strategy to use when performing the graph layout. This signature allows an application to create a layout object directly and simply set the pointer through this method.

- `obj.SetLayoutStrategy (vtkGraphLayoutStrategy s)` - The layout strategy to use when performing the graph layout. This signature allows an application to create a layout object directly and simply set the pointer through this method.

- `obj.SetEdgeLayoutStrategy (string name)` - The layout strategy to use when performing the edge layout. The possible strings are: "Arc Parallel" - Arc parallel edges and self loops. "Pass Through" - Use edge routes assigned to the input. Default is "Arc Parallel".

- `obj.SetEdgeLayoutStrategyToArcParallel ()` - The layout strategy to use when performing the edge layout. The possible strings are: "Arc Parallel" - Arc parallel edges and self loops. "Pass Through" - Use edge routes assigned to the input. Default is "Arc Parallel".

- `obj.SetEdgeLayoutStrategyToPassThrough ()` - The layout strategy to use when performing the edge layout. The possible strings are: "Arc Parallel" - Arc parallel edges and self loops. "Pass Through" - Use edge routes assigned to the input. Default is "Arc Parallel".

- `string = obj.GetEdgeLayoutStrategyName ()` - The layout strategy to use when performing the edge layout. The possible strings are: "Arc Parallel" - Arc parallel edges and self loops. "Pass Through" - Use edge routes assigned to the input. Default is "Arc Parallel".

- `vtkEdgeLayoutStrategy = obj.GetEdgeLayoutStrategy ()` - The layout strategy to use when performing the edge layout. This signature allows an application to create a layout object directly and simply set the pointer through this method.

- `obj.SetEdgeLayoutStrategy (vtkEdgeLayoutStrategy s)` - The layout strategy to use when performing the edge layout. This signature allows an application to create a layout object directly and simply set the pointer through this method.

- `obj.AddIconType (string type, int index)` - Associate the icon at index "index" in the vtkTexture to all vertices containing "type" as a value in the vertex attribute array specified by IconArrayName.

- `obj.ClearIconTypes ()` - Clear all icon mappings.

- `obj.SetIconAlignment (int alignment)` - Specify where the icons should be placed in relation to the vertex. See vtkIconGlyphFilter.h for possible values.

- `obj.SetIconVisibility (bool b)` - Whether icons are visible (default off).

- `bool = obj.GetIconVisibility ()` - Whether icons are visible (default off).
- `obj.IconVisibilityOn ()` - Whether icons are visible (default off).
- `obj.IconVisibilityOff ()` - Whether icons are visible (default off).
- `obj.SetIconArrayName (string name)` - The array used for assigning icons
- `string = obj.GetIconArrayName ()` - The array used for assigning icons
- `obj.SetGlyphType (int type)` - The type of glyph to use for the vertices
- `int = obj.GetGlyphType ()` - The type of glyph to use for the vertices
- `obj.SetVertexLabelFontSize (int size)` - The size of the font used for vertex labeling
- `int = obj.GetVertexLabelFontSize ()` - The size of the font used for vertex labeling
- `obj.SetEdgeLabelFontSize (int size)` - The size of the font used for edge labeling
- `int = obj.GetEdgeLabelFontSize ()` - The size of the font used for edge labeling
- `obj.SetEdgeScalarBarVisibility (bool vis)` - Whether the scalar bar for edges is visible. Default is off.
- `bool = obj.GetEdgeScalarBarVisibility ()` - Whether the scalar bar for edges is visible. Default is off.
- `obj.SetVertexScalarBarVisibility (bool vis)` - Whether the scalar bar for vertices is visible. Default is off.
- `bool = obj.GetVertexScalarBarVisibility ()` - Whether the scalar bar for vertices is visible. Default is off.
- `obj.ZoomToSelection ()` - Reset the camera based on the bounds of the selected region.
- `int = obj.IsLayoutComplete ()` - Is the graph layout complete? This method is useful for when the strategy is iterative and the application wants to show the iterative progress of the graph layout See Also: UpdateLayout();
- `obj.UpdateLayout ()` - This method is useful for when the strategy is iterative and the application wants to show the iterative progress of the graph layout. The application would have something like while(!IsLayoutComplete()) UpdateLayout(); See Also: IsLayoutComplete();

40.5 vtkHierarchicalGraphPipeline

40.5.1 Usage

vtkHierarchicalGraphPipeline renders bundled edges that are meant to be viewed as an overlay on a tree. This class is not for general use, but is used in the internals of vtkRenderedHierarchyRepresentation and vtkRenderedTreeAreaRepresentation.

To create an instance of class vtkHierarchicalGraphPipeline, simply invoke its constructor as follows

```
obj = vtkHierarchicalGraphPipeline
```

40.5.2 Methods

The class vtkHierarchicalGraphPipeline has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalGraphPipeline class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHierarchicalGraphPipeline = obj.NewInstance ()`
- `vtkHierarchicalGraphPipeline = obj.SafeDownCast (vtkObject o)`
- `vtkActor = obj.GetActor ()` - The actor associated with the hierarchical graph.
- `vtkActor2D = obj.GetLabelActor ()` - The actor associated with the hierarchical graph.
- `obj.SetBundlingStrength (double strength)` - The bundling strength for the bundled edges.
- `double = obj.GetBundlingStrength ()` - The bundling strength for the bundled edges.
- `obj.SetLabelArrayName (string name)` - The edge label array name.
- `string = obj.GetLabelArrayName ()` - The edge label array name.
- `obj.SetLabelVisibility (bool vis)` - The edge label visibility.
- `bool = obj.GetLabelVisibility ()` - The edge label visibility.
- `obj.LabelVisibilityOn ()` - The edge label visibility.
- `obj.LabelVisibilityOff ()` - The edge label visibility.
- `obj.SetLabelTextProperty (vtkTextProperty prop)` - The edge label text property.
- `vtkTextProperty = obj.GetLabelTextProperty ()` - The edge label text property.
- `obj.SetColorArrayName (string name)` - The edge color array.
- `string = obj.GetColorArrayName ()` - The edge color array.
- `obj.SetColorEdgesByArray (bool vis)` - Whether to color the edges by an array.
- `bool = obj.GetColorEdgesByArray ()` - Whether to color the edges by an array.
- `obj.ColorEdgesByArrayOn ()` - Whether to color the edges by an array.
- `obj.ColorEdgesByArrayOff ()` - Whether to color the edges by an array.
- `obj.SetVisibility (bool vis)` - The visibility of this graph.
- `bool = obj.GetVisibility ()` - The visibility of this graph.
- `obj.VisibilityOn ()` - The visibility of this graph.
- `obj.VisibilityOff ()` - The visibility of this graph.
- `vtkSelection = obj.ConvertSelection (vtkDataRepresentation rep, vtkSelection sel)` - Returns a new selection relevant to this graph based on an input selection and the view that this graph is contained in.

- `obj.PrepareInputConnections (vtkAlgorithmOutput graphConn, vtkAlgorithmOutput treeConn, vtkAlgorith` - Sets the input connections for this graph. graphConn is the input graph connection. treeConn is the input tree connection. annConn is the annotation link connection.
- `obj.ApplyViewTheme (vtkViewTheme theme)` - Applies the view theme to this graph.
- `obj.SetHoverArrayName (string )` - The array to use while hovering over an edge.
- `string = obj.GetHoverArrayName ()` - The array to use while hovering over an edge.
- `obj.SetSplineType (int type)` - The spline mode to use in vtkSplineGraphEdges. vtkSplineGraphEdges::CUSTOM uses a vtkCardinalSpline. vtkSplineGraphEdges::BSPLINE uses a b-spline. The default is CUSTOM.
- `int = obj.GetSplineType ()` - The spline mode to use in vtkSplineGraphEdges. vtkSplineGraphEdges::CUSTOM uses a vtkCardinalSpline. vtkSplineGraphEdges::BSPLINE uses a b-spline. The default is CUSTOM.
- `obj.RegisterProgress (vtkRenderView view)` - Register progress with a view.

40.6 vtkHierarchicalGraphView

40.6.1 Usage

Takes a graph and a hierarchy (currently a tree) and lays out the graph vertices based on their categorization within the hierarchy.

.SEE ALSO vtkGraphLayoutView

.SECTION Thanks Thanks to the turtle with jets for feet, without you this class wouldn't have been possible.

To create an instance of class vtkHierarchicalGraphView, simply invoke its constructor as follows

```
obj = vtkHierarchicalGraphView
```

40.6.2 Methods

The class vtkHierarchicalGraphView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHierarchicalGraphView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHierarchicalGraphView = obj.NewInstance ()`
- `vtkHierarchicalGraphView = obj.SafeDownCast (vtkObject o)`
- `vtkDataRepresentation = obj.SetHierarchyFromInputConnection (vtkAlgorithmOutput conn)` - Set the tree and graph representations to the appropriate input ports.
- `vtkDataRepresentation = obj.SetHierarchyFromInput (vtkDataObject input)` - Set the tree and graph representations to the appropriate input ports.
- `vtkDataRepresentation = obj.SetGraphFromInputConnection (vtkAlgorithmOutput conn)` - Set the tree and graph representations to the appropriate input ports.
- `vtkDataRepresentation = obj.SetGraphFromInput (vtkDataObject input)` - Set the tree and graph representations to the appropriate input ports.
- `obj.SetGraphEdgeLabelArrayName (string name)` - The array to use for edge labeling. Default is "label".

- `string = obj.GetGraphEdgeLabelArrayName ()` - The array to use for edge labeling. Default is "label".
- `obj.SetGraphEdgeLabelVisibility (bool vis)` - Whether to show edge labels. Default is off.
- `bool = obj.GetGraphEdgeLabelVisibility ()` - Whether to show edge labels. Default is off.
- `obj.GraphEdgeLabelVisibilityOn ()` - Whether to show edge labels. Default is off.
- `obj.GraphEdgeLabelVisibilityOff ()` - Whether to show edge labels. Default is off.
- `obj.SetGraphEdgeColorArrayName (string name)` - The array to use for coloring edges. Default is "color".
- `string = obj.GetGraphEdgeColorArrayName ()` - The array to use for coloring edges. Default is "color".
- `obj.SetGraphEdgeColorToSplineFraction ()` - Set the color to be the spline fraction
- `obj.SetColorGraphEdgesByArray (bool vis)` - Whether to color edges. Default is off.
- `bool = obj.GetColorGraphEdgesByArray ()` - Whether to color edges. Default is off.
- `obj.ColorGraphEdgesByArrayOn ()` - Whether to color edges. Default is off.
- `obj.ColorGraphEdgesByArrayOff ()` - Whether to color edges. Default is off.
- `obj.SetBundlingStrength (double strength)` - Set the bundling strength.
- `double = obj.GetBundlingStrength ()` - Set the bundling strength.
- `obj.SetGraphVisibility (bool b)` - Whether the graph edges are visible (default off).
- `bool = obj.GetGraphVisibility ()` - Whether the graph edges are visible (default off).
- `obj.GraphVisibilityOn ()` - Whether the graph edges are visible (default off).
- `obj.GraphVisibilityOff ()` - Whether the graph edges are visible (default off).
- `obj.SetGraphEdgeLabelFontSize (int size)` - The size of the font used for edge labeling
- `int = obj.GetGraphEdgeLabelFontSize ()` - The size of the font used for edge labeling

40.7 vtkIcicleView

40.7.1 Usage

vtkIcicleView shows a vtkTree in horizontal layers where each vertex in the tree is represented by a bar. Child sectors are below (or above) parent sectors, and may be colored and sized by various parameters.

To create an instance of class vtkIcicleView, simply invoke its constructor as follows

```
obj = vtkIcicleView
```

40.7.2 Methods

The class vtkIcicleView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkIcicleView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkIcicleView = obj.NewInstance ()`
- `vtkIcicleView = obj.SafeDownCast (vtkObject o)`
- `obj.SetTopToBottom (bool value)` - Sets whether the stacks go from top to bottom or bottom to top.
- `bool = obj.GetTopToBottom ()` - Sets whether the stacks go from top to bottom or bottom to top.
- `obj.TopToBottomOn ()` - Sets whether the stacks go from top to bottom or bottom to top.
- `obj.TopToBottomOff ()` - Sets whether the stacks go from top to bottom or bottom to top.
- `obj.SetRootWidth (double width)` - Set the width of the root node
- `double = obj.GetRootWidth ()` - Set the width of the root node
- `obj.SetLayerThickness (double thickness)` - Set the thickness of each layer
- `double = obj.GetLayerThickness ()` - Set the thickness of each layer
- `obj.SetUseGradientColoring (bool value)` - Turn on/off gradient coloring.
- `bool = obj.GetUseGradientColoring ()` - Turn on/off gradient coloring.
- `obj.UseGradientColoringOn ()` - Turn on/off gradient coloring.
- `obj.UseGradientColoringOff ()` - Turn on/off gradient coloring.

40.8 vtkInteractorStyleAreaSelectHover

40.8.1 Usage

The vtkInteractorStyleAreaSelectHover specifically works with pipelines that create a hierarchical tree. Such pipelines will have a vtkAreaLayout filter which must be passed to this interactor style for it to function correctly. This interactor style allows only 2D panning and zooming, rubber band selection and provides a balloon containing the name of the vertex hovered over.

To create an instance of class vtkInteractorStyleAreaSelectHover, simply invoke its constructor as follows

```
obj = vtkInteractorStyleAreaSelectHover
```

40.8.2 Methods

The class vtkInteractorStyleAreaSelectHover has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleAreaSelectHover class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkInteractorStyleAreaSelectHover = obj.NewInstance ()`
- `vtkInteractorStyleAreaSelectHover = obj.SafeDownCast (vtkObject o)`
- `obj.SetLayout (vtkAreaLayout layout)` - Must be set to the vtkAreaLayout used to compute the bounds of each vertex.
- `vtkAreaLayout = obj.GetLayout ()` - Must be set to the vtkAreaLayout used to compute the bounds of each vertex.
- `obj.SetLabelField (string )` - The name of the field to use when displaying text in the hover balloon.
- `string = obj.GetLabelField ()` - The name of the field to use when displaying text in the hover balloon.
- `obj.SetUseRectangularCoordinates (bool )` - Determine whether or not to use rectangular coordinates instead of polar coordinates.
- `bool = obj.GetUseRectangularCoordinates ()` - Determine whether or not to use rectangular coordinates instead of polar coordinates.
- `obj.UseRectangularCoordinatesOn ()` - Determine whether or not to use rectangular coordinates instead of polar coordinates.
- `obj.UseRectangularCoordinatesOff ()` - Determine whether or not to use rectangular coordinates instead of polar coordinates.
- `obj.OnMouseMove ()` - Overridden from vtkInteractorStyleImage to provide the desired interaction behavior.
- `obj.SetInteractor (vtkRenderWindowInteractor rwi)` - Set the interactor that this interactor style works with.
- `obj.SetHighLightColor (double r, double g, double b)` - Set the color used to highlight the hovered vertex.
- `obj.SetHighLightWidth (double lw)` - The width of the line around the hovered vertex.
- `double = obj.GetHighLightWidth ()` - The width of the line around the hovered vertex.
- `vtkIdType = obj.GetIdAtPos (int x, int y)` - Obtain the tree vertex id at the position specified.

40.9 vtkInteractorStyleTreeMapHover

40.9.1 Usage

The vtkInteractorStyleTreeMapHover specifically works with pipelines that create a tree map. Such pipelines will have a vtkTreeMapLayout filter and a vtkTreeMapToPolyData filter, both of which must be passed to this interactor style for it to function correctly. This interactor style allows only 2D panning and zooming, and additionally provides a balloon containing the name of the vertex hovered over, and allows the user to highlight a vertex by clicking on it.

To create an instance of class vtkInteractorStyleTreeMapHover, simply invoke its constructor as follows

```
obj = vtkInteractorStyleTreeMapHover
```

40.9.2 Methods

The class vtkInteractorStyleTreeMapHover has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkInteractorStyleTreeMapHover class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkInteractorStyleTreeMapHover = obj.NewInstance ()`
- `vtkInteractorStyleTreeMapHover = obj.SafeDownCast (vtkObject o)`
- `obj.SetLayout (vtkTreeMapLayout layout)` - Must be set to the vtkTreeMapLayout used to compute the bounds of each vertex for the tree map.
- `vtkTreeMapLayout = obj.GetLayout ()` - Must be set to the vtkTreeMapLayout used to compute the bounds of each vertex for the tree map.
- `obj.SetTreeMapToPolyData (vtkTreeMapToPolyData filter)` - Must be set to the vtkTreeMapToPolyData used to convert the tree map into polydata.
- `vtkTreeMapToPolyData = obj.GetTreeMapToPolyData ()` - Must be set to the vtkTreeMapToPolyData used to convert the tree map into polydata.
- `obj.SetLabelField (string )` - The name of the field to use when displaying text in the hover balloon.
- `string = obj.GetLabelField ()` - The name of the field to use when displaying text in the hover balloon.
- `obj.OnMouseMove ()` - Overridden from vtkInteractorStyleImage to provide the desired interaction behavior.
- `obj.OnLeftButtonUp ()` - Overridden from vtkInteractorStyleImage to provide the desired interaction behavior.
- `obj.HighLightItem (vtkIdType id)` - Highlights a specific vertex.
- `obj.HighLightCurrentSelectedItem ()` - Highlights a specific vertex.
- `obj.SetInteractor (vtkRenderWindowInteractor rwi)`
- `obj.SetHighLightColor (double r, double g, double b)` - Set the color used to highlight the hovered vertex.
- `obj.SetSelectionLightColor (double r, double g, double b)` - Set the color used to highlight the selected vertex.
- `obj.SetHighLightWidth (double lw)` - The width of the line around the hovered vertex.
- `double = obj.GetHighLightWidth ()` - The width of the line around the hovered vertex.
- `obj.SetSelectionWidth (double lw)` - The width of the line around the selected vertex.
- `double = obj.GetSelectionWidth ()` - The width of the line around the selected vertex.

40.10 vtkParallelCoordinatesHistogramRepresentation

40.10.1 Usage

A parallel coordinates plot represents each variable in a multivariate data set as a separate axis. Individual samples of that data set are represented as a polyline that pass through each variable axis at positions that correspond to data values. This class can generate parallel coordinates plots identical to its superclass (vtkParallelCoordinatesRepresentation) and has the same interaction styles.

In addition to the standard parallel coordinates plot, this class also can draw a histogram summary of the parallel coordinates plot. Rather than draw every row in an input data set, first it computes a 2D histogram for all neighboring variable axes, then it draws bar (thickness corresponds to bin size) for each bin the histogram with opacity weighted by the number of rows contained in the bin. The result is essentially a density map.

Because this emphasizes dense regions over sparse outliers, this class also uses a vtkComputeHistogram2DOutliers instance to identify outlier table rows and draws those as standard parallel coordinates lines.

To create an instance of class vtkParallelCoordinatesHistogramRepresentation, simply invoke its constructor as follows

```
obj = vtkParallelCoordinatesHistogramRepresentation
```

40.10.2 Methods

The class vtkParallelCoordinatesHistogramRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParallelCoordinatesHistogramRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParallelCoordinatesHistogramRepresentation = obj.NewInstance ()`
- `vtkParallelCoordinatesHistogramRepresentation = obj.SafeDownCast (vtkObject o)`
- `obj.ApplyViewTheme (vtkViewTheme theme)` - Apply the theme to this view.
- `obj.SetUseHistograms (int )` - Whether to use the histogram rendering mode or the superclass's line rendering mode
- `int = obj.GetUseHistograms ()` - Whether to use the histogram rendering mode or the superclass's line rendering mode
- `obj.UseHistogramsOn ()` - Whether to use the histogram rendering mode or the superclass's line rendering mode
- `obj.UseHistogramsOff ()` - Whether to use the histogram rendering mode or the superclass's line rendering mode
- `obj.SetShowOutliers (int )` - Whether to compute and show outlier lines
- `int = obj.GetShowOutliers ()` - Whether to compute and show outlier lines
- `obj.ShowOutliersOn ()` - Whether to compute and show outlier lines
- `obj.ShowOutliersOff ()` - Whether to compute and show outlier lines
- `obj.SetHistogramLookupTableRange (double , double )` - Control over the range of the lookup table used to draw the histogram quads.

- `obj.SetHistogramLookupTableRange (double a[2])` - Control over the range of the lookup table used to draw the histogram quads.
- `double = obj. GetHistogramLookupTableRange ()` - Control over the range of the lookup table used to draw the histogram quads.
- `obj.SetPreferredNumberOfOutliers (int )` - Target maximum number of outliers to be drawn, although not guaranteed.
- `int = obj.GetPreferredNumberOfOutliers ()` - Target maximum number of outliers to be drawn, although not guaranteed.
- `int = obj.SwapAxisPositions (int position1, int position2)` - Calls superclass swap, and assures that only histograms affected by the swap get recomputed.
- `int = obj.SetRangeAtPosition (int position, double range[2])` - Calls the superclass method, and assures that only the two histograms affect by this call get recomputed.

40.11 vtkParallelCoordinatesRepresentation

40.11.1 Usage

A parallel coordinates plot represents each variable in a multivariate data set as a separate axis. Individual samples of that data set are represented as a polyline that pass through each variable axis at positions that correspond to data values. vtkParallelCoordinatesRepresentation generates this plot when added to a vtkParallelCoordinatesView, which handles interaction and highlighting. Sample polylines can alternatively be represented as s-curves by enabling the UseCurves flag.

There are three selection modes: lasso, angle, and function. Lasso selection picks sample lines that pass through a polyline. Angle selection picks sample lines that have similar slope to a line segment. Function selection picks sample lines that are near a linear function defined on two variables. This function specified by passing two (x,y) variable value pairs.

All primitives are plotted in normalized view coordinates [0,1].

To create an instance of class vtkParallelCoordinatesRepresentation, simply invoke its constructor as follows

```
obj = vtkParallelCoordinatesRepresentation
```

40.11.2 Methods

The class vtkParallelCoordinatesRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParallelCoordinatesRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParallelCoordinatesRepresentation = obj.NewInstance ()`
- `vtkParallelCoordinatesRepresentation = obj.SafeDownCast (vtkObject o)`
- `obj.ApplyViewTheme (vtkViewTheme theme)` - Apply the theme to this view. CellColor is used for line coloring and titles. EdgeLabelColor is used for axis color. CellOpacity is used for line opacity.
- `string = obj.GetHoverText (vtkView view, int x, int y)` - Returns the hover text at an x,y location.

- `int = obj.SetPositionAndSize (double position, double size)` - Change the position of the plot
- `int = obj.GetPositionAndSize (double position, double size)` - Change the position of the plot
- `obj.SetAxisTitles (vtkStringArray )` - Set/Get the axis titles
- `obj.SetAxisTitles (vtkAlgorithmOutput )` - Set/Get the axis titles
- `obj.SetPlotTitle (string )` - Set the title for the entire plot
- `int = obj.GetNumberOfAxes ()` - Get the number of axes in the plot
- `int = obj.GetNumberOfSamples ()`
- `obj.SetNumberOfAxisLabels (int num)` - Set/Get the number of labels to display on each axis
- `int = obj.GetNumberOfAxisLabels ()` - Set/Get the number of labels to display on each axis
- `int = obj.SwapAxisPositions (int position1, int position2)` - Move an axis to a particular screen position. Using these methods requires an Update() before they will work properly.
- `int = obj.SetXCoordinateOfPosition (int position, double xcoord)` - Move an axis to a particular screen position. Using these methods requires an Update() before they will work properly.
- `double = obj.GetXCoordinateOfPosition (int axis)` - Move an axis to a particular screen position. Using these methods requires an Update() before they will work properly.
- `obj.GetXCoordinatesOfPositions (double coords)` - Move an axis to a particular screen position. Using these methods requires an Update() before they will work properly.
- `int = obj.GetPositionNearXCoordinate (double xcoord)` - Move an axis to a particular screen position. Using these methods requires an Update() before they will work properly.
- `obj.SetUseCurves (int )` - Whether or not to display using curves
- `int = obj.GetUseCurves ()` - Whether or not to display using curves
- `obj.UseCurvesOn ()` - Whether or not to display using curves
- `obj.UseCurvesOff ()` - Whether or not to display using curves
- `obj.SetCurveResolution (int )` - Resolution of the curves displayed, enabled by setting UseCurves
- `int = obj.GetCurveResolution ()` - Resolution of the curves displayed, enabled by setting UseCurves
- `double = obj.GetLineOpacity ()` - Access plot properties
- `double = obj.GetFontSize ()` - Access plot properties
- `double = obj. GetLineColor ()` - Access plot properties
- `double = obj. GetAxisColor ()` - Access plot properties
- `double = obj. GetAxisLabelColor ()` - Access plot properties
- `obj.SetLineOpacity (double )` - Access plot properties
- `obj.SetFontSize (double )` - Access plot properties
- `obj.SetLineColor (double , double , double )` - Access plot properties

- `obj.SetLineColor (double  a[3])` - Access plot properties
- `obj.SetAxisColor (double , double , double )` - Access plot properties
- `obj.SetAxisColor (double  a[3])` - Access plot properties
- `obj.SetAxisLabelColor (double , double , double )` - Access plot properties
- `obj.SetAxisLabelColor (double  a[3])` - Access plot properties
- `obj.SetAngleBrushThreshold (double )` - Maximum angle difference (in degrees) of selection using angle/function brushes
- `double = obj.GetAngleBrushThreshold ()` - Maximum angle difference (in degrees) of selection using angle/function brushes
- `obj.SetFunctionBrushThreshold (double )` - Maximum angle difference (in degrees) of selection using angle/function brushes
- `double = obj.GetFunctionBrushThreshold ()` - Maximum angle difference (in degrees) of selection using angle/function brushes
- `int = obj.GetRangeAtPosition (int position, double range[2])` - Set/get the value range of the axis at a particular screen position
- `int = obj.SetRangeAtPosition (int position, double range[2])` - Set/get the value range of the axis at a particular screen position
- `obj.ResetAxes ()` - Reset the axes to their default positions and orders
- `obj.LassoSelect (int brushClass, int brushOperator, vtkPoints brushPoints)` - Do a selection of the lines. See the main description for how to use these functions. RangeSelect is currently stubbed out.
- `obj.AngleSelect (int brushClass, int brushOperator, double p1, double p2)` - Do a selection of the lines. See the main description for how to use these functions. RangeSelect is currently stubbed out.
- `obj.FunctionSelect (int brushClass, int brushOperator, double p1, double p2, double q1, double q2)` - Do a selection of the lines. See the main description for how to use these functions. RangeSelect is currently stubbed out.
- `obj.RangeSelect (int brushClass, int brushOperator, double p1, double p2)` - Do a selection of the lines. See the main description for how to use these functions. RangeSelect is currently stubbed out.

40.12 vtkParallelCoordinatesView

40.12.1 Usage

This class manages interaction with the vtkParallelCoordinatesRepresentation. There are two inspection modes: axis manipulation and line selection. In axis manipulation mode, PC axes can be dragged and reordered with the LMB, axis ranges can be increased/decreased by dragging up/down with the LMB, and RMB controls zoom and pan.

In line selection mode, there are three subclasses of selections: lasso, angle, and function selection. Lasso selection lets the user brush a line and select all PC lines that pass nearby. Angle selection lets the user draw a representative line between axes and select all lines that have similar orientation. Function selection lets the user draw two representative lines between a pair of axes and select all lines that match the linear interpolation of those lines.

There are several self-explanatory operators for combining selections: ADD, SUBTRACT REPLACE, and INTERSECT.

To create an instance of class vtkParallelCoordinatesView, simply invoke its constructor as follows

```
obj = vtkParallelCoordinatesView
```

40.12.2 Methods

The class vtkParallelCoordinatesView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParallelCoordinatesView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParallelCoordinatesView = obj.NewInstance ()`
- `vtkParallelCoordinatesView = obj.SafeDownCast (vtkObject o)`
- `obj.SetBrushMode (int )`
- `obj.SetBrushModeToLasso ()`
- `obj.SetBrushModeToAngle ()`
- `obj.SetBrushModeToFunction ()`
- `obj.SetBrushModeToAxisThreshold ()`
- `int = obj.GetBrushMode ()`
- `obj.SetBrushOperator (int )`
- `obj.SetBrushOperatorToAdd ()`
- `obj.SetBrushOperatorToSubtract ()`
- `obj.SetBrushOperatorToIntersect ()`
- `obj.SetBrushOperatorToReplace ()`
- `int = obj.GetBrushOperator ()`
- `obj.SetInspectMode (int )`
- `obj.SetInspectModeToManipulateAxes ()`
- `obj.SetInpsectModeToSelectData ()`
- `int = obj.GetInspectMode ()`
- `obj.SetMaximumNumberOfBrushPoints (int )`
- `int = obj.GetMaximumNumberOfBrushPoints ()`
- `obj.SetCurrentBrushClass (int )`
- `int = obj.GetCurrentBrushClass ()`
- `obj.ApplyViewTheme (vtkViewTheme theme)`

40.13 vtkRenderedGraphRepresentation

40.13.1 Usage

To create an instance of class vtkRenderedGraphRepresentation, simply invoke its constructor as follows

```
obj = vtkRenderedGraphRepresentation
```

40.13.2 Methods

The class vtkRenderedGraphRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderedGraphRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderedGraphRepresentation = obj.NewInstance ()`
- `vtkRenderedGraphRepresentation = obj.SafeDownCast (vtkObject o)`
- `obj.SetVertexLabelArrayName (string name)`
- `string = obj.GetVertexLabelArrayName ()`
- `obj.SetVertexLabelPriorityArrayName (string name)`
- `string = obj.GetVertexLabelPriorityArrayName ()`
- `obj.SetVertexLabelVisibility (bool b)`
- `bool = obj.GetVertexLabelVisibility ()`
- `obj.VertexLabelVisibilityOn ()`
- `obj.VertexLabelVisibilityOff ()`
- `obj.SetVertexLabelTextProperty (vtkTextProperty p)`
- `vtkTextProperty = obj.GetVertexLabelTextProperty ()`
- `obj.SetVertexHoverArrayName (string )`
- `string = obj.GetVertexHoverArrayName ()`
- `obj.SetHideVertexLabelsOnInteraction (bool )` - Whether to hide the display of vertex labels during mouse interaction. Default is off.
- `bool = obj.GetHideVertexLabelsOnInteraction ()` - Whether to hide the display of vertex labels during mouse interaction. Default is off.
- `obj.HideVertexLabelsOnInteractionOn ()` - Whether to hide the display of vertex labels during mouse interaction. Default is off.
- `obj.HideVertexLabelsOnInteractionOff ()` - Whether to hide the display of vertex labels during mouse interaction. Default is off.
- `obj.SetEdgeLabelArrayName (string name)`
- `string = obj.GetEdgeLabelArrayName ()`

- obj.SetEdgeLabelPriorityArrayName (string name)
- string = obj.GetEdgeLabelPriorityArrayName ()
- obj.SetEdgeLabelVisibility (bool b)
- bool = obj.GetEdgeLabelVisibility ()
- obj.EdgeLabelVisibilityOn ()
- obj.EdgeLabelVisibilityOff ()
- obj.SetEdgeLabelTextProperty (vtkTextProperty p)
- vtkTextProperty = obj.GetEdgeLabelTextProperty ()
- obj.SetEdgeHoverArrayName (string)
- string = obj.GetEdgeHoverArrayName ()
- obj.SetHideEdgeLabelsOnInteraction (bool) - Whether to hide the display of edge labels during mouse interaction. Default is off.
- bool = obj.GetHideEdgeLabelsOnInteraction () - Whether to hide the display of edge labels during mouse interaction. Default is off.
- obj.HideEdgeLabelsOnInteractionOn () - Whether to hide the display of edge labels during mouse interaction. Default is off.
- obj.HideEdgeLabelsOnInteractionOff () - Whether to hide the display of edge labels during mouse interaction. Default is off.
- obj.SetVertexIconArrayName (string name)
- string = obj.GetVertexIconArrayName ()
- obj.SetVertexIconPriorityArrayName (string name)
- string = obj.GetVertexIconPriorityArrayName ()
- obj.SetVertexIconVisibility (bool b)
- bool = obj.GetVertexIconVisibility ()
- obj.VertexIconVisibilityOn ()
- obj.VertexIconVisibilityOff ()
- obj.AddVertexIconType (string name, int type)
- obj.ClearVertexIconTypes ()
- obj.SetUseVertexIconTypeMap (bool b)
- bool = obj.GetUseVertexIconTypeMap ()
- obj.UseVertexIconTypeMapOn ()
- obj.UseVertexIconTypeMapOff ()
- obj.SetVertexIconAlignment (int align)
- int = obj.GetVertexIconAlignment ()
- obj.SetVertexSelectedIcon (int icon)

- `int = obj.GetVertexSelectedIcon ()`
- `obj.SetVertexIconSelectionMode (int mode)` - Set the mode to one of ¡ul¿ ¡li¿vtkApplyIcons::SELECTED_ICON - use VertexSelectedIcon ¡li¿vtkApplyIcons::SELECTED_OFFSET - use VertexSelectedIcon as offset ¡li¿vtkApplyIcons::ANNOTATION_ICON - use current annotation icon ¡li¿vtkApplyIcons::IGNORE_SELECTION - ignore selected elements ¡/ul¿ The default is IGNORE_SELECTION.
- `int = obj.GetVertexIconSelectionMode ()` - Set the mode to one of ¡ul¿ ¡li¿vtkApplyIcons::SELECTED_ICON - use VertexSelectedIcon ¡li¿vtkApplyIcons::SELECTED_OFFSET - use VertexSelectedIcon as offset ¡li¿vtkApplyIcons::ANNOTATION_ICON - use current annotation icon ¡li¿vtkApplyIcons::IGNORE_SELECTION - ignore selected elements ¡/ul¿ The default is IGNORE_SELECTION.
- `obj.SetVertexIconSelectionModeToSelectedIcon ()` - Set the mode to one of ¡ul¿ ¡li¿vtkApplyIcons::SELECTED_IC - use VertexSelectedIcon ¡li¿vtkApplyIcons::SELECTED_OFFSET - use VertexSelectedIcon as offset ¡li¿vtkApplyIcons::ANNOTATION_ICON - use current annotation icon ¡li¿vtkApplyIcons::IGNORE_SELECTION - ignore selected elements ¡/ul¿ The default is IGNORE_SELECTION.
- `obj.SetVertexIconSelectionModeToSelectedOffset ()` - Set the mode to one of ¡ul¿ ¡li¿vtkApplyIcons::SELECTED - use VertexSelectedIcon ¡li¿vtkApplyIcons::SELECTED_OFFSET - use VertexSelectedIcon as offset ¡li¿vtkApplyIcons::ANNOTATION_ICON - use current annotation icon ¡li¿vtkApplyIcons::IGNORE_SELECTION - ignore selected elements ¡/ul¿ The default is IGNORE_SELECTION.
- `obj.SetVertexIconSelectionModeToAnnotationIcon ()` - Set the mode to one of ¡ul¿ ¡li¿vtkApplyIcons::SELECTED - use VertexSelectedIcon ¡li¿vtkApplyIcons::SELECTED_OFFSET - use VertexSelectedIcon as offset ¡li¿vtkApplyIcons::ANNOTATION_ICON - use current annotation icon ¡li¿vtkApplyIcons::IGNORE_SELECTION - ignore selected elements ¡/ul¿ The default is IGNORE_SELECTION.
- `obj.SetVertexIconSelectionModeToIgnoreSelection ()`
- `obj.SetEdgeIconArrayName (string name)`
- `string = obj.GetEdgeIconArrayName ()`
- `obj.SetEdgeIconPriorityArrayName (string name)`
- `string = obj.GetEdgeIconPriorityArrayName ()`
- `obj.SetEdgeIconVisibility (bool b)`
- `bool = obj.GetEdgeIconVisibility ()`
- `obj.EdgeIconVisibilityOn ()`
- `obj.EdgeIconVisibilityOff ()`
- `obj.AddEdgeIconType (string name, int type)`
- `obj.ClearEdgeIconTypes ()`
- `obj.SetUseEdgeIconTypeMap (bool b)`
- `bool = obj.GetUseEdgeIconTypeMap ()`
- `obj.UseEdgeIconTypeMapOn ()`
- `obj.UseEdgeIconTypeMapOff ()`
- `obj.SetEdgeIconAlignment (int align)`
- `int = obj.GetEdgeIconAlignment ()`
- `obj.SetColorVerticesByArray (bool b)`

- `bool = obj.GetColorVerticesByArray ()`
- `obj.ColorVerticesByArrayOn ()`
- `obj.ColorVerticesByArrayOff ()`
- `obj.SetVertexColorArrayName (string name)`
- `string = obj.GetVertexColorArrayName ()`
- `obj.SetColorEdgesByArray (bool b)`
- `bool = obj.GetColorEdgesByArray ()`
- `obj.ColorEdgesByArrayOn ()`
- `obj.ColorEdgesByArrayOff ()`
- `obj.SetEdgeColorArrayName (string name)`
- `string = obj.GetEdgeColorArrayName ()`
- `obj.SetEnableVerticesByArray (bool b)`
- `bool = obj.GetEnableVerticesByArray ()`
- `obj.EnableVerticesByArrayOn ()`
- `obj.EnableVerticesByArrayOff ()`
- `obj.SetEnabledVerticesArrayName (string name)`
- `string = obj.GetEnabledVerticesArrayName ()`
- `obj.SetEnableEdgesByArray (bool b)`
- `bool = obj.GetEnableEdgesByArray ()`
- `obj.EnableEdgesByArrayOn ()`
- `obj.EnableEdgesByArrayOff ()`
- `obj.SetEnabledEdgesArrayName (string name)`
- `string = obj.GetEnabledEdgesArrayName ()`
- `obj.SetEdgeVisibility (bool b)`
- `bool = obj.GetEdgeVisibility ()`
- `obj.EdgeVisibilityOn ()`
- `obj.EdgeVisibilityOff ()`
- `obj.SetLayoutStrategy (vtkGraphLayoutStrategy strategy)` - Set/get the graph layout strategy.
- `vtkGraphLayoutStrategy = obj.GetLayoutStrategy ()` - Set/get the graph layout strategy.
- `obj.SetLayoutStrategy (string name)` - Get/set the layout strategy by name.
- `string = obj.GetLayoutStrategyName ()` - Get/set the layout strategy by name.
- `obj.SetLayoutStrategyToRandom ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToForceDirected ()` - Set predefined layout strategies.

- `obj.SetLayoutStrategyToSimple2D ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToClustering2D ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToCommunity2D ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToFast2D ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToPassThrough ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToCircular ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToTree ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToCosmicTree ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToCone ()` - Set predefined layout strategies.
- `obj.SetLayoutStrategyToSpanTree ()` - Set the layout strategy to use coordinates from arrays. The x array must be specified. The y and z arrays are optional.
- `obj.SetLayoutStrategyToAssignCoordinates (string xarr, string yarr, string zarr)` - Set the layout strategy to use coordinates from arrays. The x array must be specified. The y and z arrays are optional.
- `obj.SetLayoutStrategyToTree (bool radial, double angle, double leafSpacing, double logSpacing)` - Set the layout strategy to a tree layout. Radial indicates whether to do a radial or standard top-down tree layout. The angle parameter is the angular distance spanned by the tree. Leaf spacing is a value from 0 to 1 indicating how much of the radial layout should be allocated to leaf nodes (as opposed to between tree branches). The log spacing value is a non-negative value where ¿ 1 will create expanding levels, ¡ 1 will create contracting levels, and = 1 makes all levels the same size. See vtkTreeLayoutStrategy for more information.
- `obj.SetLayoutStrategyToCosmicTree (string nodeSizeArrayName, bool sizeLeafNodesOnlytrue, int layout` - Set the layout strategy to a cosmic tree layout. nodeSizeArrayName is the array used to size the circles (default is NULL, which makes leaf nodes the same size). sizeLeafNodesOnly only uses the leaf node sizes, and computes the parent size as the sum of the child sizes (default true). layoutDepth stops layout at a certain depth (default is 0, which does the entire tree). layoutRoot is the vertex that will be considered the root node of the layout (default is -1, which will use the tree's root). See vtkCosmicTreeLayoutStrategy for more information.
- `obj.SetEdgeLayoutStrategy (vtkEdgeLayoutStrategy strategy)` - Set/get the graph layout strategy.
- `vtkEdgeLayoutStrategy = obj.GetEdgeLayoutStrategy ()` - Set/get the graph layout strategy.
- `obj.SetEdgeLayoutStrategyToArcParallel ()` - Set/get the graph layout strategy.
- `obj.SetEdgeLayoutStrategyToPassThrough ()` - Set the edge layout strategy to a geospatial arced strategy appropriate for vtkGeoView.
- `obj.SetEdgeLayoutStrategyToGeo (double explodeFactor)` - Set the edge layout strategy to a geospatial arced strategy appropriate for vtkGeoView.
- `obj.SetEdgeLayoutStrategy (string name)` - Set the edge layout strategy by name.
- `string = obj.GetEdgeLayoutStrategyName ()` - Set the edge layout strategy by name.
- `obj.ApplyViewTheme (vtkViewTheme theme)` - Apply a theme to this representation.
- `obj.SetGlyphType (int type)` - Set the graph vertex glyph type.

- `int = obj.GetGlyphType ()` - Set the graph vertex glyph type.
- `obj.SetScaling (bool b)` - Set whether to scale vertex glyphs.
- `bool = obj.GetScaling ()` - Set whether to scale vertex glyphs.
- `obj.ScalingOn ()` - Set whether to scale vertex glyphs.
- `obj.ScalingOff ()` - Set whether to scale vertex glyphs.
- `obj.SetScalingArrayName (string name)` - Set the glyph scaling array name.
- `string = obj.GetScalingArrayName ()` - Set the glyph scaling array name.
- `obj.SetVertexScalarBarVisibility (bool b)` - Vertex/edge scalar bar visibility.
- `bool = obj.GetVertexScalarBarVisibility ()` - Vertex/edge scalar bar visibility.
- `obj.SetEdgeScalarBarVisibility (bool b)` - Vertex/edge scalar bar visibility.
- `bool = obj.GetEdgeScalarBarVisibility ()` - Vertex/edge scalar bar visibility.
- `bool = obj.IsLayoutComplete ()` - Whether the current graph layout is complete.
- `obj.UpdateLayout ()` - Performs another iteration on the graph layout.
- `obj.ComputeSelectedGraphBounds (double bounds[6])` - Compute the bounding box of the selected subgraph.

40.14 vtkRenderedHierarchyRepresentation

40.14.1 Usage

To create an instance of class vtkRenderedHierarchyRepresentation, simply invoke its constructor as follows

```
obj = vtkRenderedHierarchyRepresentation
```

40.14.2 Methods

The class vtkRenderedHierarchyRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderedHierarchyRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderedHierarchyRepresentation = obj.NewInstance ()`
- `vtkRenderedHierarchyRepresentation = obj.SafeDownCast (vtkObject o)`
- `obj.SetGraphEdgeLabelArrayName (string name)` -
- `obj.SetGraphEdgeLabelArrayName (string name, int idx)` -
- `string = obj.GetGraphEdgeLabelArrayName ()` -
- `string = obj.GetGraphEdgeLabelArrayName (int idx)` -
- `obj.SetGraphEdgeLabelVisibility (bool vis)`

- obj.SetGraphEdgeLabelVisibility (bool vis, int idx)
- bool = obj.GetGraphEdgeLabelVisibility ()
- bool = obj.GetGraphEdgeLabelVisibility (int idx)
- obj.GraphEdgeLabelVisibilityOn ()
- obj.GraphEdgeLabelVisibilityOff ()
- obj.SetGraphEdgeColorArrayName (string name)
- obj.SetGraphEdgeColorArrayName (string name, int idx)
- string = obj.GetGraphEdgeColorArrayName ()
- string = obj.GetGraphEdgeColorArrayName (int idx)
- obj.SetColorGraphEdgesByArray (bool vis)
- obj.SetColorGraphEdgesByArray (bool vis, int idx)
- bool = obj.GetColorGraphEdgesByArray ()
- bool = obj.GetColorGraphEdgesByArray (int idx)
- obj.ColorGraphEdgesByArrayOn ()
- obj.ColorGraphEdgesByArrayOff ()
- obj.SetGraphEdgeColorToSplineFraction ()
- obj.SetGraphEdgeColorToSplineFraction (int idx)
- obj.SetGraphVisibility (bool vis)
- obj.SetGraphVisibility (bool vis, int idx)
- bool = obj.GetGraphVisibility ()
- bool = obj.GetGraphVisibility (int idx)
- obj.GraphVisibilityOn ()
- obj.GraphVisibilityOff ()
- obj.SetBundlingStrength (double strength)
- obj.SetBundlingStrength (double strength, int idx)
- double = obj.GetBundlingStrength ()
- double = obj.GetBundlingStrength (int idx)
- obj.SetGraphEdgeLabelFontSize (int size)
- obj.SetGraphEdgeLabelFontSize (int size, int idx)
- int = obj.GetGraphEdgeLabelFontSize ()
- int = obj.GetGraphEdgeLabelFontSize (int idx)

40.15 vtkRenderedRepresentation

40.15.1 Usage

To create an instance of class vtkRenderedRepresentation, simply invoke its constructor as follows

```
obj = vtkRenderedRepresentation
```

40.15.2 Methods

The class vtkRenderedRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderedRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderedRepresentation = obj.NewInstance ()`
- `vtkRenderedRepresentation = obj.SafeDownCast (vtkObject o)`
- `obj.SetLabelRenderMode (int )` - Set the label render mode. vtkRenderView::QT - Use Qt-based labeler with fitted labeling and unicode support. Requires VTK_USE_QT to be on. vtkRenderView::FREETYPE - Use standard freetype text rendering.
- `int = obj.GetLabelRenderMode ()` - Set the label render mode. vtkRenderView::QT - Use Qt-based labeler with fitted labeling and unicode support. Requires VTK_USE_QT to be on. vtkRenderView::FREETYPE - Use standard freetype text rendering.

40.16 vtkRenderedSurfaceRepresentation

40.16.1 Usage

vtkRenderedSurfaceRepresentation is used to show a geometric dataset in a view. The representation uses a vtkGeometryFilter to convert the dataset to polygonal data (e.g. volumetric data is converted to its external surface). The representation may then be added to vtkRenderView.

To create an instance of class vtkRenderedSurfaceRepresentation, simply invoke its constructor as follows

```
obj = vtkRenderedSurfaceRepresentation
```

40.16.2 Methods

The class vtkRenderedSurfaceRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderedSurfaceRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderedSurfaceRepresentation = obj.NewInstance ()`
- `vtkRenderedSurfaceRepresentation = obj.SafeDownCast (vtkObject o)`
- `obj.SetCellColorArrayName (string arrayName)`
- `string = obj.GetCellColorArrayName ()` - Apply a theme to this representation.
- `obj.ApplyViewTheme (vtkViewTheme theme)` - Apply a theme to this representation.

40.17 vtkRenderedTreeAreaRepresentation

40.17.1 Usage

To create an instance of class vtkRenderedTreeAreaRepresentation, simply invoke its constructor as follows

```
obj = vtkRenderedTreeAreaRepresentation
```

40.17.2 Methods

The class vtkRenderedTreeAreaRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderedTreeAreaRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderedTreeAreaRepresentation = obj.NewInstance ()`
- `vtkRenderedTreeAreaRepresentation = obj.SafeDownCast (vtkObject o)`
- `obj.SetLabelRenderMode (int mode)` - Set the label render mode. QT - Use vtkQtTreeRingLabeler with fitted labeling and unicode support. Requires VTK_USE_QT to be on. FREETYPE - Use standard freetype text rendering.
- `obj.SetAreaLabelArrayName (string name)` - The array to use for area labeling. Default is "label".
- `string = obj.GetAreaLabelArrayName ()` - The array to use for area labeling. Default is "label".
- `obj.SetAreaSizeArrayName (string name)` - The array to use for area sizes. Default is "size".
- `string = obj.GetAreaSizeArrayName ()` - The array to use for area sizes. Default is "size".
- `obj.SetAreaLabelPriorityArrayName (string name)` - The array to use for area labeling priority. Default is "GraphVertexDegree".
- `string = obj.GetAreaLabelPriorityArrayName ()` - The array to use for area labeling priority. Default is "GraphVertexDegree".
- `obj.SetGraphEdgeLabelArrayName (string name)` - The array to use for edge labeling. Default is "label".
- `obj.SetGraphEdgeLabelArrayName (string name, int idx)` - The array to use for edge labeling. Default is "label".
- `string = obj.GetGraphEdgeLabelArrayName ()` - The array to use for edge labeling. Default is "label".
- `string = obj.GetGraphEdgeLabelArrayName (int idx)` - The array to use for edge labeling. Default is "label".
- `obj.SetGraphEdgeLabelTextProperty (vtkTextProperty tp)` - The text property for the graph edge labels.
- `obj.SetGraphEdgeLabelTextProperty (vtkTextProperty tp, int idx)` - The text property for the graph edge labels.
- `vtkTextProperty = obj.GetGraphEdgeLabelTextProperty ()` - The text property for the graph edge labels.

- `vtkTextProperty = obj.GetGraphEdgeLabelTextProperty (int idx)` - The text property for the graph edge labels.
- `obj.SetAreaHoverArrayName (string )` - The name of the array whose value appears when the mouse hovers over a rectangle in the treemap.
- `string = obj.GetAreaHoverArrayName ()` - The name of the array whose value appears when the mouse hovers over a rectangle in the treemap.
- `obj.SetAreaLabelVisibility (bool vis)` - Whether to show area labels. Default is off.
- `bool = obj.GetAreaLabelVisibility ()` - Whether to show area labels. Default is off.
- `obj.AreaLabelVisibilityOn ()` - Whether to show area labels. Default is off.
- `obj.AreaLabelVisibilityOff ()` - Whether to show area labels. Default is off.
- `obj.SetAreaLabelTextProperty (vtkTextProperty tp)` - The text property for the area labels.
- `vtkTextProperty = obj.GetAreaLabelTextProperty ()` - The text property for the area labels.
- `obj.SetGraphEdgeLabelVisibility (bool vis)` - Whether to show edge labels. Default is off.
- `obj.SetGraphEdgeLabelVisibility (bool vis, int idx)` - Whether to show edge labels. Default is off.
- `bool = obj.GetGraphEdgeLabelVisibility ()` - Whether to show edge labels. Default is off.
- `bool = obj.GetGraphEdgeLabelVisibility (int idx)` - Whether to show edge labels. Default is off.
- `obj.GraphEdgeLabelVisibilityOn ()` - Whether to show edge labels. Default is off.
- `obj.GraphEdgeLabelVisibilityOff ()` - Whether to show edge labels. Default is off.
- `obj.SetAreaColorArrayName (string name)` - The array to use for coloring vertices. Default is "color".
- `string = obj.GetAreaColorArrayName ()` - The array to use for coloring vertices. Default is "color".
- `obj.SetColorAreasByArray (bool vis)` - Whether to color vertices. Default is off.
- `bool = obj.GetColorAreasByArray ()` - Whether to color vertices. Default is off.
- `obj.ColorAreasByArrayOn ()` - Whether to color vertices. Default is off.
- `obj.ColorAreasByArrayOff ()` - Whether to color vertices. Default is off.
- `obj.SetGraphEdgeColorArrayName (string name)` - The array to use for coloring edges. Default is "color".
- `obj.SetGraphEdgeColorArrayName (string name, int idx)` - The array to use for coloring edges. Default is "color".
- `string = obj.GetGraphEdgeColorArrayName ()` - The array to use for coloring edges. Default is "color".
- `string = obj.GetGraphEdgeColorArrayName (int idx)` - The array to use for coloring edges. Default is "color".
- `obj.SetGraphEdgeColorToSplineFraction ()` - Set the color to be the spline fraction
- `obj.SetGraphEdgeColorToSplineFraction (int idx)` - Set the color to be the spline fraction

- `obj.SetColorGraphEdgesByArray (bool vis)` - Whether to color edges. Default is off.
- `obj.SetColorGraphEdgesByArray (bool vis, int idx)` - Whether to color edges. Default is off.
- `bool = obj.GetColorGraphEdgesByArray ()` - Whether to color edges. Default is off.
- `bool = obj.GetColorGraphEdgesByArray (int idx)` - Whether to color edges. Default is off.
- `obj.ColorGraphEdgesByArrayOn ()` - Whether to color edges. Default is off.
- `obj.ColorGraphEdgesByArrayOff ()` - Whether to color edges. Default is off.
- `obj.SetGraphHoverArrayName (string name)` - The name of the array whose value appears when the mouse hovers over a graph edge.
- `obj.SetGraphHoverArrayName (string name, int idx)` - The name of the array whose value appears when the mouse hovers over a graph edge.
- `string = obj.GetGraphHoverArrayName ()` - The name of the array whose value appears when the mouse hovers over a graph edge.
- `string = obj.GetGraphHoverArrayName (int idx)` - The name of the array whose value appears when the mouse hovers over a graph edge.
- `obj.SetShrinkPercentage (double value)` - Set the region shrink percentage between 0.0 and 1.0.
- `double = obj.GetShrinkPercentage ()` - Set the region shrink percentage between 0.0 and 1.0.
- `obj.SetGraphBundlingStrength (double strength)` - Set the bundling strength.
- `obj.SetGraphBundlingStrength (double strength, int idx)` - Set the bundling strength.
- `double = obj.GetGraphBundlingStrength ()` - Set the bundling strength.
- `double = obj.GetGraphBundlingStrength (int idx)` - Set the bundling strength.
- `obj.SetGraphSplineType (int type, int idx)` - Sets the spline type for the graph edges. vtkSplineGraphEdges::CUSTOM uses a vtkCardinalSpline. vtkSplineGraphEdges::BSPLINE uses a b-spline. The default is CUSTOM.
- `int = obj.GetGraphSplineType (int idx)` - Sets the spline type for the graph edges. vtkSplineGraphEdges::CUSTOM uses a vtkCardinalSpline. vtkSplineGraphEdges::BSPLINE uses a b-spline. The default is CUSTOM.
- `obj.SetAreaLayoutStrategy (vtkAreaLayoutStrategy strategy)` - The layout strategy for producing spatial regions for the tree.
- `vtkAreaLayoutStrategy = obj.GetAreaLayoutStrategy ()` - The layout strategy for producing spatial regions for the tree.
- `obj.SetAreaToPolyData (vtkPolyDataAlgorithm areaToPoly)` - The filter for converting areas to polydata. This may e.g. be vtkTreeMapToPolyData or vtkTreeRingToPolyData. The filter must take a vtkTree as input and produce vtkPolyData.
- `vtkPolyDataAlgorithm = obj.GetAreaToPolyData ()` - The filter for converting areas to polydata. This may e.g. be vtkTreeMapToPolyData or vtkTreeRingToPolyData. The filter must take a vtkTree as input and produce vtkPolyData.
- `obj.SetUseRectangularCoordinates (bool )` - Whether the area represents radial or rectangular coordinates.

- `bool = obj.GetUseRectangularCoordinates ()` - Whether the area represents radial or rectangular coordinates.
- `obj.UseRectangularCoordinatesOn ()` - Whether the area represents radial or rectangular coordinates.
- `obj.UseRectangularCoordinatesOff ()` - Whether the area represents radial or rectangular coordinates.
- `obj.SetAreaLabelMapper (vtkLabeledDataMapper mapper)` - The mapper for rendering labels on areas. This may e.g. be vtkDynamic2DLabelMapper or vtkTreeMapLabelMapper.
- `vtkLabeledDataMapper = obj.GetAreaLabelMapper ()` - The mapper for rendering labels on areas. This may e.g. be vtkDynamic2DLabelMapper or vtkTreeMapLabelMapper.
- `obj.ApplyViewTheme (vtkViewTheme theme)` - Apply the theme to this view.
- `obj.SetEdgeScalarBarVisibility (bool b)` - Visibility of scalar bar actor for edges.
- `bool = obj.GetEdgeScalarBarVisibility ()` - Visibility of scalar bar actor for edges.

40.18 vtkRenderView

40.18.1 Usage

vtkRenderView is a view which contains a vtkRenderer. You may add vtkActors directly to the renderer, or add certain vtkDataRepresentation subclasses to the renderer. The render view supports drag selection with the mouse to select cells.

This class is also the parent class for any more specialized view which uses a renderer.

To create an instance of class vtkRenderView, simply invoke its constructor as follows

```
obj = vtkRenderView
```

40.18.2 Methods

The class vtkRenderView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRenderView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRenderView = obj.NewInstance ()`
- `vtkRenderView = obj.SafeDownCast (vtkObject o)`
- `vtkRenderer = obj.GetRenderer ()` - Gets the renderer for this view.
- `vtkRenderWindow = obj.GetRenderWindow ()` - Get a handle to the render window.
- `vtkRenderWindowInteractor = obj.GetInteractor ()` - The render window interactor.
- `obj.SetInteractor (vtkRenderWindowInteractor interactor)` - The render window interactor.
- `obj.SetInteractorStyle (vtkInteractorObserver style)` - The interactor style associated with the render view.
- `vtkInteractorObserver = obj.GetInteractorStyle ()` - The interactor style associated with the render view.

- `obj.SetInteractionMode (int mode)` - Set the interaction mode for the view. Choices are: vtkRenderView::INTERACTION_MODE_2D - 2D interactor vtkRenderView::INTERACTION_MODE_3D - 3D interactor
- `int = obj.GetInteractionMode ()` - Set the interaction mode for the view. Choices are: vtkRenderView::INTERACTION_MODE_2D - 2D interactor vtkRenderView::INTERACTION_MODE_3D - 3D interactor
- `obj.SetInteractionModeTo2D ()` - Set the interaction mode for the view. Choices are: vtkRenderView::INTERACTION_MODE_2D - 2D interactor vtkRenderView::INTERACTION_MODE_3D - 3D interactor
- `obj.SetInteractionModeTo3D ()` - Applies a view theme to this view.
- `obj.ApplyViewTheme (vtkViewTheme theme)` - Applies a view theme to this view.
- `obj.SetTransform (vtkAbstractTransform transform)` - Set the view's transform. All vtkRenderedRepresentations added to this view should use this transform.
- `vtkAbstractTransform = obj.GetTransform ()` - Set the view's transform. All vtkRenderedRepresentations added to this view should use this transform.
- `obj.SetDisplayHoverText (bool b)` - Whether the view should display hover text.
- `bool = obj.GetDisplayHoverText ()` - Whether the view should display hover text.
- `obj.DisplayHoverTextOn ()` - Whether the view should display hover text.
- `obj.DisplayHoverTextOff ()` - Whether the view should display hover text.
- `obj.SetSelectionMode (int )` - Sets the selection mode for the render view. SURFACE selection uses vtkHardwareSelector to perform a selection of visible cells. FRUSTUM selection just creates a view frustum selection, which will select everything in the frustum.
- `int = obj.GetSelectionModeMinValue ()` - Sets the selection mode for the render view. SURFACE selection uses vtkHardwareSelector to perform a selection of visible cells. FRUSTUM selection just creates a view frustum selection, which will select everything in the frustum.
- `int = obj.GetSelectionModeMaxValue ()` - Sets the selection mode for the render view. SURFACE selection uses vtkHardwareSelector to perform a selection of visible cells. FRUSTUM selection just creates a view frustum selection, which will select everything in the frustum.
- `int = obj.GetSelectionMode ()` - Sets the selection mode for the render view. SURFACE selection uses vtkHardwareSelector to perform a selection of visible cells. FRUSTUM selection just creates a view frustum selection, which will select everything in the frustum.
- `obj.SetSelectionModeToSurface ()` - Sets the selection mode for the render view. SURFACE selection uses vtkHardwareSelector to perform a selection of visible cells. FRUSTUM selection just creates a view frustum selection, which will select everything in the frustum.
- `obj.SetSelectionModeToFrustum ()` - Updates the representations, then calls Render() on the render window associated with this view.
- `obj.Render ()` - Updates the representations, then calls Render() on the render window associated with this view.
- `obj.ResetCamera ()` - Updates the representations, then calls ResetCamera() on the renderer associated with this view.
- `obj.ResetCameraClippingRange ()` - Updates the representations, then calls ResetCameraClippingRange() on the renderer associated with this view.

- `obj.AddLabels (vtkAlgorithmOutput conn)` - Add labels from an input connection with an associated text property. The output must be a vtkLabelHierarchy (normally the output of vtkPointSetToLabelHierarchy).
- `obj.RemoveLabels (vtkAlgorithmOutput conn)` - Remove labels from an input connection.
- `obj.SetIconTexture (vtkTexture texture)` - Set the icon sheet to use for rendering icons.
- `vtkTexture = obj.GetIconTexture ()` - Set the icon sheet to use for rendering icons.
- `obj.SetIconSize (int , int )` - Set the size of each icon in the icon texture.
- `obj.SetIconSize (int  a[2])` - Set the size of each icon in the icon texture.
- `int = obj. GetIconSize ()` - Set the size of each icon in the icon texture.
- `obj.SetLabelPlacementMode (int mode)` - Label placement mode. NO_OVERLAP uses vtkLabelPlacementMapper, which has a faster startup time and works with 2D or 3D labels. ALL displays all labels (Warning: This may cause incredibly slow render times on datasets with more than a few hundred labels).
- `int = obj.GetLabelPlacementMode ()` - Label placement mode. NO_OVERLAP uses vtkLabelPlacementMapper, which has a faster startup time and works with 2D or 3D labels. ALL displays all labels (Warning: This may cause incredibly slow render times on datasets with more than a few hundred labels).
- `obj.SetLabelPlacementModeToNoOverlap ()` - Label placement mode. NO_OVERLAP uses vtkLabelPlacementMapper, which has a faster startup time and works with 2D or 3D labels. ALL displays all labels (Warning: This may cause incredibly slow render times on datasets with more than a few hundred labels).
- `obj.SetLabelPlacementModeToAll ()` - Label render mode. FREETYPE uses the freetype label rendering. QT uses more advanced Qt-based label rendering.
- `obj.SetLabelRenderMode (int mode)` - Label render mode. FREETYPE uses the freetype label rendering. QT uses more advanced Qt-based label rendering.
- `int = obj.GetLabelRenderMode ()` - Label render mode. FREETYPE uses the freetype label rendering. QT uses more advanced Qt-based label rendering.
- `obj.SetLabelRenderModeToFreetype ()` - Label render mode. FREETYPE uses the freetype label rendering. QT uses more advanced Qt-based label rendering.
- `obj.SetLabelRenderModeToQt ()`

40.19 vtkTreeAreaView

40.19.1 Usage

Takes a graph and a hierarchy (currently a tree) and lays out the graph vertices based on their categorization within the hierarchy.

.SEE ALSO vtkGraphLayoutView

.SECTION Thanks Thanks to Jason Shepherd for implementing this class

To create an instance of class vtkTreeAreaView, simply invoke its constructor as follows

```
obj = vtkTreeAreaView
```

40.19.2 Methods

The class vtkTreeAreaView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeAreaView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeAreaView = obj.NewInstance ()`
- `vtkTreeAreaView = obj.SafeDownCast (vtkObject o)`
- `vtkDataRepresentation = obj.SetTreeFromInputConnection (vtkAlgorithmOutput conn)` - Set the tree and graph representations to the appropriate input ports.
- `vtkDataRepresentation = obj.SetTreeFromInput (vtkTree input)` - Set the tree and graph representations to the appropriate input ports.
- `vtkDataRepresentation = obj.SetGraphFromInputConnection (vtkAlgorithmOutput conn)` - Set the tree and graph representations to the appropriate input ports.
- `vtkDataRepresentation = obj.SetGraphFromInput (vtkGraph input)` - Set the tree and graph representations to the appropriate input ports.
- `obj.SetAreaLabelArrayName (string name)` - The array to use for area labeling. Default is "label".
- `string = obj.GetAreaLabelArrayName ()` - The array to use for area labeling. Default is "label".
- `obj.SetAreaSizeArrayName (string name)` - The array to use for area sizes. Default is "size".
- `string = obj.GetAreaSizeArrayName ()` - The array to use for area sizes. Default is "size".
- `obj.SetLabelPriorityArrayName (string name)` - The array to use for area labeling priority. Default is "GraphVertexDegree".
- `string = obj.GetLabelPriorityArrayName ()` - The array to use for area labeling priority. Default is "GraphVertexDegree".
- `obj.SetEdgeLabelArrayName (string name)` - The array to use for edge labeling. Default is "label".
- `string = obj.GetEdgeLabelArrayName ()` - The array to use for edge labeling. Default is "label".
- `obj.SetAreaHoverArrayName (string name)` - The name of the array whose value appears when the mouse hovers over a rectangle in the treemap. This must be a string array.
- `string = obj.GetAreaHoverArrayName ()` - The name of the array whose value appears when the mouse hovers over a rectangle in the treemap. This must be a string array.
- `obj.SetAreaLabelVisibility (bool vis)` - Whether to show area labels. Default is off.
- `bool = obj.GetAreaLabelVisibility ()` - Whether to show area labels. Default is off.
- `obj.AreaLabelVisibilityOn ()` - Whether to show area labels. Default is off.
- `obj.AreaLabelVisibilityOff ()` - Whether to show area labels. Default is off.
- `obj.SetEdgeLabelVisibility (bool vis)` - Whether to show edge labels. Default is off.
- `bool = obj.GetEdgeLabelVisibility ()` - Whether to show edge labels. Default is off.

- `obj.EdgeLabelVisibilityOn ()` - Whether to show edge labels. Default is off.
- `obj.EdgeLabelVisibilityOff ()` - Whether to show edge labels. Default is off.
- `obj.SetAreaColorArrayName (string name)` - The array to use for coloring vertices. Default is "color".
- `string = obj.GetAreaColorArrayName ()` - The array to use for coloring vertices. Default is "color".
- `obj.SetColorAreas (bool vis)` - Whether to color vertices. Default is off.
- `bool = obj.GetColorAreas ()` - Whether to color vertices. Default is off.
- `obj.ColorAreasOn ()` - Whether to color vertices. Default is off.
- `obj.ColorAreasOff ()` - Whether to color vertices. Default is off.
- `obj.SetEdgeColorArrayName (string name)` - The array to use for coloring edges. Default is "color".
- `string = obj.GetEdgeColorArrayName ()` - The array to use for coloring edges. Default is "color".
- `obj.SetEdgeColorToSplineFraction ()` - Set the color to be the spline fraction
- `obj.SetShrinkPercentage (double value)` - Set the region shrink percentage between 0.0 and 1.0.
- `double = obj.GetShrinkPercentage ()` - Set the region shrink percentage between 0.0 and 1.0.
- `obj.SetColorEdges (bool vis)` - Whether to color edges. Default is off.
- `bool = obj.GetColorEdges ()` - Whether to color edges. Default is off.
- `obj.ColorEdgesOn ()` - Whether to color edges. Default is off.
- `obj.ColorEdgesOff ()` - Whether to color edges. Default is off.
- `obj.SetBundlingStrength (double strength)` - Set the bundling strength.
- `double = obj.GetBundlingStrength ()` - Set the bundling strength.
- `obj.SetAreaLabelFontSize (int size)` - The size of the font used for area labeling
- `int = obj.GetAreaLabelFontSize ()` - The size of the font used for area labeling
- `obj.SetEdgeLabelFontSize (int size)` - The size of the font used for edge labeling
- `int = obj.GetEdgeLabelFontSize ()` - The size of the font used for edge labeling
- `obj.SetLayoutStrategy (vtkAreaLayoutStrategy strategy)` - The layout strategy for producing spatial regions for the tree.
- `vtkAreaLayoutStrategy = obj.GetLayoutStrategy ()` - The layout strategy for producing spatial regions for the tree.
- `obj.SetUseRectangularCoordinates (bool rect)` - Whether the area represents radial or rectangular coordinates.
- `bool = obj.GetUseRectangularCoordinates ()` - Whether the area represents radial or rectangular coordinates.
- `obj.UseRectangularCoordinatesOn ()` - Whether the area represents radial or rectangular coordinates.
- `obj.UseRectangularCoordinatesOff ()` - Whether the area represents radial or rectangular coordinates.
- `obj.SetEdgeScalarBarVisibility (bool b)` - Visibility of scalar bar actor for edges.
- `bool = obj.GetEdgeScalarBarVisibility ()` - Visibility of scalar bar actor for edges.

40.20 vtkTreeMapView

40.20.1 Usage

vtkTreeMapView shows a vtkTree in a tree map, where each vertex in the tree is represented by a box. Child boxes are contained within the parent box, and may be colored and sized by various parameters.

To create an instance of class vtkTreeMapView, simply invoke its constructor as follows

```
obj = vtkTreeMapView
```

40.20.2 Methods

The class vtkTreeMapView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeMapView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeMapView = obj.NewInstance ()`
- `vtkTreeMapView = obj.SafeDownCast (vtkObject o)`
- `obj.SetLayoutStrategy (vtkAreaLayoutStrategy s)` - Sets the treemap layout strategy
- `obj.SetLayoutStrategy (string name)` - Sets the treemap layout strategy
- `obj.SetLayoutStrategyToBox ()` - Sets the treemap layout strategy
- `obj.SetLayoutStrategyToSliceAndDice ()` - Sets the treemap layout strategy
- `obj.SetLayoutStrategyToSquarify ()` - Sets the treemap layout strategy
- `obj.SetFontSizeRange (int maxSize, int minSize, int delta)` - The sizes of the fonts used for labeling.
- `obj.GetFontSizeRange (int range[3])` - The sizes of the fonts used for labeling.

40.21 vtkTreeRingView

40.21.1 Usage

Accepts a graph and a hierarchy - currently a tree - and provides a hierarchy-aware display. Currently, this means displaying the hierarchy using a tree ring layout, then rendering the graph vertices as leaves of the tree with curved graph edges between leaves.

.SEE ALSO vtkGraphLayoutView

.SECTION Thanks Thanks to Jason Shepherd for implementing this class

To create an instance of class vtkTreeRingView, simply invoke its constructor as follows

```
obj = vtkTreeRingView
```

40.21.2 Methods

The class vtkTreeRingView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTreeRingView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkTreeRingView = obj.NewInstance ()`
- `vtkTreeRingView = obj.SafeDownCast (vtkObject o)`
- `obj.SetRootAngles (double start, double end)` - Set the root angles for laying out the hierarchy.
- `obj.SetRootAtCenter (bool value)` - Sets whether the root is at the center or around the outside.
- `bool = obj.GetRootAtCenter ()` - Sets whether the root is at the center or around the outside.
- `obj.RootAtCenterOn ()` - Sets whether the root is at the center or around the outside.
- `obj.RootAtCenterOff ()` - Sets whether the root is at the center or around the outside.
- `obj.SetLayerThickness (double thickness)` - Set the thickness of each layer.
- `double = obj.GetLayerThickness ()` - Set the thickness of each layer.
- `obj.SetInteriorRadius (double thickness)` - Set the interior radius of the tree (i.e. the size of the "hole" in the center).
- `double = obj.GetInteriorRadius ()` - Set the interior radius of the tree (i.e. the size of the "hole" in the center).
- `obj.SetInteriorLogSpacingValue (double thickness)` - Set the log spacing factor for the invisible interior tree used for routing edges of the overlaid graph.
- `double = obj.GetInteriorLogSpacingValue ()` - Set the log spacing factor for the invisible interior tree used for routing edges of the overlaid graph.

40.22 vtkView

40.22.1 Usage

vtkView is the superclass for views. A view is generally an area of an application's canvas devoted to displaying one or more VTK data objects. Associated representations (subclasses of vtkDataRepresentation) are responsible for converting the data into a displayable format. These representations are then added to the view.

For views which display only one data object at a time you may set a data object or pipeline connection directly on the view itself (e.g. vtkGraphLayoutView, vtkLandscapeView, vtkTreeMapView). The view will internally create a vtkDataRepresentation for the data.

A view has the concept of linked selection. If the same data is displayed in multiple views, their selections may be linked by setting the same vtkAnnotationLink on their representations (see vtkDataRepresentation).

To create an instance of class vtkView, simply invoke its constructor as follows

```
obj = vtkView
```

40.22.2 Methods

The class vtkView has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkView class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkView = obj.NewInstance ()`
- `vtkView = obj.SafeDownCast (vtkObject o)`
- `obj.AddRepresentation (vtkDataRepresentation rep)` - Adds the representation to the view.
- `obj.SetRepresentation (vtkDataRepresentation rep)` - Set the representation to the view.
- `vtkDataRepresentation = obj.AddRepresentationFromInputConnection (vtkAlgorithmOutput conn)` - Convenience method which creates a simple representation with the connection and adds it to the view. Returns the representation internally created. NOTE: The returned representation pointer is not reference-counted, so you MUST call Register() on the representation if you want to keep a reference to it.
- `vtkDataRepresentation = obj.SetRepresentationFromInputConnection (vtkAlgorithmOutput conn)` - Convenience method which sets the representation with the connection and adds it to the view. Returns the representation internally created. NOTE: The returned representation pointer is not reference-counted, so you MUST call Register() on the representation if you want to keep a reference to it.
- `vtkDataRepresentation = obj.AddRepresentationFromInput (vtkDataObject input)` - Convenience method which creates a simple representation with the specified input and adds it to the view. NOTE: The returned representation pointer is not reference-counted, so you MUST call Register() on the representation if you want to keep a reference to it.
- `vtkDataRepresentation = obj.SetRepresentationFromInput (vtkDataObject input)` - Convenience method which sets the representation to the specified input and adds it to the view. NOTE: The returned representation pointer is not reference-counted, so you MUST call Register() on the representation if you want to keep a reference to it.
- `obj.RemoveRepresentation (vtkDataRepresentation rep)` - Removes the representation from the view.
- `obj.RemoveRepresentation (vtkAlgorithmOutput rep)` - Removes any representation with this connection from the view.
- `obj.RemoveAllRepresentations ()` - Removes all representations from the view.
- `int = obj.GetNumberOfRepresentations ()` - Returns the number of representations from first port(0) in this view.
- `vtkDataRepresentation = obj.GetRepresentation (int index)` - The representation at a specified index.
- `bool = obj.IsRepresentationPresent (vtkDataRepresentation rep)` - Check to see if a representation is present in the view.
- `obj.Update ()` - Update the view.

- `obj.ApplyViewTheme (vtkViewTheme )` - Meant for use by subclasses and vtkRepresentation subclasses. Call this method to register vtkObjects (generally vtkAlgorithm subclasses) which fire vtkCommand::ProgressEvent with the view. The view listens to vtkCommand::ProgressEvent and fires ViewProgressEvent with ViewProgressEventCallData containing the message and the progress amount. If message is not provided, then the class name for the algorithm is used.
- `obj.RegisterProgress (vtkObject algorithm, string messageNULL)` - Meant for use by subclasses and vtkRepresentation subclasses. Call this method to register vtkObjects (generally vtkAlgorithm subclasses) which fire vtkCommand::ProgressEvent with the view. The view listens to vtkCommand::ProgressEvent and fires ViewProgressEvent with ViewProgressEventCallData containing the message and the progress amount. If message is not provided, then the class name for the algorithm is used.
- `obj.UnRegisterProgress (vtkObject algorithm)` - Unregister objects previously registered with RegisterProgress.

40.23 vtkViewUpdater

40.23.1 Usage

vtkViewUpdater registers with annotation change events for a set of annotation links, and updates all views when an annotation link fires an annotation changed event. This is often needed when multiple views share a selection with vtkAnnotationLink.

To create an instance of class vtkViewUpdater, simply invoke its constructor as follows

```
obj = vtkViewUpdater
```

40.23.2 Methods

The class vtkViewUpdater has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkViewUpdater class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkViewUpdater = obj.NewInstance ()`
- `vtkViewUpdater = obj.SafeDownCast (vtkObject o)`
- `obj.AddView (vtkView view)`
- `obj.AddAnnotationLink (vtkAnnotationLink link)`

Chapter 41

Visualization Toolkit Volume Rendering Classes

41.1 vtkDirectionEncoder

41.1.1 Usage

Given a direction, encode it into an integer value. This value should be less than 65536, which is the maximum number of encoded directions supported by this superclass. A direction encoded is used to encode normals in a volume for use during volume rendering, and the amount of space that is allocated per normal is 2 bytes. This is an abstract superclass - see the subclasses for specific implementation details.

To create an instance of class vtkDirectionEncoder, simply invoke its constructor as follows

```
obj = vtkDirectionEncoder
```

41.1.2 Methods

The class vtkDirectionEncoder has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDirectionEncoder class.

- `string = obj.GetClassName ()` - Get the name of this class
- `int = obj.IsA (string name)` - Get the name of this class
- `vtkDirectionEncoder = obj.NewInstance ()` - Get the name of this class
- `vtkDirectionEncoder = obj.SafeDownCast (vtkObject o)` - Get the name of this class
- `int = obj.GetEncodedDirection (float n[3])` - Given a normal vector n, return the encoded direction
- `float = obj.GetDecodedGradient (int value)` - / Given an encoded value, return a pointer to the normal vector
- `int = obj.GetNumberOfEncodedDirections (void )` - Return the number of encoded directions

41.2 vtkEncodedGradientEstimator

41.2.1 Usage

vtkEncodedGradientEstimator is an abstract superclass for gradient estimation. It takes a scalar input of vtkImageData, computes a gradient value for every point, and encodes this value into a three byte value (2 for direction, 1 for magnitude) using the vtkDirectionEncoder. The direction encoder is defaulted to a vtkRecursiveSphereDirectionEncoder, but can be overridden with the SetDirectionEncoder method. The scale and the bias values for the gradient magnitude are used to convert it into a one byte value according to v = m*scale + bias where m is the magnitude and v is the resulting one byte value.

To create an instance of class vtkEncodedGradientEstimator, simply invoke its constructor as follows

```
obj = vtkEncodedGradientEstimator
```

41.2.2 Methods

The class vtkEncodedGradientEstimator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEncodedGradientEstimator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEncodedGradientEstimator = obj.NewInstance ()`
- `vtkEncodedGradientEstimator = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImageData )` - Set/Get the scalar input for which the normals will be calculated
- `vtkImageData = obj.GetInput ()` - Set/Get the scalar input for which the normals will be calculated
- `obj.SetGradientMagnitudeScale (float )` - Set/Get the scale and bias for the gradient magnitude
- `float = obj.GetGradientMagnitudeScale ()` - Set/Get the scale and bias for the gradient magnitude
- `obj.SetGradientMagnitudeBias (float )` - Set/Get the scale and bias for the gradient magnitude
- `float = obj.GetGradientMagnitudeBias ()` - Set/Get the scale and bias for the gradient magnitude
- `obj.SetBoundsClip (int )` - Turn on / off the bounding of the normal computation by the this-¿Bounds bounding box
- `int = obj.GetBoundsClipMinValue ()` - Turn on / off the bounding of the normal computation by the this-¿Bounds bounding box
- `int = obj.GetBoundsClipMaxValue ()` - Turn on / off the bounding of the normal computation by the this-¿Bounds bounding box
- `int = obj.GetBoundsClip ()` - Turn on / off the bounding of the normal computation by the this-¿Bounds bounding box
- `obj.BoundsClipOn ()` - Turn on / off the bounding of the normal computation by the this-¿Bounds bounding box
- `obj.BoundsClipOff ()` - Turn on / off the bounding of the normal computation by the this-¿Bounds bounding box

- `obj.SetBounds (int , int , int , int , int , int )` - Set / Get the bounds of the computation (used if this-¿ComputationBounds is 1.) The bounds are specified xmin, xmax, ymin, ymax, zmin, zmax.
- `obj.SetBounds (int a[6])` - Set / Get the bounds of the computation (used if this-¿ComputationBounds is 1.) The bounds are specified xmin, xmax, ymin, ymax, zmin, zmax.
- `int = obj. GetBounds ()` - Set / Get the bounds of the computation (used if this-¿ComputationBounds is 1.) The bounds are specified xmin, xmax, ymin, ymax, zmin, zmax.
- `obj.Update (void )` - Recompute the encoded normals and gradient magnitudes.
- `int = obj.GetEncodedNormalIndex (int xyz\_index)` - Get the encoded normal at an x,y,z location in the volume
- `int = obj.GetEncodedNormalIndex (int x\_index, int y\_index, int z\_index)` - Get the encoded normal at an x,y,z location in the volume
- `obj.SetNumberOfThreads (int )` - Get/Set the number of threads to create when encoding normals This defaults to the number of available processors on the machine
- `int = obj.GetNumberOfThreadsMinValue ()` - Get/Set the number of threads to create when encoding normals This defaults to the number of available processors on the machine
- `int = obj.GetNumberOfThreadsMaxValue ()` - Get/Set the number of threads to create when encoding normals This defaults to the number of available processors on the machine
- `int = obj.GetNumberOfThreads ()` - Get/Set the number of threads to create when encoding normals This defaults to the number of available processors on the machine
- `obj.SetDirectionEncoder (vtkDirectionEncoder direnc)` - Set / Get the direction encoder used to encode normal directions to fit within two bytes
- `vtkDirectionEncoder = obj.GetDirectionEncoder ()` - Set / Get the direction encoder used to encode normal directions to fit within two bytes
- `obj.SetComputeGradientMagnitudes (int )` - If you don't want to compute gradient magnitudes (but you do want normals for shading) this can be used. Be careful - if if you a non-constant gradient magnitude transfer function and you turn this on, it may crash
- `int = obj.GetComputeGradientMagnitudes ()` - If you don't want to compute gradient magnitudes (but you do want normals for shading) this can be used. Be careful - if if you a non-constant gradient magnitude transfer function and you turn this on, it may crash
- `obj.ComputeGradientMagnitudesOn ()` - If you don't want to compute gradient magnitudes (but you do want normals for shading) this can be used. Be careful - if if you a non-constant gradient magnitude transfer function and you turn this on, it may crash
- `obj.ComputeGradientMagnitudesOff ()` - If you don't want to compute gradient magnitudes (but you do want normals for shading) this can be used. Be careful - if if you a non-constant gradient magnitude transfer function and you turn this on, it may crash
- `obj.SetCylinderClip (int )` - If the data in each slice is only contained within a circle circumscribed within the slice, and the slice is square, then don't compute anything outside the circle. This circle through the slices forms a cylinder.
- `int = obj.GetCylinderClip ()` - If the data in each slice is only contained within a circle circumscribed within the slice, and the slice is square, then don't compute anything outside the circle. This circle through the slices forms a cylinder.

- `obj.CylinderClipOn ()` - If the data in each slice is only contained within a circle circumscribed within the slice, and the slice is square, then don't compute anything outside the circle. This circle through the slices forms a cylinder.
- `obj.CylinderClipOff ()` - If the data in each slice is only contained within a circle circumscribed within the slice, and the slice is square, then don't compute anything outside the circle. This circle through the slices forms a cylinder.
- `float = obj.GetLastUpdateTimeInSeconds ()` - Get the time required for the last update in seconds or cpu seconds
- `float = obj.GetLastUpdateTimeInCPUSeconds ()` - Get the time required for the last update in seconds or cpu seconds
- `int = obj.GetUseCylinderClip ()`
- `obj.SetZeroNormalThreshold (float v)` - Set / Get the ZeroNormalThreshold - this defines the minimum magnitude of a gradient that is considered sufficient to define a direction. Gradients with magnitudes at or less than this value are given a "zero normal" index. These are handled specially in the shader, and you can set the intensity of light for these zero normals in the gradient shader.
- `float = obj.GetZeroNormalThreshold ()` - Set / Get the ZeroNormalThreshold - this defines the minimum magnitude of a gradient that is considered sufficient to define a direction. Gradients with magnitudes at or less than this value are given a "zero normal" index. These are handled specially in the shader, and you can set the intensity of light for these zero normals in the gradient shader.
- `obj.SetZeroPad (int )` - Assume that the data value outside the volume is zero when computing normals.
- `int = obj.GetZeroPadMinValue ()` - Assume that the data value outside the volume is zero when computing normals.
- `int = obj.GetZeroPadMaxValue ()` - Assume that the data value outside the volume is zero when computing normals.
- `int = obj.GetZeroPad ()` - Assume that the data value outside the volume is zero when computing normals.
- `obj.ZeroPadOn ()` - Assume that the data value outside the volume is zero when computing normals.
- `obj.ZeroPadOff ()` - Assume that the data value outside the volume is zero when computing normals.

41.3 vtkEncodedGradientShader

41.3.1 Usage

vtkEncodedGradientShader computes shading tables for encoded normals that indicates the amount of diffuse and specular illumination that is received from all light sources at a surface location with that normal. For diffuse illumination this is accurate, but for specular illumination it is approximate for perspective projections since the center view direction is always used as the view direction. Since the shading table is dependent on the volume (for the transformation that must be applied to the normals to put them into world coordinates) there is a shading table per volume. This is necessary because multiple volumes can share a volume mapper.

To create an instance of class vtkEncodedGradientShader, simply invoke its constructor as follows

```
obj = vtkEncodedGradientShader
```

41.3.2 Methods

The class vtkEncodedGradientShader has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEncodedGradientShader class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkEncodedGradientShader = obj.NewInstance ()`
- `vtkEncodedGradientShader = obj.SafeDownCast (vtkObject o)`
- `obj.SetZeroNormalDiffuseIntensity (float )` - Set / Get the intensity diffuse / specular light used for the zero normals.
- `float = obj.GetZeroNormalDiffuseIntensityMinValue ()` - Set / Get the intensity diffuse / specular light used for the zero normals.
- `float = obj.GetZeroNormalDiffuseIntensityMaxValue ()` - Set / Get the intensity diffuse / specular light used for the zero normals.
- `float = obj.GetZeroNormalDiffuseIntensity ()` - Set / Get the intensity diffuse / specular light used for the zero normals.
- `obj.SetZeroNormalSpecularIntensity (float )` - Set / Get the intensity diffuse / specular light used for the zero normals.
- `float = obj.GetZeroNormalSpecularIntensityMinValue ()` - Set / Get the intensity diffuse / specular light used for the zero normals.
- `float = obj.GetZeroNormalSpecularIntensityMaxValue ()` - Set / Get the intensity diffuse / specular light used for the zero normals.
- `float = obj.GetZeroNormalSpecularIntensity ()` - Set / Get the intensity diffuse / specular light used for the zero normals.
- `obj.UpdateShadingTable (vtkRenderer ren, vtkVolume vol, vtkEncodedGradientEstimator gradest)` - Cause the shading table to be updated
- `obj.SetActiveComponent (int )` - Set the active component for shading. This component's ambient / diffuse / specular / specular power values will be used to create the shading table. The default is 1.0
- `int = obj.GetActiveComponentMinValue ()` - Set the active component for shading. This component's ambient / diffuse / specular / specular power values will be used to create the shading table. The default is 1.0
- `int = obj.GetActiveComponentMaxValue ()` - Set the active component for shading. This component's ambient / diffuse / specular / specular power values will be used to create the shading table. The default is 1.0
- `int = obj.GetActiveComponent ()` - Set the active component for shading. This component's ambient / diffuse / specular / specular power values will be used to create the shading table. The default is 1.0

41.4 vtkFiniteDifferenceGradientEstimator

41.4.1 Usage

vtkFiniteDifferenceGradientEstimator is a concrete subclass of vtkEncodedGradientEstimator that uses a central differences technique to estimate the gradient. The gradient at some sample location (x,y,z) would be estimated by:

nx = (f(x-dx,y,z) - f(x+dx,y,z)) / 2*dx; ny = (f(x,y-dy,z) - f(x,y+dy,z)) / 2*dy; nz = (f(x,y,z-dz) - f(x,y,z+dz)) / 2*dz;

This value is normalized to determine a unit direction vector and a magnitude. The normal is computed in voxel space, and dx = dy = dz = SampleSpacingInVoxels. A scaling factor is applied to convert this normal from voxel space to world coordinates.

To create an instance of class vtkFiniteDifferenceGradientEstimator, simply invoke its constructor as follows

```
obj = vtkFiniteDifferenceGradientEstimator
```

41.4.2 Methods

The class vtkFiniteDifferenceGradientEstimator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFiniteDifferenceGradientEstimator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFiniteDifferenceGradientEstimator = obj.NewInstance ()`
- `vtkFiniteDifferenceGradientEstimator = obj.SafeDownCast (vtkObject o)`
- `obj.SetSampleSpacingInVoxels (int )` - Set/Get the spacing between samples for the finite differences method used to compute the normal. This spacing is in voxel units.
- `int = obj.GetSampleSpacingInVoxels ()` - Set/Get the spacing between samples for the finite differences method used to compute the normal. This spacing is in voxel units.

41.5 vtkFixedPointRayCastImage

41.5.1 Usage

This is a helper class for storing the ray cast image including the underlying data and the size of the image. This class is not intended to be used directly - just as an internal class in the vtkFixedPointVolumeRayCastMapper so that multiple mappers can share the same image. This class also stored the ZBuffer (if necessary due to intermixed geometry). Perhaps this class could be generalized in the future to be used for other ray cast methods other than the fixed point method.

To create an instance of class vtkFixedPointRayCastImage, simply invoke its constructor as follows

```
obj = vtkFixedPointRayCastImage
```

41.5.2 Methods

The class vtkFixedPointRayCastImage has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFixedPointRayCastImage class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFixedPointRayCastImage = obj.NewInstance ()`
- `vtkFixedPointRayCastImage = obj.SafeDownCast (vtkObject o)`
- `obj.SetImageViewportSize (int , int )` - Set / Get the ImageViewportSize. This is the size of the whole viewport in pixels.
- `obj.SetImageViewportSize (int a[2])` - Set / Get the ImageViewportSize. This is the size of the whole viewport in pixels.
- `int = obj. GetImageViewportSize ()` - Set / Get the ImageViewportSize. This is the size of the whole viewport in pixels.
- `obj.SetImageMemorySize (int , int )` - Set / Get the ImageMemorySize. This is the size in pixels of the Image ivar. This will be a power of two in order to ensure that the texture can be rendered by graphics hardware that requires power of two textures.
- `obj.SetImageMemorySize (int a[2])` - Set / Get the ImageMemorySize. This is the size in pixels of the Image ivar. This will be a power of two in order to ensure that the texture can be rendered by graphics hardware that requires power of two textures.
- `int = obj. GetImageMemorySize ()` - Set / Get the ImageMemorySize. This is the size in pixels of the Image ivar. This will be a power of two in order to ensure that the texture can be rendered by graphics hardware that requires power of two textures.
- `obj.SetImageInUseSize (int , int )` - Set / Get the size of the image we are actually using. As long as the memory size is big enough, but not too big, we won't bother deleting and re-allocated, we'll just continue to use the memory size we have. This size will always be equal to or less than the ImageMemorySize.
- `obj.SetImageInUseSize (int a[2])` - Set / Get the size of the image we are actually using. As long as the memory size is big enough, but not too big, we won't bother deleting and re-allocated, we'll just continue to use the memory size we have. This size will always be equal to or less than the ImageMemorySize.
- `int = obj. GetImageInUseSize ()` - Set / Get the size of the image we are actually using. As long as the memory size is big enough, but not too big, we won't bother deleting and re-allocated, we'll just continue to use the memory size we have. This size will always be equal to or less than the ImageMemorySize.
- `obj.SetImageOrigin (int , int )` - Set / Get the origin of the image. This is the starting pixel within the whole viewport that our Image starts on. That is, we could be generating just a subregion of the whole viewport due to the fact that our volume occupies only a portion of the viewport. The Image pixels will start from this location.
- `obj.SetImageOrigin (int a[2])` - Set / Get the origin of the image. This is the starting pixel within the whole viewport that our Image starts on. That is, we could be generating just a subregion of the whole viewport due to the fact that our volume occupies only a portion of the viewport. The Image pixels will start from this location.
- `int = obj. GetImageOrigin ()` - Set / Get the origin of the image. This is the starting pixel within the whole viewport that our Image starts on. That is, we could be generating just a subregion of the whole viewport due to the fact that our volume occupies only a portion of the viewport. The Image pixels will start from this location.

- `obj.SetImageSampleDistance (float )` - Set / Get the ImageSampleDistance that will be used for rendering. This is a copy of the value stored in the mapper. It is stored here for sharing between all mappers that are participating in the creation of this image.

- `float = obj.GetImageSampleDistance ()` - Set / Get the ImageSampleDistance that will be used for rendering. This is a copy of the value stored in the mapper. It is stored here for sharing between all mappers that are participating in the creation of this image.

- `obj.AllocateImage ()` - Call this method once the ImageMemorySize has been set the allocate the image. If an image already exists, it will be deleted first.

- `obj.ClearImage ()` - Clear the image to (0,0,0,0) for each pixel.

- `obj.SetZBufferSize (int , int )` - Set / Get the size of the ZBuffer in pixels. The zbuffer will be captured for the region of the screen covered by the ImageInUseSize image. However, due to subsampling, the size of the ImageInUseSize image may be smaller than this ZBuffer image which will be captured at screen resolution.

- `obj.SetZBufferSize (int  a[2])` - Set / Get the size of the ZBuffer in pixels. The zbuffer will be captured for the region of the screen covered by the ImageInUseSize image. However, due to subsampling, the size of the ImageInUseSize image may be smaller than this ZBuffer image which will be captured at screen resolution.

- `int = obj. GetZBufferSize ()` - Set / Get the size of the ZBuffer in pixels. The zbuffer will be captured for the region of the screen covered by the ImageInUseSize image. However, due to subsampling, the size of the ImageInUseSize image may be smaller than this ZBuffer image which will be captured at screen resolution.

- `obj.SetZBufferOrigin (int , int )` - Set / Get the origin of the ZBuffer. This is the distance from the lower left corner of the viewport where the ZBuffer started (multiply the ImageOrigin by the ImageSampleDistance) This is the pixel location on the full resolution viewport where the ZBuffer capture will start. These values are used to convert the (x,y) pixel location within the ImageInUseSize image into a ZBuffer location.

- `obj.SetZBufferOrigin (int  a[2])` - Set / Get the origin of the ZBuffer. This is the distance from the lower left corner of the viewport where the ZBuffer started (multiply the ImageOrigin by the ImageSampleDistance) This is the pixel location on the full resolution viewport where the ZBuffer capture will start. These values are used to convert the (x,y) pixel location within the ImageInUseSize image into a ZBuffer location.

- `int = obj. GetZBufferOrigin ()` - Set / Get the origin of the ZBuffer. This is the distance from the lower left corner of the viewport where the ZBuffer started (multiply the ImageOrigin by the ImageSampleDistance) This is the pixel location on the full resolution viewport where the ZBuffer capture will start. These values are used to convert the (x,y) pixel location within the ImageInUseSize image into a ZBuffer location.

- `obj.SetUseZBuffer (int )` - The UseZBuffer flag indicates whether the ZBuffer is in use. The ZBuffer is captured and used when IntermixIntersectingGeometry is on in the mapper, and when there are props that have been rendered before the current volume.

- `int = obj.GetUseZBufferMinValue ()` - The UseZBuffer flag indicates whether the ZBuffer is in use. The ZBuffer is captured and used when IntermixIntersectingGeometry is on in the mapper, and when there are props that have been rendered before the current volume.

- `int = obj.GetUseZBufferMaxValue ()` - The UseZBuffer flag indicates whether the ZBuffer is in use. The ZBuffer is captured and used when IntermixIntersectingGeometry is on in the mapper, and when there are props that have been rendered before the current volume.

- `int = obj.GetUseZBuffer ()` - The UseZBuffer flag indicates whether the ZBuffer is in use. The ZBuffer is captured and used when IntermixIntersectingGeometry is on in the mapper, and when there are props that have been rendered before the current volume.
- `obj.UseZBufferOn ()` - The UseZBuffer flag indicates whether the ZBuffer is in use. The ZBuffer is captured and used when IntermixIntersectingGeometry is on in the mapper, and when there are props that have been rendered before the current volume.
- `obj.UseZBufferOff ()` - The UseZBuffer flag indicates whether the ZBuffer is in use. The ZBuffer is captured and used when IntermixIntersectingGeometry is on in the mapper, and when there are props that have been rendered before the current volume.
- `float = obj.GetZBufferValue (int x, int y)` - Get the ZBuffer value corresponding to location (x,y) where (x,y) are indexing into the ImageInUse image. This must be converted to the zbuffer image coordinates. Nearest neighbor value is returned. If UseZBuffer is off, then 1.0 is always returned.
- `obj.AllocateZBuffer ()`

41.6 vtkFixedPointVolumeRayCastCompositeGOHelper

41.6.1 Usage

This is one of the helper classes for the vtkFixedPointVolumeRayCastMapper. It will generate composite images using an alpha blending operation. This class should not be used directly, it is a helper class for the mapper and has no user-level API.

To create an instance of class vtkFixedPointVolumeRayCastCompositeGOHelper, simply invoke its constructor as follows

```
obj = vtkFixedPointVolumeRayCastCompositeGOHelper
```

41.6.2 Methods

The class vtkFixedPointVolumeRayCastCompositeGOHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFixedPointVolumeRayCastCompositeGOHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFixedPointVolumeRayCastCompositeGOHelper = obj.NewInstance ()`
- `vtkFixedPointVolumeRayCastCompositeGOHelper = obj.SafeDownCast (vtkObject o)`
- `obj.GenerateImage (int threadID, int threadCount, vtkVolume vol, vtkFixedPointVolumeRayCastMapper m`

41.7 vtkFixedPointVolumeRayCastCompositeGOShadeHelper

41.7.1 Usage

This is one of the helper classes for the vtkFixedPointVolumeRayCastMapper. It will generate composite images using an alpha blending operation. This class should not be used directly, it is a helper class for the mapper and has no user-level API.

To create an instance of class vtkFixedPointVolumeRayCastCompositeGOShadeHelper, simply invoke its constructor as follows

```
obj = vtkFixedPointVolumeRayCastCompositeGOShadeHelper
```

41.7.2 Methods

The class vtkFixedPointVolumeRayCastCompositeGOShadeHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFixedPointVolumeRayCastCompositeGOShadeHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFixedPointVolumeRayCastCompositeGOShadeHelper = obj.NewInstance ()`
- `vtkFixedPointVolumeRayCastCompositeGOShadeHelper = obj.SafeDownCast (vtkObject o)`
- `obj.GenerateImage (int threadID, int threadCount, vtkVolume vol, vtkFixedPointVolumeRayCastMapper m`

41.8 vtkFixedPointVolumeRayCastCompositeHelper

41.8.1 Usage

This is one of the helper classes for the vtkFixedPointVolumeRayCastMapper. It will generate composite images using an alpha blending operation. This class should not be used directly, it is a helper class for the mapper and has no user-level API.

To create an instance of class vtkFixedPointVolumeRayCastCompositeHelper, simply invoke its constructor as follows

```
obj = vtkFixedPointVolumeRayCastCompositeHelper
```

41.8.2 Methods

The class vtkFixedPointVolumeRayCastCompositeHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFixedPointVolumeRayCastCompositeHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFixedPointVolumeRayCastCompositeHelper = obj.NewInstance ()`
- `vtkFixedPointVolumeRayCastCompositeHelper = obj.SafeDownCast (vtkObject o)`
- `obj.GenerateImage (int threadID, int threadCount, vtkVolume vol, vtkFixedPointVolumeRayCastMapper m`

41.9 vtkFixedPointVolumeRayCastCompositeShadeHelper

41.9.1 Usage

This is one of the helper classes for the vtkFixedPointVolumeRayCastMapper. It will generate composite images using an alpha blending operation. This class should not be used directly, it is a helper class for the mapper and has no user-level API.

To create an instance of class vtkFixedPointVolumeRayCastCompositeShadeHelper, simply invoke its constructor as follows

```
obj = vtkFixedPointVolumeRayCastCompositeShadeHelper
```

41.9.2 Methods

The class vtkFixedPointVolumeRayCastCompositeShadeHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFixedPointVolumeRayCastCompositeShadeHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFixedPointVolumeRayCastCompositeShadeHelper = obj.NewInstance ()`
- `vtkFixedPointVolumeRayCastCompositeShadeHelper = obj.SafeDownCast (vtkObject o)`
- `obj.GenerateImage (int threadID, int threadCount, vtkVolume vol, vtkFixedPointVolumeRayCastMapper m`

41.10 vtkFixedPointVolumeRayCastHelper

41.10.1 Usage

This is the abstract superclass of all helper classes for the vtkFixedPointVolumeRayCastMapper. This class should not be used directly.

To create an instance of class vtkFixedPointVolumeRayCastHelper, simply invoke its constructor as follows

```
obj = vtkFixedPointVolumeRayCastHelper
```

41.10.2 Methods

The class vtkFixedPointVolumeRayCastHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFixedPointVolumeRayCastHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFixedPointVolumeRayCastHelper = obj.NewInstance ()`
- `vtkFixedPointVolumeRayCastHelper = obj.SafeDownCast (vtkObject o)`
- `obj.GenerateImage (int , int , vtkVolume , vtkFixedPointVolumeRayCastMapper )`

41.11 vtkFixedPointVolumeRayCastMapper

41.11.1 Usage

This is a software ray caster for rendering volumes in vtkImageData. It works with all input data types and up to four components. It performs composite or MIP rendering, and can be intermixed with geometric data. Space leaping is used to speed up the rendering process. In addition, calculation are performed in 15 bit fixed point precision. This mapper is threaded, and will interleave scan lines across processors.

This mapper is a good replacement for vtkVolumeRayCastMapper EXCEPT: - it does not do isosurface ray casting - it does only interpolate before classify compositing - it does only maximum scalar value MIP

The vtkVolumeRayCastMapper CANNOT be used in these instances when a vtkFixedPointVolumeRay-CastMapper can be used: - if the data is not unsigned char or unsigned short - if the data has more than one component

This mapper handles all data type from unsigned char through double. However, some of the internal calcultions are performed in float and therefore even the full float range may cause problems for this mapper (both in scalar data values and in spacing between samples).

Space leaping is performed by creating a sub-sampled volume. 4x4x4 cells in the original volume are represented by a min, max, and combined gradient and flag value. The min max volume has three unsigned shorts per 4x4x4 group of cells from the original volume - one reprenting the minumum scalar index (the scalar value adjusted to fit in the 15 bit range), the maximum scalar index, and a third unsigned short which is both the maximum gradient opacity in the neighborhood (an unsigned char) and the flag that is filled in for the current lookup tables to indicate whether this region can be skipped.

To create an instance of class vtkFixedPointVolumeRayCastMapper, simply invoke its constructor as follows

```
obj = vtkFixedPointVolumeRayCastMapper
```

41.11.2 Methods

The class vtkFixedPointVolumeRayCastMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFixedPointVolumeRayCastMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFixedPointVolumeRayCastMapper = obj.NewInstance ()`
- `vtkFixedPointVolumeRayCastMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetSampleDistance (float )` - Set/Get the distance between samples used for rendering when AutoAdjustSampleDistances is off, or when this mapper has more than 1 second allocated to it for rendering.
- `float = obj.GetSampleDistance ()` - Set/Get the distance between samples used for rendering when AutoAdjustSampleDistances is off, or when this mapper has more than 1 second allocated to it for rendering.
- `obj.SetInteractiveSampleDistance (float )` - Set/Get the distance between samples when interactive rendering is happening. In this case, interactive is defined as this volume mapper having less than 1 second allocated for rendering. When AutoAdjustSampleDistance is On, and the allocated render time is less than 1 second, then this InteractiveSampleDistance will be used instead of the SampleDistance above.
- `float = obj.GetInteractiveSampleDistance ()` - Set/Get the distance between samples when interactive rendering is happening. In this case, interactive is defined as this volume mapper having less than 1 second allocated for rendering. When AutoAdjustSampleDistance is On, and the allocated render time is less than 1 second, then this InteractiveSampleDistance will be used instead of the SampleDistance above.
- `obj.SetImageSampleDistance (float )` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels. This value will be adjusted to meet a desired frame rate when AutoAdjustSampleDistances is on.
- `float = obj.GetImageSampleDistanceMinValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels. This value will be adjusted to meet a desired frame rate when AutoAdjustSampleDistances is on.

- `float = obj.GetImageSampleDistanceMaxValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels. This value will be adjusted to meet a desired frame rate when AutoAdjustSampleDistances is on.

- `float = obj.GetImageSampleDistance ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels. This value will be adjusted to meet a desired frame rate when AutoAdjustSampleDistances is on.

- `obj.SetMinimumImageSampleDistance (float )` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted.

- `float = obj.GetMinimumImageSampleDistanceMinValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted.

- `float = obj.GetMinimumImageSampleDistanceMaxValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted.

- `float = obj.GetMinimumImageSampleDistance ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted.

- `obj.SetMaximumImageSampleDistance (float )` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted.

- `float = obj.GetMaximumImageSampleDistanceMinValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted.

- `float = obj.GetMaximumImageSampleDistanceMaxValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted.

- `float = obj.GetMaximumImageSampleDistance ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted.

- `obj.SetAutoAdjustSampleDistances (int )` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance and the SampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use). If this is an interactive render (more than 1 frame per second) the SampleDistance will be increased, otherwise it will not be altered (a binary decision, as opposed to the ImageSampleDistance which will vary continuously).

- `int = obj.GetAutoAdjustSampleDistancesMinValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance and the SampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use). If this is an interactive render (more than 1 frame per second) the SampleDistance will be increased, otherwise it will not be altered (a binary decision, as opposed to the ImageSampleDistance which will vary continuously).

- `int = obj.GetAutoAdjustSampleDistancesMaxValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance and the SampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use). If this is an interactive render (more than 1 frame per second) the SampleDistance will be increased, otherwise it will not be altered (a binary decision, as opposed to the ImageSampleDistance which will vary continuously).

- `int = obj.GetAutoAdjustSampleDistances ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance and the SampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use). If this is an interactive render (more than 1 frame per second) the SampleDistance will be increased, otherwise it will not be altered (a binary decision, as opposed to the ImageSampleDistance which will vary continuously).

- `obj.AutoAdjustSampleDistancesOn ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance and the SampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use). If this is an interactive render (more than 1 frame per second) the SampleDistance will be increased, otherwise it will not be altered (a binary decision, as opposed to the ImageSampleDistance which will vary continuously).

- `obj.AutoAdjustSampleDistancesOff ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance and the SampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use). If this is an interactive render (more than 1 frame per second) the SampleDistance will be increased, otherwise it will not be altered (a binary decision, as opposed to the ImageSampleDistance which will vary continuously).

- `obj.SetLockSampleDistanceToInputSpacing (int )` - Automatically compute the sample distance from the data spacing. When the number of voxels is 8, the sample distance will be roughly 1/200 the average voxel size. The distance will grow proportionally to numVoxels$\hat{}$(1/3) until it reaches 1/2 average voxel size when number of voxels is 1E6. Note that ScalarOpacityUnitDistance is still taken into account and if different than 1, will effect the sample distance.

- `int = obj.GetLockSampleDistanceToInputSpacingMinValue ()` - Automatically compute the sample distance from the data spacing. When the number of voxels is 8, the sample distance will be roughly 1/200 the average voxel size. The distance will grow proportionally to numVoxels$\hat{}$(1/3) until it reaches 1/2 average voxel size when number of voxels is 1E6. Note that ScalarOpacityUnitDistance is still taken into account and if different than 1, will effect the sample distance.

- `int = obj.GetLockSampleDistanceToInputSpacingMaxValue ()` - Automatically compute the sample distance from the data spacing. When the number of voxels is 8, the sample distance will be roughly 1/200 the average voxel size. The distance will grow proportionally to numVoxels$\hat{}$(1/3) until it reaches 1/2 average voxel size when number of voxels is 1E6. Note that ScalarOpacityUnitDistance is still taken into account and if different than 1, will effect the sample distance.

- `int = obj.GetLockSampleDistanceToInputSpacing ()` - Automatically compute the sample distance from the data spacing. When the number of voxels is 8, the sample distance will be roughly 1/200 the average voxel size. The distance will grow proportionally to numVoxels$\hat{}$(1/3) until it reaches 1/2 average voxel size when number of voxels is 1E6. Note that ScalarOpacityUnitDistance is still taken into account and if different than 1, will effect the sample distance.

- `obj.LockSampleDistanceToInputSpacingOn ()` - Automatically compute the sample distance from the data spacing. When the number of voxels is 8, the sample distance will be roughly 1/200 the average voxel size. The distance will grow proportionally to numVoxels$\hat{}$(1/3) until it reaches 1/2 average voxel size when number of voxels is 1E6. Note that ScalarOpacityUnitDistance is still taken into account and if different than 1, will effect the sample distance.

- `obj.LockSampleDistanceToInputSpacingOff ()` - Automatically compute the sample distance from the data spacing. When the number of voxels is 8, the sample distance will be roughly 1/200 the average voxel size. The distance will grow proportionally to numVoxels$\hat{}$(1/3) until it reaches 1/2 average voxel size when number of voxels is 1E6. Note that ScalarOpacityUnitDistance is still taken into account and if different than 1, will effect the sample distance.

- `obj.SetNumberOfThreads (int num)` - Set/Get the number of threads to use. This by default is equal to the number of available processors detected.

- `int = obj.GetNumberOfThreads ()` - Set/Get the number of threads to use. This by default is equal to the number of available processors detected.

- `obj.SetIntermixIntersectingGeometry (int )` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.

- `int = obj.GetIntermixIntersectingGeometryMinValue ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `int = obj.GetIntermixIntersectingGeometryMaxValue ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `int = obj.GetIntermixIntersectingGeometry ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `obj.IntermixIntersectingGeometryOn ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `obj.IntermixIntersectingGeometryOff ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `float = obj.ComputeRequiredImageSampleDistance (float desiredTime, vtkRenderer ren)` - What is the image sample distance required to achieve the desired time? A version of this method is provided that does not require the volume argument since if you are using an LODProp3D you may not know this information. If you use this version you must be certain that the ray cast mapper is only used for one volume (and not shared among multiple volumes)
- `float = obj.ComputeRequiredImageSampleDistance (float desiredTime, vtkRenderer ren, vtkVolume vol)` - What is the image sample distance required to achieve the desired time? A version of this method is provided that does not require the volume argument since if you are using an LODProp3D you may not know this information. If you use this version you must be certain that the ray cast mapper is only used for one volume (and not shared among multiple volumes)
- `vtkRenderWindow = obj.GetRenderWindow ()`
- `vtkFixedPointVolumeRayCastMIPHelper = obj.GetMIPHelper ()`
- `vtkFixedPointVolumeRayCastCompositeHelper = obj.GetCompositeHelper ()`
- `vtkFixedPointVolumeRayCastCompositeGOHelper = obj.GetCompositeGOHelper ()`
- `vtkFixedPointVolumeRayCastCompositeGOShadeHelper = obj.GetCompositeGOShadeHelper ()`
- `vtkFixedPointVolumeRayCastCompositeShadeHelper = obj.GetCompositeShadeHelper ()`
- `float = obj. GetTableShift ()`
- `float = obj. GetTableScale ()`
- `int = obj.GetShadingRequired ()`
- `int = obj.GetGradientOpacityRequired ()`
- `vtkDataArray = obj.GetCurrentScalars ()`
- `vtkDataArray = obj.GetPreviousScalars ()`
- `vtkVolume = obj.GetVolume ()`
- `obj.ComputeRayInfo (int x, int y, int pos[3], int dir[3], int numSteps)`
- `obj.InitializeRayInfo (vtkVolume vol)`
- `int = obj.ShouldUseNearestNeighborInterpolation (vtkVolume vol)`
- `obj.SetRayCastImage (vtkFixedPointRayCastImage )` - Set / Get the underlying image object. One will be automatically created - only need to set it when using from an AMR mapper which renders multiple times into the same image.

- `vtkFixedPointRayCastImage = obj.GetRayCastImage ()` - Set / Get the underlying image object. One will be automatically created - only need to set it when using from an AMR mapper which renders multiple times into the same image.

- `int = obj.PerImageInitialization (vtkRenderer , vtkVolume , int , double , double , int )`

- `obj.PerVolumeInitialization (vtkRenderer , vtkVolume )`

- `obj.PerSubVolumeInitialization (vtkRenderer , vtkVolume , int )`

- `obj.RenderSubVolume ()`

- `obj.DisplayRenderedImage (vtkRenderer , vtkVolume )`

- `obj.AbortRender ()`

- `obj.CreateCanonicalView (vtkVolume volume, vtkImageData image, int blend\_mode, double viewDirectio`

- `float = obj.GetEstimatedRenderTime (vtkRenderer ren, vtkVolume vol)` - Get an estimate of the rendering time for a given volume / renderer. Only valid if this mapper has been used to render that volume for that renderer previously. Estimate is good when the viewing parameters have not changed much since that last render.

- `float = obj.GetEstimatedRenderTime (vtkRenderer ren)` - Set/Get the window / level applied to the final color. This allows brightness / contrast adjustments on the final image. window is the width of the window. level is the center of the window. Initial window value is 1.0 Initial level value is 0.5 window cannot be null but can be negative, this way values will be reversed. —window— can be larger than 1.0 level can be any real value.

- `obj.SetFinalColorWindow (float )` - Set/Get the window / level applied to the final color. This allows brightness / contrast adjustments on the final image. window is the width of the window. level is the center of the window. Initial window value is 1.0 Initial level value is 0.5 window cannot be null but can be negative, this way values will be reversed. —window— can be larger than 1.0 level can be any real value.

- `float = obj.GetFinalColorWindow ()` - Set/Get the window / level applied to the final color. This allows brightness / contrast adjustments on the final image. window is the width of the window. level is the center of the window. Initial window value is 1.0 Initial level value is 0.5 window cannot be null but can be negative, this way values will be reversed. —window— can be larger than 1.0 level can be any real value.

- `obj.SetFinalColorLevel (float )` - Set/Get the window / level applied to the final color. This allows brightness / contrast adjustments on the final image. window is the width of the window. level is the center of the window. Initial window value is 1.0 Initial level value is 0.5 window cannot be null but can be negative, this way values will be reversed. —window— can be larger than 1.0 level can be any real value.

- `float = obj.GetFinalColorLevel ()` - Set/Get the window / level applied to the final color. This allows brightness / contrast adjustments on the final image. window is the width of the window. level is the center of the window. Initial window value is 1.0 Initial level value is 0.5 window cannot be null but can be negative, this way values will be reversed. —window— can be larger than 1.0 level can be any real value.

- `int = obj.GetFlipMIPComparison ()`

41.12 vtkFixedPointVolumeRayCastMIPHelper

41.12.1 Usage

This is one of the helper classes for the vtkFixedPointVolumeRayCastMapper. It will generate maximum intensity images. This class should not be used directly, it is a helper class for the mapper and has no user-level API.

To create an instance of class vtkFixedPointVolumeRayCastMIPHelper, simply invoke its constructor as follows

```
obj = vtkFixedPointVolumeRayCastMIPHelper
```

41.12.2 Methods

The class vtkFixedPointVolumeRayCastMIPHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFixedPointVolumeRayCastMIPHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkFixedPointVolumeRayCastMIPHelper = obj.NewInstance ()`
- `vtkFixedPointVolumeRayCastMIPHelper = obj.SafeDownCast (vtkObject o)`
- `obj.GenerateImage (int threadID, int threadCount, vtkVolume vol, vtkFixedPointVolumeRayCastMapper m`

41.13 vtkGPUVolumeRayCastMapper

41.13.1 Usage

vtkGPUVolumeRayCastMapper is a volume mapper that performs ray casting on the GPU using fragment programs.

To create an instance of class vtkGPUVolumeRayCastMapper, simply invoke its constructor as follows

```
obj = vtkGPUVolumeRayCastMapper
```

41.13.2 Methods

The class vtkGPUVolumeRayCastMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkGPUVolumeRayCastMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkGPUVolumeRayCastMapper = obj.NewInstance ()`
- `vtkGPUVolumeRayCastMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetAutoAdjustSampleDistances (int )` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).

- `int = obj.GetAutoAdjustSampleDistancesMinValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `int = obj.GetAutoAdjustSampleDistancesMaxValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `int = obj.GetAutoAdjustSampleDistances ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `obj.AutoAdjustSampleDistancesOn ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `obj.AutoAdjustSampleDistancesOff ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `obj.SetSampleDistance (float )` - Set/Get the distance between samples used for rendering when AutoAdjustSampleDistances is off, or when this mapper has more than 1 second allocated to it for rendering. Initial value is 1.0.
- `float = obj.GetSampleDistance ()` - Set/Get the distance between samples used for rendering when AutoAdjustSampleDistances is off, or when this mapper has more than 1 second allocated to it for rendering. Initial value is 1.0.
- `obj.SetImageSampleDistance (float )` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels. This value will be adjusted to meet a desired frame rate when AutoAdjustSampleDistances is on.
- `float = obj.GetImageSampleDistanceMinValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels. This value will be adjusted to meet a desired frame rate when AutoAdjustSampleDistances is on.
- `float = obj.GetImageSampleDistanceMaxValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels. This value will be adjusted to meet a desired frame rate when AutoAdjustSampleDistances is on.
- `float = obj.GetImageSampleDistance ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels. This value will be adjusted to meet a desired frame rate when AutoAdjustSampleDistances is on.
- `obj.SetMinimumImageSampleDistance (float )` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted.
- `float = obj.GetMinimumImageSampleDistanceMinValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted.
- `float = obj.GetMinimumImageSampleDistanceMaxValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted.
- `float = obj.GetMinimumImageSampleDistance ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted.

- `obj.SetMaximumImageSampleDistance (float )` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted.
- `float = obj.GetMaximumImageSampleDistanceMinValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted.
- `float = obj.GetMaximumImageSampleDistanceMaxValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted.
- `float = obj.GetMaximumImageSampleDistance ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted.
- `obj.SetFinalColorWindow (float )` - Set/Get the window / level applied to the final color. This allows brightness / contrast adjustments on the final image. window is the width of the window. level is the center of the window. Initial window value is 1.0 Initial level value is 0.5 window cannot be null but can be negative, this way values will be reversed. —window— can be larger than 1.0 level can be any real value.
- `float = obj.GetFinalColorWindow ()` - Set/Get the window / level applied to the final color. This allows brightness / contrast adjustments on the final image. window is the width of the window. level is the center of the window. Initial window value is 1.0 Initial level value is 0.5 window cannot be null but can be negative, this way values will be reversed. —window— can be larger than 1.0 level can be any real value.
- `obj.SetFinalColorLevel (float )` - Set/Get the window / level applied to the final color. This allows brightness / contrast adjustments on the final image. window is the width of the window. level is the center of the window. Initial window value is 1.0 Initial level value is 0.5 window cannot be null but can be negative, this way values will be reversed. —window— can be larger than 1.0 level can be any real value.
- `float = obj.GetFinalColorLevel ()` - Set/Get the window / level applied to the final color. This allows brightness / contrast adjustments on the final image. window is the width of the window. level is the center of the window. Initial window value is 1.0 Initial level value is 0.5 window cannot be null but can be negative, this way values will be reversed. —window— can be larger than 1.0 level can be any real value.
- `obj.SetMaxMemoryInBytes (vtkIdType )` - Maximum size of the 3D texture in GPU memory. Will default to the size computed from the graphics card. Can be adjusted by the user.
- `vtkIdType = obj.GetMaxMemoryInBytes ()` - Maximum size of the 3D texture in GPU memory. Will default to the size computed from the graphics card. Can be adjusted by the user.
- `obj.SetMaxMemoryFraction (float )` - Maximum fraction of the MaxMemoryInBytes that should be used to hold the texture. Valid values are 0.1 to 1.0.
- `float = obj.GetMaxMemoryFractionMinValue ()` - Maximum fraction of the MaxMemoryInBytes that should be used to hold the texture. Valid values are 0.1 to 1.0.
- `float = obj.GetMaxMemoryFractionMaxValue ()` - Maximum fraction of the MaxMemoryInBytes that should be used to hold the texture. Valid values are 0.1 to 1.0.
- `float = obj.GetMaxMemoryFraction ()` - Maximum fraction of the MaxMemoryInBytes that should be used to hold the texture. Valid values are 0.1 to 1.0.
- `obj.SetReportProgress (bool )` - Tells if the mapper will report intermediate progress. Initial value is true. As the progress works with a GL blocking call (glFinish()), this can be useful for huge dataset but can slow down rendering of small dataset. It should be set to true for big dataset or complex shading and streaming but to false for small datasets.

- `bool = obj.GetReportProgress ()` - Tells if the mapper will report intermediate progress. Initial value is true. As the progress works with a GL blocking call (glFinish()), this can be useful for huge dataset but can slow down rendering of small dataset. It should be set to true for big dataset or complex shading and streaming but to false for small datasets.
- `int = obj.IsRenderSupported (vtkRenderWindow , vtkVolumeProperty )`
- `obj.CreateCanonicalView (vtkRenderer ren, vtkVolume volume, vtkImageData image, int blend\_mode, do`
- `obj.SetMaskInput (vtkImageData mask)`
- `vtkImageData = obj.GetMaskInput ()`
- `obj.SetMaskBlendFactor (float )` - Tells how much mask color transfer function is used compared to the standard color transfer function when the mask is true. 0.0 means only standard color transfer function. 1.0 means only mask color tranfer function. Initial value is 1.0.
- `float = obj.GetMaskBlendFactorMinValue ()` - Tells how much mask color transfer function is used compared to the standard color transfer function when the mask is true. 0.0 means only standard color transfer function. 1.0 means only mask color tranfer function. Initial value is 1.0.
- `float = obj.GetMaskBlendFactorMaxValue ()` - Tells how much mask color transfer function is used compared to the standard color transfer function when the mask is true. 0.0 means only standard color transfer function. 1.0 means only mask color tranfer function. Initial value is 1.0.
- `float = obj.GetMaskBlendFactor ()` - Tells how much mask color transfer function is used compared to the standard color transfer function when the mask is true. 0.0 means only standard color transfer function. 1.0 means only mask color tranfer function. Initial value is 1.0.

41.14 vtkHAVSVolumeMapper

41.14.1 Usage

vtkHAVSVolumeMapper is a class that renders polygonal data (represented as an unstructured grid) using the Hardware-Assisted Visibility Sorting (HAVS) algorithm. First the unique triangles are sorted in object space, then they are sorted in image space using a fixed size A-buffer implemented on the GPU called the k-buffer. The HAVS algorithm excels at rendering large datasets quickly. The trade-off is that the algorithm may produce some rendering artifacts due to an insufficient k size (currently 2 or 6 is supported) or read/write race conditions.

A built in level-of-detail (LOD) approach samples the geometry using one of two heuristics (field or area). If LOD is enabled, the amount of geometry that is sampled and rendered changes dynamically to stay within the target frame rate. The field sampling method generally works best for datasets with cell sizes that don't vary much in size. On the contrary, the area sampling approach gives better approximations when the volume has a lot of variation in cell size.

The HAVS algorithm uses several advanced features on graphics hardware. The k-buffer sorting network is implemented using framebuffer objects (FBOs) with multiple render targets (MRTs). Therefore, only cards that support these features can run the algorithm (at least an ATI 9500 or an NVidia NV40 (6600)).

.SECTION Notes

Several issues had to be addressed to get the HAVS algorithm working within the vtk framework. These additions forced the code to forsake speed for the sake of compliance and robustness.

The HAVS algorithm operates on the triangles that compose the mesh. Therefore, before rendering, the cells are decomposed into unique triangles and stored on the GPU for efficient rendering. The use of GPU data structures is only recommended if the entire geometry can fit in graphics memory. Otherwise this feature should be disabled.

Another new feature is the handling of mixed data types (eg., polygonal data with volume data). This is handled by reading the z-buffer from the current window and copying it into the framebuffer object for off-screen rendering. The depth test is then enabled so that the volume only appears over the opaque geometry.

Finally, the results of the off-screen rendering are blended into the framebuffer as a transparent, view-aligned texture.

Instead of using a preintegrated 3D lookup table for storing the ray integral, this implementation uses partial pre-integration. This improves the performance of dynamic transfer function updates by avoiding a costly preprocess of the table.

A final change to the original algorithm is the handling of non-convexities in the mesh. Due to read/write hazards that may create undesired artifacts with non-convexities when using a inside/outside toggle in the fragment program, another approach was employed. To handle non-convexities, the fragment shader determines if a ray-gap is larger than the max cell size and kill the fragment if so. This approximation performs rather well in practice but may miss small non-convexities.

For more information on the HAVS algorithm see:

"Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering" by S. P. Callahan, M. Ikits, J. L. D. Comba, and C. T. Silva, IEEE Transactions of Visualization and Computer Graphics; May/June 2005.

For more information on the Level-of-Detail algorithm, see:

"Interactive Rendering of Large Unstructured Grids Using Dynamic Level-of-Detail" by S. P. Callahan, J. L. D. Comba, P. Shirley, and C. T. Silva, Proceedings of IEEE Visualization '05, Oct. 2005.

.SECTION Acknowledgments

This code was developed by Steven P. Callahan under the supervision of Prof. Claudio T. Silva. The code also contains contributions from Milan Ikits, Linh Ha, Huy T. Vo, Carlos E. Scheidegger, and Joao L. D. Comba.

The work was supported by grants, contracts, and gifts from the National Science Foundation, the Department of Energy, the Army Research Office, and IBM.

The port of HAVS to VTK and ParaView has been primarily supported by Sandia National Labs.

To create an instance of class vtkHAVSVolumeMapper, simply invoke its constructor as follows

```
obj = vtkHAVSVolumeMapper
```

41.14.2 Methods

The class vtkHAVSVolumeMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHAVSVolumeMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkHAVSVolumeMapper = obj.NewInstance ()`
- `vtkHAVSVolumeMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetPartiallyRemoveNonConvexities (bool )` - regions by removing ray segments larger than the max cell size.
- `bool = obj.GetPartiallyRemoveNonConvexities ()` - regions by removing ray segments larger than the max cell size.
- `obj.SetLevelOfDetailTargetTime (float )` - Set/get the desired level of detail target time measured in frames/sec.
- `float = obj.GetLevelOfDetailTargetTime ()` - Set/get the desired level of detail target time measured in frames/sec.
- `obj.SetLevelOfDetail (bool )` - Turn on/off level-of-detail volume rendering
- `bool = obj.GetLevelOfDetail ()` - Turn on/off level-of-detail volume rendering

- `obj.SetLevelOfDetailMethod (int )` - Set/get the current level-of-detail method
- `int = obj.GetLevelOfDetailMethod ()` - Set/get the current level-of-detail method
- `obj.SetLevelOfDetailMethodField ()` - Set/get the current level-of-detail method
- `obj.SetLevelOfDetailMethodArea ()` - Set the kbuffer size
- `obj.SetKBufferSize (int )` - Set the kbuffer size
- `int = obj.GetKBufferSize ()` - Set the kbuffer size
- `obj.SetKBufferSizeTo2 ()` - Set the kbuffer size
- `obj.SetKBufferSizeTo6 ()` - Check hardware support for the HAVS algorithm. Necessary features include off-screen rendering, 32-bit fp textures, multiple render targets, and framebuffer objects. Subclasses must override this method to indicate if supported by Hardware.
- `bool = obj.SupportedByHardware ()` - Set/get whether or not the data structures should be stored on the GPU for better peformance.
- `obj.SetGPUDataStructures (bool )` - Set/get whether or not the data structures should be stored on the GPU for better peformance.
- `bool = obj.GetGPUDataStructures ()` - Set/get whether or not the data structures should be stored on the GPU for better peformance.

41.15 vtkOpenGLGPUVolumeRayCastMapper

41.15.1 Usage

This is the concrete implementation of a ray cast image display helper - a helper class responsible for drawing the image to the screen.

To create an instance of class vtkOpenGLGPUVolumeRayCastMapper, simply invoke its constructor as follows

```
obj = vtkOpenGLGPUVolumeRayCastMapper
```

41.15.2 Methods

The class vtkOpenGLGPUVolumeRayCastMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLGPUVolumeRayCastMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLGPUVolumeRayCastMapper = obj.NewInstance ()`
- `vtkOpenGLGPUVolumeRayCastMapper = obj.SafeDownCast (vtkObject o)`
- `int = obj.IsRenderSupported (vtkRenderWindow window, vtkVolumeProperty property)` - Based on hardware and properties, we may or may not be able to render using 3D texture mapping. This indicates if 3D texture mapping is supported by the hardware, and if the other extensions necessary to support the specific properties are available.

41.16 vtkOpenGLHAVSVolumeMapper

41.16.1 Usage

vtkHAVSVolumeMapper is a class that renders polygonal data (represented as an unstructured grid) using the Hardware-Assisted Visibility Sorting (HAVS) algorithm. First the unique triangles are sorted in object space, then they are sorted in image space using a fixed size A-buffer implemented on the GPU called the k-buffer. The HAVS algorithm excels at rendering large datasets quickly. The trade-off is that the algorithm may produce some rendering artifacts due to an insufficient k size (currently 2 or 6 is supported) or read/write race conditions.

A built in level-of-detail (LOD) approach samples the geometry using one of two heuristics (field or area). If LOD is enabled, the amount of geometry that is sampled and rendered changes dynamically to stay within the target frame rate. The field sampling method generally works best for datasets with cell sizes that don't vary much in size. On the contrary, the area sampling approach gives better approximations when the volume has a lot of variation in cell size.

The HAVS algorithm uses several advanced features on graphics hardware. The k-buffer sorting network is implemented using framebuffer objects (FBOs) with multiple render targets (MRTs). Therefore, only cards that support these features can run the algorithm (at least an ATI 9500 or an NVidia NV40 (6600)).

.SECTION Notes

Several issues had to be addressed to get the HAVS algorithm working within the vtk framework. These additions forced the code to forsake speed for the sake of compliance and robustness.

The HAVS algorithm operates on the triangles that compose the mesh. Therefore, before rendering, the cells are decomposed into unique triangles and stored on the GPU for efficient rendering. The use of GPU data structures is only recommended if the entire geometry can fit in graphics memory. Otherwise this feature should be disabled.

Another new feature is the handling of mixed data types (eg., polygonal data with volume data). This is handled by reading the z-buffer from the current window and copying it into the framebuffer object for off-screen rendering. The depth test is then enabled so that the volume only appears over the opaque geometry. Finally, the results of the off-screen rendering are blended into the framebuffer as a transparent, view-aligned texture.

Instead of using a preintegrated 3D lookup table for storing the ray integral, this implementation uses partial pre-integration. This improves the performance of dynamic transfer function updates by avoiding a costly preprocess of the table.

A final change to the original algorithm is the handling of non-convexities in the mesh. Due to read/write hazards that may create undesired artifacts with non-convexities when using a inside/outside toggle in the fragment program, another approach was employed. To handle non-convexities, the fragment shader determines if a ray-gap is larger than the max cell size and kill the fragment if so. This approximation performs rather well in practice but may miss small non-convexities.

For more information on the HAVS algorithm see:

"Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering" by S. P. Callahan, M. Ikits, J. L. D. Comba, and C. T. Silva, IEEE Transactions of Visualization and Computer Graphics; May/June 2005.

For more information on the Level-of-Detail algorithm, see:

"Interactive Rendering of Large Unstructured Grids Using Dynamic Level-of-Detail" by S. P. Callahan, J. L. D. Comba, P. Shirley, and C. T. Silva, Proceedings of IEEE Visualization '05, Oct. 2005.

.SECTION Acknowledgments

This code was developed by Steven P. Callahan under the supervision of Prof. Claudio T. Silva. The code also contains contributions from Milan Ikits, Linh Ha, Huy T. Vo, Carlos E. Scheidegger, and Joao L. D. Comba.

The work was supported by grants, contracts, and gifts from the National Science Foundation, the Department of Energy, the Army Research Office, and IBM.

The port of HAVS to VTK and ParaView has been primarily supported by Sandia National Labs.

To create an instance of class vtkOpenGLHAVSVolumeMapper, simply invoke its constructor as follows

```
obj = vtkOpenGLHAVSVolumeMapper
```

41.16.2 Methods

The class vtkOpenGLHAVSVolumeMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLHAVSVolumeMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLHAVSVolumeMapper = obj.NewInstance ()`
- `vtkOpenGLHAVSVolumeMapper = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer ren, vtkVolume vol)` - Render the volume
- `obj.ReleaseGraphicsResources (vtkWindow )`
- `obj.SetGPUDataStructures (bool )` - Set/get whether or not the data structures should be stored on the GPU for better peformance.
- `bool = obj.SupportedByHardware ()` - Check hardware support for the HAVS algorithm. Necessary features include off-screen rendering, 32-bit fp textures, multiple render targets, and framebuffer objects. Subclasses must override this method to indicate if supported by Hardware.

41.17 vtkOpenGLRayCastImageDisplayHelper

41.17.1 Usage

This is the concrete implementation of a ray cast image display helper - a helper class responsible for drawing the image to the screen.

To create an instance of class vtkOpenGLRayCastImageDisplayHelper, simply invoke its constructor as follows

```
obj = vtkOpenGLRayCastImageDisplayHelper
```

41.17.2 Methods

The class vtkOpenGLRayCastImageDisplayHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLRayCastImageDisplayHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLRayCastImageDisplayHelper = obj.NewInstance ()`
- `vtkOpenGLRayCastImageDisplayHelper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderTexture (vtkVolume vol, vtkRenderer ren, int imageMemorySize[2], int imageViewportSize[2]`
- `obj.RenderTexture (vtkVolume vol, vtkRenderer ren, int imageMemorySize[2], int imageViewportSize[2]`
- `obj.RenderTexture (vtkVolume vol, vtkRenderer ren, vtkFixedPointRayCastImage image, float requested`

41.18 vtkOpenGLVolumeTextureMapper2D

41.18.1 Usage

vtkOpenGLVolumeTextureMapper2D renders a volume using 2D texture mapping.

To create an instance of class vtkOpenGLVolumeTextureMapper2D, simply invoke its constructor as follows

```
obj = vtkOpenGLVolumeTextureMapper2D
```

41.18.2 Methods

The class vtkOpenGLVolumeTextureMapper2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLVolumeTextureMapper2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLVolumeTextureMapper2D = obj.NewInstance ()`
- `vtkOpenGLVolumeTextureMapper2D = obj.SafeDownCast (vtkObject o)`

41.19 vtkOpenGLVolumeTextureMapper3D

41.19.1 Usage

vtkOpenGLVolumeTextureMapper3D renders a volume using 3D texture mapping. See vtkVolumeTextureMapper3D for full description.

To create an instance of class vtkOpenGLVolumeTextureMapper3D, simply invoke its constructor as follows

```
obj = vtkOpenGLVolumeTextureMapper3D
```

41.19.2 Methods

The class vtkOpenGLVolumeTextureMapper3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOpenGLVolumeTextureMapper3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOpenGLVolumeTextureMapper3D = obj.NewInstance ()`
- `vtkOpenGLVolumeTextureMapper3D = obj.SafeDownCast (vtkObject o)`
- `int = obj.IsRenderSupported (vtkVolumeProperty )` - Is hardware rendering supported? No if the input data is more than one independent component, or if the hardware does not support the required extensions
- `int = obj.GetInitialized ()`
- `obj.ReleaseGraphicsResources (vtkWindow )` - Release any graphics resources that are being consumed by this texture. The parameter window could be used to determine which graphic resources to release.

41.20 vtkProjectedTetrahedraMapper

41.20.1 Usage

vtkProjectedTetrahedraMapper is an implementation of the classic Projected Tetrahedra algorithm presented by Shirley and Tuchman in "A Polygonal Approximation to Direct Scalar Volume Rendering" in Computer Graphics, December 1990.

.SECTION Bugs This mapper relies highly on the implementation of the OpenGL pipeline. A typical hardware driver has lots of options and some settings can cause this mapper to produce artifacts.

To create an instance of class vtkProjectedTetrahedraMapper, simply invoke its constructor as follows

```
obj = vtkProjectedTetrahedraMapper
```

41.20.2 Methods

The class vtkProjectedTetrahedraMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkProjectedTetrahedraMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkProjectedTetrahedraMapper = obj.NewInstance ()`
- `vtkProjectedTetrahedraMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetVisibilitySort (vtkVisibilitySort sort)`
- `vtkVisibilitySort = obj.GetVisibilitySort ()`

41.21 vtkRayCastImageDisplayHelper

41.21.1 Usage

This is a helper class for drawing images created from ray casting on the screen. This is the abstract device-independent superclass.

To create an instance of class vtkRayCastImageDisplayHelper, simply invoke its constructor as follows

```
obj = vtkRayCastImageDisplayHelper
```

41.21.2 Methods

The class vtkRayCastImageDisplayHelper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRayCastImageDisplayHelper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkRayCastImageDisplayHelper = obj.NewInstance ()`
- `vtkRayCastImageDisplayHelper = obj.SafeDownCast (vtkObject o)`
- `obj.RenderTexture (vtkVolume vol, vtkRenderer ren, int imageMemorySize[2], int imageViewportSize[2]`
- `obj.RenderTexture (vtkVolume vol, vtkRenderer ren, int imageMemorySize[2], int imageViewportSize[2]`

- obj.RenderTexture (vtkVolume vol, vtkRenderer ren, vtkFixedPointRayCastImage image, float requested
- obj.SetPreMultipliedColors (int)
- int = obj.GetPreMultipliedColorsMinValue ()
- int = obj.GetPreMultipliedColorsMaxValue ()
- int = obj.GetPreMultipliedColors ()
- obj.PreMultipliedColorsOn ()
- obj.PreMultipliedColorsOff ()
- obj.SetPixelScale (float) - Set / Get the pixel scale to be applied to the image before display. Can be set to scale the incoming pixel values - for example the fixed point mapper uses the unsigned short API but with 15 bit values so needs a scale of 2.0.
- float = obj.GetPixelScale () - Set / Get the pixel scale to be applied to the image before display. Can be set to scale the incoming pixel values - for example the fixed point mapper uses the unsigned short API but with 15 bit values so needs a scale of 2.0.

41.22 vtkRecursiveSphereDirectionEncoder

41.22.1 Usage

vtkRecursiveSphereDirectionEncoder is a direction encoder which uses the vertices of a recursive subdivision of an octahedron (with the vertices pushed out onto the surface of an enclosing sphere) to encode directions into a two byte value.

To create an instance of class vtkRecursiveSphereDirectionEncoder, simply invoke its constructor as follows

```
obj = vtkRecursiveSphereDirectionEncoder
```

41.22.2 Methods

The class vtkRecursiveSphereDirectionEncoder has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, obj is an instance of the vtkRecursiveSphereDirectionEncoder class.

- string = obj.GetClassName ()
- int = obj.IsA (string name)
- vtkRecursiveSphereDirectionEncoder = obj.NewInstance ()
- vtkRecursiveSphereDirectionEncoder = obj.SafeDownCast (vtkObject o)
- int = obj.GetEncodedDirection (float n[3]) - Given a normal vector n, return the encoded direction
- float = obj.GetDecodedGradient (int value) - / Given an encoded value, return a pointer to the normal vector
- int = obj.GetNumberOfEncodedDirections (void) - Return the number of encoded directions

- `obj.SetRecursionDepth (int )` - Set / Get the recursion depth for the subdivision. This indicates how many time one triangle on the initial 8-sided sphere model is replaced by four triangles formed by connecting triangle edge midpoints. A recursion level of 0 yields 8 triangles with 6 unique vertices. The normals are the vectors from the sphere center through the vertices. The number of directions will be 11 since the four normals with 0 z values will be duplicated in the table - once with +0 values and the other time with -0 values, and an addition index will be used to represent the (0,0,0) normal. If we instead choose a recursion level of 6 (the maximum that can fit within 2 bytes) the number of directions is 16643, with 16386 unique directions and a zero normal.

- `int = obj.GetRecursionDepthMinValue ()` - Set / Get the recursion depth for the subdivision. This indicates how many time one triangle on the initial 8-sided sphere model is replaced by four triangles formed by connecting triangle edge midpoints. A recursion level of 0 yields 8 triangles with 6 unique vertices. The normals are the vectors from the sphere center through the vertices. The number of directions will be 11 since the four normals with 0 z values will be duplicated in the table - once with +0 values and the other time with -0 values, and an addition index will be used to represent the (0,0,0) normal. If we instead choose a recursion level of 6 (the maximum that can fit within 2 bytes) the number of directions is 16643, with 16386 unique directions and a zero normal.

- `int = obj.GetRecursionDepthMaxValue ()` - Set / Get the recursion depth for the subdivision. This indicates how many time one triangle on the initial 8-sided sphere model is replaced by four triangles formed by connecting triangle edge midpoints. A recursion level of 0 yields 8 triangles with 6 unique vertices. The normals are the vectors from the sphere center through the vertices. The number of directions will be 11 since the four normals with 0 z values will be duplicated in the table - once with +0 values and the other time with -0 values, and an addition index will be used to represent the (0,0,0) normal. If we instead choose a recursion level of 6 (the maximum that can fit within 2 bytes) the number of directions is 16643, with 16386 unique directions and a zero normal.

- `int = obj.GetRecursionDepth ()` - Set / Get the recursion depth for the subdivision. This indicates how many time one triangle on the initial 8-sided sphere model is replaced by four triangles formed by connecting triangle edge midpoints. A recursion level of 0 yields 8 triangles with 6 unique vertices. The normals are the vectors from the sphere center through the vertices. The number of directions will be 11 since the four normals with 0 z values will be duplicated in the table - once with +0 values and the other time with -0 values, and an addition index will be used to represent the (0,0,0) normal. If we instead choose a recursion level of 6 (the maximum that can fit within 2 bytes) the number of directions is 16643, with 16386 unique directions and a zero normal.

41.23 vtkSphericalDirectionEncoder

41.23.1 Usage

vtkSphericalDirectionEncoder is a direction encoder which uses spherical coordinates for mapping (nx, ny, nz) into an azimuth, elevation pair.

To create an instance of class vtkSphericalDirectionEncoder, simply invoke its constructor as follows

```
obj = vtkSphericalDirectionEncoder
```

41.23.2 Methods

The class vtkSphericalDirectionEncoder has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSphericalDirectionEncoder class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkSphericalDirectionEncoder = obj.NewInstance ()`
- `vtkSphericalDirectionEncoder = obj.SafeDownCast (vtkObject o)`
- `int = obj.GetEncodedDirection (float n[3])` - Given a normal vector n, return the encoded direction
- `float = obj.GetDecodedGradient (int value)` - / Given an encoded value, return a pointer to the normal vector
- `int = obj.GetNumberOfEncodedDirections (void )` - Get the decoded gradient table. There are this-¿GetNumberOfEncodedDirections() entries in the table, each containing a normal (direction) vector. This is a flat structure - 3 times the number of directions floats in an array.

41.24 vtkUnstructuredGridBunykRayCastFunction

41.24.1 Usage

vtkUnstructuredGridBunykRayCastFunction is a concrete implementation of a ray cast function for unstructured grid data. This class was based on the paper "Simple, Fast, Robust Ray Casting of Irregular Grids" by Paul Bunyk, Arie Kaufmna, and Claudio Silva. This method is quite memory intensive (with extra explicit copies of the data) and therefore should not be used for very large data. This method assumes that the input data is composed entirely of tetras - use vtkDataSetTriangleFilter before setting the input on the mapper.

The basic idea of this method is as follows:

1) Enumerate the triangles. At each triangle have space for some information that will be used during rendering. This includes which tetra the triangles belong to, the plane equation and the Barycentric coefficients.

2) Keep a reference to all four triangles for each tetra.

3) At the beginning of each render, do the precomputation. This includes creating an array of transformed points (in view coordinates) and computing the view dependent info per triangle (plane equations and barycentric coords in view space)

4) Find all front facing boundary triangles (a triangle is on the boundary if it belongs to only one tetra). For each triangle, find all pixels in the image that intersect the triangle, and add this to the sorted (by depth) intersection list at each pixel.

5) For each ray cast, traverse the intersection list. At each intersection, accumulate opacity and color contribution per tetra along the ray until you reach an exiting triangle (on the boundary).

To create an instance of class vtkUnstructuredGridBunykRayCastFunction, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridBunykRayCastFunction
```

41.24.2 Methods

The class vtkUnstructuredGridBunykRayCastFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridBunykRayCastFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridBunykRayCastFunction = obj.NewInstance ()`
- `vtkUnstructuredGridBunykRayCastFunction = obj.SafeDownCast (vtkObject o)`

41.25 vtkUnstructuredGridHomogeneousRayIntegrator

41.25.1 Usage

vtkUnstructuredGridHomogeneousRayIntegrator performs homogeneous ray integration. This is a good method to use when volume rendering scalars that are defined on cells.

To create an instance of class vtkUnstructuredGridHomogeneousRayIntegrator, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridHomogeneousRayIntegrator
```

41.25.2 Methods

The class vtkUnstructuredGridHomogeneousRayIntegrator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridHomogeneousRayIntegrator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridHomogeneousRayIntegrator = obj.NewInstance ()`
- `vtkUnstructuredGridHomogeneousRayIntegrator = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (vtkVolume volume, vtkDataArray scalars)`
- `obj.Integrate (vtkDoubleArray intersectionLengths, vtkDataArray nearIntersections, vtkDataArray far`
- `obj.SetTransferFunctionTableSize (int )` - For quick lookup, the transfer function is sampled into a table. This parameter sets how big of a table to use. By default, 1024 entries are used.
- `int = obj.GetTransferFunctionTableSize ()` - For quick lookup, the transfer function is sampled into a table. This parameter sets how big of a table to use. By default, 1024 entries are used.

41.26 vtkUnstructuredGridLinearRayIntegrator

41.26.1 Usage

vtkUnstructuredGridLinearRayIntegrator performs piecewise linear ray integration. Considering that transfer functions in VTK are piecewise linear, this class should give the "correct" integration under most circumstances. However, the computations performed are fairly hefty and should, for the most part, only be used as a benchmark for other, faster methods.

To create an instance of class vtkUnstructuredGridLinearRayIntegrator, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridLinearRayIntegrator
```

41.26.2 Methods

The class vtkUnstructuredGridLinearRayIntegrator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridLinearRayIntegrator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkUnstructuredGridLinearRayIntegrator = obj.NewInstance ()`
- `vtkUnstructuredGridLinearRayIntegrator = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (vtkVolume volume, vtkDataArray scalars)`
- `obj.Integrate (vtkDoubleArray intersectionLengths, vtkDataArray nearIntersections, vtkDataArray far`

41.27 vtkUnstructuredGridPartialPreIntegration

41.27.1 Usage

vtkUnstructuredGridPartialPreIntegration performs piecewise linear ray integration. This will give the same results as vtkUnstructuredGridLinearRayIntegration (with potentially a error due to table lookup quantization), but should be notably faster. The algorithm used is given by Moreland and Angel, "A Fast High Accuracy Volume Renderer for Unstructured Data."

This class is thread safe only after the first instance is created.

To create an instance of class vtkUnstructuredGridPartialPreIntegration, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridPartialPreIntegration
```

41.27.2 Methods

The class vtkUnstructuredGridPartialPreIntegration has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridPartialPreIntegration class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridPartialPreIntegration = obj.NewInstance ()`
- `vtkUnstructuredGridPartialPreIntegration = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (vtkVolume volume, vtkDataArray scalars)`
- `obj.Integrate (vtkDoubleArray intersectionLengths, vtkDataArray nearIntersections, vtkDataArray far`

41.28 vtkUnstructuredGridPreIntegration

41.28.1 Usage

vtkUnstructuredGridPreIntegration performs ray integration by looking into a precomputed table. The result should be equivalent to that computed by vtkUnstructuredGridLinearRayIntegrator and vtkUnstructuredGridPartialPreIntegration, but faster than either one. The pre-integration algorithm was first introduced by Roettger, Kraus, and Ertl in "Hardware-Accelerated Volume And Isosurface Rendering Based On Cell-Projection."

Due to table size limitations, a table can only be indexed by independent scalars. Thus, dependent scalars are not supported.

To create an instance of class vtkUnstructuredGridPreIntegration, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridPreIntegration
```

41.28.2 Methods

The class vtkUnstructuredGridPreIntegration has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridPreIntegration class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridPreIntegration = obj.NewInstance ()`
- `vtkUnstructuredGridPreIntegration = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (vtkVolume volume, vtkDataArray scalars)`
- `obj.Integrate (vtkDoubleArray intersectionLengths, vtkDataArray nearIntersections, vtkDataArray far`
- `vtkUnstructuredGridVolumeRayIntegrator = obj.GetIntegrator ()` - The class used to fill the pre integration table. By default, a vtkUnstructuredGridPartialPreIntegration is built.
- `obj.SetIntegrator (vtkUnstructuredGridVolumeRayIntegrator )` - The class used to fill the pre integration table. By default, a vtkUnstructuredGridPartialPreIntegration is built.
- `obj.SetIntegrationTableScalarResolution (int )` - Set/Get the size of the integration table built.
- `int = obj.GetIntegrationTableScalarResolution ()` - Set/Get the size of the integration table built.
- `obj.SetIntegrationTableLengthResolution (int )` - Set/Get the size of the integration table built.
- `int = obj.GetIntegrationTableLengthResolution ()` - Set/Get the size of the integration table built.
- `double = obj.GetIntegrationTableScalarShift (int component)` - Get how an integration table is indexed.
- `double = obj.GetIntegrationTableScalarScale (int component)` - Get how an integration table is indexed.
- `double = obj.GetIntegrationTableLengthScale ()` - Get how an integration table is indexed.
- `int = obj.GetIncrementalPreIntegration ()` - Get/set whether to use incremental pre-integration (by default it's on). Incremental pre-integration is much faster but can introduce error due to numerical imprecision. Under most circumstances, the error is not noticable.
- `obj.SetIncrementalPreIntegration (int )` - Get/set whether to use incremental pre-integration (by default it's on). Incremental pre-integration is much faster but can introduce error due to numerical imprecision. Under most circumstances, the error is not noticable.
- `obj.IncrementalPreIntegrationOn ()` - Get/set whether to use incremental pre-integration (by default it's on). Incremental pre-integration is much faster but can introduce error due to numerical imprecision. Under most circumstances, the error is not noticable.
- `obj.IncrementalPreIntegrationOff ()` - Get/set whether to use incremental pre-integration (by default it's on). Incremental pre-integration is much faster but can introduce error due to numerical imprecision. Under most circumstances, the error is not noticable.

41.29 vtkUnstructuredGridVolumeMapper

41.29.1 Usage

vtkUnstructuredGridVolumeMapper is the abstract definition of a volume mapper for unstructured data (vtkUnstructuredGrid). Several basic types of volume mappers are supported as subclasses

To create an instance of class vtkUnstructuredGridVolumeMapper, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridVolumeMapper
```

41.29.2 Methods

The class vtkUnstructuredGridVolumeMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridVolumeMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridVolumeMapper = obj.NewInstance ()`
- `vtkUnstructuredGridVolumeMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkUnstructuredGrid )` - Set/Get the input data
- `obj.SetInput (vtkDataSet )` - Set/Get the input data
- `vtkUnstructuredGrid = obj.GetInput ()` - Set/Get the input data
- `obj.SetBlendMode (int )`
- `obj.SetBlendModeToComposite ()`
- `obj.SetBlendModeToMaximumIntensity ()`
- `int = obj.GetBlendMode ()`

41.30 vtkUnstructuredGridVolumeRayCastFunction

41.30.1 Usage

vtkUnstructuredGridVolumeRayCastFunction is a superclass for ray casting functions that can be used within a vtkUnstructuredGridVolumeRayCastMapper.

To create an instance of class vtkUnstructuredGridVolumeRayCastFunction, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridVolumeRayCastFunction
```

41.30.2 Methods

The class vtkUnstructuredGridVolumeRayCastFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridVolumeRayCastFunction class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkUnstructuredGridVolumeRayCastFunction = obj.NewInstance ()`
- `vtkUnstructuredGridVolumeRayCastFunction = obj.SafeDownCast (vtkObject o)`

41.31 vtkUnstructuredGridVolumeRayCastIterator

41.31.1 Usage

vtkUnstructuredGridVolumeRayCastIterator is a superclass for iterating over the intersections of a viewing ray with a group of unstructured cells. These iterators are created with a vtkUnstructuredGridVolumeRayCastFunction.

To create an instance of class vtkUnstructuredGridVolumeRayCastIterator, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridVolumeRayCastIterator
```

41.31.2 Methods

The class vtkUnstructuredGridVolumeRayCastIterator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridVolumeRayCastIterator class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridVolumeRayCastIterator = obj.NewInstance ()`
- `vtkUnstructuredGridVolumeRayCastIterator = obj.SafeDownCast (vtkObject o)`
- `obj.Initialize (int x, int y)` - Initializes the iteration to the start of the ray at the given screen coordinates.
- `vtkIdType = obj.GetNextIntersections (vtkIdList intersectedCells, vtkDoubleArray intersectionLengt` - Get the intersections of the next several cells. The cell ids are stored in ţntersectedCells and the length of each ray segment within the cell is stored in ţntersectionLengths. The point scalars şcalars are interpolated and stored in ņearIntersections and ţarIntersections. ţntersectedCells, ţntersectionLengths, or şcalars may be ŅULL to supress passing the associated information. The number of intersections actually encountered is returned. 0 is returned if and only if no more intersections are to be found.
- `obj.SetBounds (double , double )` - Set/get the bounds of the cast ray (in viewing coordinates). By default the range is [0,1].
- `obj.SetBounds (double  a[2])` - Set/get the bounds of the cast ray (in viewing coordinates). By default the range is [0,1].
- `double = obj. GetBounds ()` - Set/get the bounds of the cast ray (in viewing coordinates). By default the range is [0,1].
- `obj.SetMaxNumberOfIntersections (vtkIdType )`
- `vtkIdType = obj.GetMaxNumberOfIntersections ()`

41.32 vtkUnstructuredGridVolumeRayCastMapper

41.32.1 Usage

This is a software ray caster for rendering volumes in vtkUnstructuredGrid.

To create an instance of class vtkUnstructuredGridVolumeRayCastMapper, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridVolumeRayCastMapper
```

41.32.2 Methods

The class vtkUnstructuredGridVolumeRayCastMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridVolumeRayCastMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridVolumeRayCastMapper = obj.NewInstance ()`
- `vtkUnstructuredGridVolumeRayCastMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetImageSampleDistance (float )` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `float = obj.GetImageSampleDistanceMinValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `float = obj.GetImageSampleDistanceMaxValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `float = obj.GetImageSampleDistance ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `obj.SetMinimumImageSampleDistance (float )` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMinimumImageSampleDistanceMinValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMinimumImageSampleDistanceMaxValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMinimumImageSampleDistance ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `obj.SetMaximumImageSampleDistance (float )` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMaximumImageSampleDistanceMinValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMaximumImageSampleDistanceMaxValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted

- `float = obj.GetMaximumImageSampleDistance ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted
- `obj.SetAutoAdjustSampleDistances (int )` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `int = obj.GetAutoAdjustSampleDistancesMinValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `int = obj.GetAutoAdjustSampleDistancesMaxValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `int = obj.GetAutoAdjustSampleDistances ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `obj.AutoAdjustSampleDistancesOn ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `obj.AutoAdjustSampleDistancesOff ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `obj.SetNumberOfThreads (int )` - Set/Get the number of threads to use. This by default is equal to the number of available processors detected.
- `int = obj.GetNumberOfThreads ()` - Set/Get the number of threads to use. This by default is equal to the number of available processors detected.
- `obj.SetIntermixIntersectingGeometry (int )` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `int = obj.GetIntermixIntersectingGeometryMinValue ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `int = obj.GetIntermixIntersectingGeometryMaxValue ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `int = obj.GetIntermixIntersectingGeometry ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `obj.IntermixIntersectingGeometryOn ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `obj.IntermixIntersectingGeometryOff ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `obj.SetRayCastFunction (vtkUnstructuredGridVolumeRayCastFunction f)` - Set/Get the helper class for casting rays.
- `vtkUnstructuredGridVolumeRayCastFunction = obj.GetRayCastFunction ()` - Set/Get the helper class for casting rays.
- `obj.SetRayIntegrator (vtkUnstructuredGridVolumeRayIntegrator ri)` - Set/Get the helper class for integrating rays. If set to NULL, a default integrator will be assigned.

- `vtkUnstructuredGridVolumeRayIntegrator = obj.GetRayIntegrator ()` - Set/Get the helper class for integrating rays. If set to NULL, a default integrator will be assigned.

- `obj.CastRays (int threadID, int threadCount)`

41.33 vtkUnstructuredGridVolumeRayIntegrator

41.33.1 Usage

vtkUnstructuredGridVolumeRayIntegrator is a superclass for ray integration functions that can be used within a vtkUnstructuredGridVolumeRayCastMapper.

To create an instance of class vtkUnstructuredGridVolumeRayIntegrator, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridVolumeRayIntegrator
```

41.33.2 Methods

The class vtkUnstructuredGridVolumeRayIntegrator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridVolumeRayIntegrator class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`

- `vtkUnstructuredGridVolumeRayIntegrator = obj.NewInstance ()`

- `vtkUnstructuredGridVolumeRayIntegrator = obj.SafeDownCast (vtkObject o)`

- `obj.Initialize (vtkVolume volume, vtkDataArray scalars)` - Set up the integrator with the given properties and scalars.

- `obj.Integrate (vtkDoubleArray intersectionLengths, vtkDataArray nearIntersections, vtkDataArray far` - Given a set of intersections (defined by the three arrays), compute the peicewise integration of the array in front to back order. /c intersectionLengths holds the lengths of each peicewise segment. /c nearIntersections and /c farIntersections hold the scalar values at the front and back of each segment. /c color should contain the RGBA value of the volume in front of the segments passed in, and the result will be placed back into /c color.

41.34 vtkUnstructuredGridVolumeZSweepMapper

41.34.1 Usage

This is a volume mapper for unstructured grid implemented with the ZSweep algorithm. This is a software projective method.

To create an instance of class vtkUnstructuredGridVolumeZSweepMapper, simply invoke its constructor as follows

```
obj = vtkUnstructuredGridVolumeZSweepMapper
```

41.34.2 Methods

The class vtkUnstructuredGridVolumeZSweepMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkUnstructuredGridVolumeZSweepMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkUnstructuredGridVolumeZSweepMapper = obj.NewInstance ()`
- `vtkUnstructuredGridVolumeZSweepMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetImageSampleDistance (float )` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `float = obj.GetImageSampleDistanceMinValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `float = obj.GetImageSampleDistanceMaxValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `float = obj.GetImageSampleDistance ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `obj.SetMinimumImageSampleDistance (float )` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMinimumImageSampleDistanceMinValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMinimumImageSampleDistanceMaxValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMinimumImageSampleDistance ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `obj.SetMaximumImageSampleDistance (float )` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMaximumImageSampleDistanceMinValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMaximumImageSampleDistanceMaxValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted
- `float = obj.GetMaximumImageSampleDistance ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted
- `obj.SetAutoAdjustSampleDistances (int )` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `int = obj.GetAutoAdjustSampleDistancesMinValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).

- `int = obj.GetAutoAdjustSampleDistancesMaxValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `int = obj.GetAutoAdjustSampleDistances ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `obj.AutoAdjustSampleDistancesOn ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `obj.AutoAdjustSampleDistancesOff ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).
- `obj.SetIntermixIntersectingGeometry (int )` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `int = obj.GetIntermixIntersectingGeometryMinValue ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `int = obj.GetIntermixIntersectingGeometryMaxValue ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `int = obj.GetIntermixIntersectingGeometry ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `obj.IntermixIntersectingGeometryOn ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `obj.IntermixIntersectingGeometryOff ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.
- `int = obj.GetMaxPixelListSize ()` - Maximum size allowed for a pixel list. Default is 32. During the rendering, if a list of pixel is full, incremental compositing is performed. Even if it is a user setting, it is an advanced parameter. You have to understand how the algorithm works to change this value.
- `obj.SetMaxPixelListSize (int size)` - Change the maximum size allowed for a pixel list. It is an advanced parameter.
- `obj.SetRayIntegrator (vtkUnstructuredGridVolumeRayIntegrator ri)` - Set/Get the helper class for integrating rays. If set to NULL, a default integrator will be assigned.
- `vtkUnstructuredGridVolumeRayIntegrator = obj.GetRayIntegrator ()` - Set/Get the helper class for integrating rays. If set to NULL, a default integrator will be assigned.

41.35 vtkVolumeMapper

41.35.1 Usage

vtkVolumeMapper is the abstract definition of a volume mapper for regular rectilinear data (vtkImageData). Several basic types of volume mappers are supported.

To create an instance of class vtkVolumeMapper, simply invoke its constructor as follows

```
obj = vtkVolumeMapper
```

41.35.2 Methods

The class vtkVolumeMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeMapper = obj.NewInstance ()`
- `vtkVolumeMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetInput (vtkImageData )` - Set/Get the input data
- `obj.SetInput (vtkDataSet )` - Set/Get the input data
- `vtkImageData = obj.GetInput ()` - Set/Get the input data
- `obj.SetBlendMode (int )` - Set/Get the blend mode. Currently this is only supported by the vtkFixedPointVolumeRayCastMapper - other mappers have different ways to set this (supplying a function to a vtkVolumeRayCastMapper) or don't have any options (vtkVolumeTextureMapper2D supports only compositing)
- `obj.SetBlendModeToComposite ()` - Set/Get the blend mode. Currently this is only supported by the vtkFixedPointVolumeRayCastMapper - other mappers have different ways to set this (supplying a function to a vtkVolumeRayCastMapper) or don't have any options (vtkVolumeTextureMapper2D supports only compositing)
- `obj.SetBlendModeToMaximumIntensity ()` - Set/Get the blend mode. Currently this is only supported by the vtkFixedPointVolumeRayCastMapper - other mappers have different ways to set this (supplying a function to a vtkVolumeRayCastMapper) or don't have any options (vtkVolumeTextureMapper2D supports only compositing)
- `obj.SetBlendModeToMinimumIntensity ()` - Set/Get the blend mode. Currently this is only supported by the vtkFixedPointVolumeRayCastMapper - other mappers have different ways to set this (supplying a function to a vtkVolumeRayCastMapper) or don't have any options (vtkVolumeTextureMapper2D supports only compositing)
- `int = obj.GetBlendMode ()` - Set/Get the blend mode. Currently this is only supported by the vtkFixedPointVolumeRayCastMapper - other mappers have different ways to set this (supplying a function to a vtkVolumeRayCastMapper) or don't have any options (vtkVolumeTextureMapper2D supports only compositing)
- `obj.SetCropping (int )` - Turn On/Off orthogonal cropping. (Clipping planes are perpendicular to the coordinate axes.)
- `int = obj.GetCroppingMinValue ()` - Turn On/Off orthogonal cropping. (Clipping planes are perpendicular to the coordinate axes.)
- `int = obj.GetCroppingMaxValue ()` - Turn On/Off orthogonal cropping. (Clipping planes are perpendicular to the coordinate axes.)
- `int = obj.GetCropping ()` - Turn On/Off orthogonal cropping. (Clipping planes are perpendicular to the coordinate axes.)
- `obj.CroppingOn ()` - Turn On/Off orthogonal cropping. (Clipping planes are perpendicular to the coordinate axes.)

- `obj.CroppingOff ()` - Turn On/Off orthogonal cropping. (Clipping planes are perpendicular to the coordinate axes.)
- `obj.SetCroppingRegionPlanes (double , double , double , double , double , double )` - Set/Get the Cropping Region Planes (xmin, xmax, ymin, ymax, zmin, zmax) These planes are defined in volume coordinates - spacing and origin are considered.
- `obj.SetCroppingRegionPlanes (double a[6])` - Set/Get the Cropping Region Planes (xmin, xmax, ymin, ymax, zmin, zmax) These planes are defined in volume coordinates - spacing and origin are considered.
- `double = obj. GetCroppingRegionPlanes ()` - Set/Get the Cropping Region Planes (xmin, xmax, ymin, ymax, zmin, zmax) These planes are defined in volume coordinates - spacing and origin are considered.
- `double = obj. GetVoxelCroppingRegionPlanes ()` - Get the cropping region planes in voxels. Only valid during the rendering process
- `obj.SetCroppingRegionFlags (int )` - Set the flags for the cropping regions. The clipping planes divide the volume into 27 regions - there is one bit for each region. The regions start from the one containing voxel (0,0,0), moving along the x axis fastest, the y axis next, and the z axis slowest. These are represented from the lowest bit to bit number 27 in the integer containing the flags. There are several convenience functions to set some common configurations - subvolume (the default), fence (between any of the clip plane pairs), inverted fence, cross (between any two of the clip plane pairs) and inverted cross.
- `int = obj.GetCroppingRegionFlagsMinValue ()` - Set the flags for the cropping regions. The clipping planes divide the volume into 27 regions - there is one bit for each region. The regions start from the one containing voxel (0,0,0), moving along the x axis fastest, the y axis next, and the z axis slowest. These are represented from the lowest bit to bit number 27 in the integer containing the flags. There are several convenience functions to set some common configurations - subvolume (the default), fence (between any of the clip plane pairs), inverted fence, cross (between any two of the clip plane pairs) and inverted cross.
- `int = obj.GetCroppingRegionFlagsMaxValue ()` - Set the flags for the cropping regions. The clipping planes divide the volume into 27 regions - there is one bit for each region. The regions start from the one containing voxel (0,0,0), moving along the x axis fastest, the y axis next, and the z axis slowest. These are represented from the lowest bit to bit number 27 in the integer containing the flags. There are several convenience functions to set some common configurations - subvolume (the default), fence (between any of the clip plane pairs), inverted fence, cross (between any two of the clip plane pairs) and inverted cross.
- `int = obj.GetCroppingRegionFlags ()` - Set the flags for the cropping regions. The clipping planes divide the volume into 27 regions - there is one bit for each region. The regions start from the one containing voxel (0,0,0), moving along the x axis fastest, the y axis next, and the z axis slowest. These are represented from the lowest bit to bit number 27 in the integer containing the flags. There are several convenience functions to set some common configurations - subvolume (the default), fence (between any of the clip plane pairs), inverted fence, cross (between any two of the clip plane pairs) and inverted cross.
- `obj.SetCroppingRegionFlagsToSubVolume ()` - Set the flags for the cropping regions. The clipping planes divide the volume into 27 regions - there is one bit for each region. The regions start from the one containing voxel (0,0,0), moving along the x axis fastest, the y axis next, and the z axis slowest. These are represented from the lowest bit to bit number 27 in the integer containing the flags. There are several convenience functions to set some common configurations - subvolume (the default), fence (between any of the clip plane pairs), inverted fence, cross (between any two of the clip plane pairs) and inverted cross.

- `obj.SetCroppingRegionFlagsToFence ()` - Set the flags for the cropping regions. The clipping planes divide the volume into 27 regions - there is one bit for each region. The regions start from the one containing voxel (0,0,0), moving along the x axis fastest, the y axis next, and the z axis slowest. These are represented from the lowest bit to bit number 27 in the integer containing the flags. There are several convenience functions to set some common configurations - subvolume (the default), fence (between any of the clip plane pairs), inverted fence, cross (between any two of the clip plane pairs) and inverted cross.
- `obj.SetCroppingRegionFlagsToInvertedFence ()` - Set the flags for the cropping regions. The clipping planes divide the volume into 27 regions - there is one bit for each region. The regions start from the one containing voxel (0,0,0), moving along the x axis fastest, the y axis next, and the z axis slowest. These are represented from the lowest bit to bit number 27 in the integer containing the flags. There are several convenience functions to set some common configurations - subvolume (the default), fence (between any of the clip plane pairs), inverted fence, cross (between any two of the clip plane pairs) and inverted cross.
- `obj.SetCroppingRegionFlagsToCross ()` - Set the flags for the cropping regions. The clipping planes divide the volume into 27 regions - there is one bit for each region. The regions start from the one containing voxel (0,0,0), moving along the x axis fastest, the y axis next, and the z axis slowest. These are represented from the lowest bit to bit number 27 in the integer containing the flags. There are several convenience functions to set some common configurations - subvolume (the default), fence (between any of the clip plane pairs), inverted fence, cross (between any two of the clip plane pairs) and inverted cross.
- `obj.SetCroppingRegionFlagsToInvertedCross ()` - Set the flags for the cropping regions. The clipping planes divide the volume into 27 regions - there is one bit for each region. The regions start from the one containing voxel (0,0,0), moving along the x axis fastest, the y axis next, and the z axis slowest. These are represented from the lowest bit to bit number 27 in the integer containing the flags. There are several convenience functions to set some common configurations - subvolume (the default), fence (between any of the clip plane pairs), inverted fence, cross (between any two of the clip plane pairs) and inverted cross.

41.36 vtkVolumeOutlineSource

41.36.1 Usage

vtkVolumeOutlineSource generates a wireframe outline that corresponds to the cropping region of a vtkVolumeMapper. It requires a vtkVolumeMapper as input. The GenerateFaces option turns on the solid faces of the outline, and the GenerateScalars option generates color scalars. When GenerateScalars is on, it is possible to set an "ActivePlaneId" value in the range [0..6] to highlight one of the six cropping planes. .SECTION Thanks Thanks to David Gobbi for contributing this class to VTK.

To create an instance of class vtkVolumeOutlineSource, simply invoke its constructor as follows

```
obj = vtkVolumeOutlineSource
```

41.36.2 Methods

The class vtkVolumeOutlineSource has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeOutlineSource class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeOutlineSource = obj.NewInstance ()`

- `vtkVolumeOutlineSource = obj.SafeDownCast (vtkObject o)`
- `obj.SetVolumeMapper (vtkVolumeMapper mapper)` - Set the mapper that has the cropping region that the outline will be generated for. The mapper must have an input, because the bounds of the data must be computed in order to generate the outline.
- `vtkVolumeMapper = obj.GetVolumeMapper ()` - Set the mapper that has the cropping region that the outline will be generated for. The mapper must have an input, because the bounds of the data must be computed in order to generate the outline.
- `obj.SetGenerateScalars (int )` - Set whether to generate color scalars for the output. By default, the output has no scalars and the color must be set in the property of the actor.
- `obj.GenerateScalarsOn ()` - Set whether to generate color scalars for the output. By default, the output has no scalars and the color must be set in the property of the actor.
- `obj.GenerateScalarsOff ()` - Set whether to generate color scalars for the output. By default, the output has no scalars and the color must be set in the property of the actor.
- `int = obj.GetGenerateScalars ()` - Set whether to generate color scalars for the output. By default, the output has no scalars and the color must be set in the property of the actor.
- `obj.SetGenerateFaces (int )` - Set whether to generate polygonal faces for the output. By default, only lines are generated. The faces will form a closed, watertight surface.
- `obj.GenerateFacesOn ()` - Set whether to generate polygonal faces for the output. By default, only lines are generated. The faces will form a closed, watertight surface.
- `obj.GenerateFacesOff ()` - Set whether to generate polygonal faces for the output. By default, only lines are generated. The faces will form a closed, watertight surface.
- `int = obj.GetGenerateFaces ()` - Set whether to generate polygonal faces for the output. By default, only lines are generated. The faces will form a closed, watertight surface.
- `obj.SetColor (double , double , double )` - Set the color of the outline. This has no effect unless GenerateScalars is On. The default color is red.
- `obj.SetColor (double  a[3])` - Set the color of the outline. This has no effect unless GenerateScalars is On. The default color is red.
- `double = obj. GetColor ()` - Set the color of the outline. This has no effect unless GenerateScalars is On. The default color is red.
- `obj.SetActivePlaneId (int )` - Set the active plane, e.g. to display which plane is currently being modified by an interaction. Set this to -1 if there is no active plane. The default value is -1.
- `int = obj.GetActivePlaneId ()` - Set the active plane, e.g. to display which plane is currently being modified by an interaction. Set this to -1 if there is no active plane. The default value is -1.
- `obj.SetActivePlaneColor (double , double , double )` - Set the color of the active cropping plane. This has no effect unless GenerateScalars is On and ActivePlaneId is non-negative. The default color is yellow.
- `obj.SetActivePlaneColor (double  a[3])` - Set the color of the active cropping plane. This has no effect unless GenerateScalars is On and ActivePlaneId is non-negative. The default color is yellow.
- `double = obj. GetActivePlaneColor ()` - Set the color of the active cropping plane. This has no effect unless GenerateScalars is On and ActivePlaneId is non-negative. The default color is yellow.

41.37 vtkVolumePicker

41.37.1 Usage

vtkVolumePicker is a subclass of vtkCellPicker. It has one advantage over vtkCellPicker for volumes: it will be able to correctly perform picking when CroppingPlanes are present. This isn't possible for vtkCellPicker since it doesn't link to the VolumeRendering classes and hence cannot access information about the CroppingPlanes.

To create an instance of class vtkVolumePicker, simply invoke its constructor as follows

```
obj = vtkVolumePicker
```

41.37.2 Methods

The class vtkVolumePicker has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumePicker class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumePicker = obj.NewInstance ()`
- `vtkVolumePicker = obj.SafeDownCast (vtkObject o)`
- `obj.SetPickCroppingPlanes (int )` - Set whether to pick the cropping planes of props that have them. If this is set, then the pick will be done on the cropping planes rather than on the data. The GetCroppingPlaneId() method will return the index of the cropping plane of the volume that was picked. This setting is only relevant to the picking of volumes.
- `obj.PickCroppingPlanesOn ()` - Set whether to pick the cropping planes of props that have them. If this is set, then the pick will be done on the cropping planes rather than on the data. The GetCroppingPlaneId() method will return the index of the cropping plane of the volume that was picked. This setting is only relevant to the picking of volumes.
- `obj.PickCroppingPlanesOff ()` - Set whether to pick the cropping planes of props that have them. If this is set, then the pick will be done on the cropping planes rather than on the data. The GetCroppingPlaneId() method will return the index of the cropping plane of the volume that was picked. This setting is only relevant to the picking of volumes.
- `int = obj.GetPickCroppingPlanes ()` - Set whether to pick the cropping planes of props that have them. If this is set, then the pick will be done on the cropping planes rather than on the data. The GetCroppingPlaneId() method will return the index of the cropping plane of the volume that was picked. This setting is only relevant to the picking of volumes.
- `int = obj.GetCroppingPlaneId ()` - Get the index of the cropping plane that the pick ray passed through on its way to the prop. This will be set regardless of whether PickCroppingPlanes is on. The crop planes are ordered as follows: xmin, xmax, ymin, ymax, zmin, zmax. If the volume is not cropped, the value will bet set to -1.

41.38 vtkVolumeProMapper

41.38.1 Usage

vtkVolumeProMapper is the superclass for VolumePRO volume rendering mappers. Any functionality that is general across all VolumePRO implementations is placed here in this class. Subclasses of this class are for

the specific board implementations. Subclasses of that are for underlying graphics languages. Users should not create subclasses directly - a vtkVolumeProMapper will automatically create the object of the right type.

If you do not have the VolumePRO libraries when building this object, then the New method will create a default renderer that will not render. You can check the NumberOfBoards ivar to see if it is a real rendering class. To build with the VolumePRO board see vtkVolumeProVP1000Mapper.h for instructions.

For more information on the VolumePRO hardware, please see:

http://www.terarecon.com/products/volumepro_prod.html

If you encounter any problems with this class, please inform Kitware, Inc. at kitware@kitware.com.

To create an instance of class vtkVolumeProMapper, simply invoke its constructor as follows

```
obj = vtkVolumeProMapper
```

41.38.2 Methods

The class vtkVolumeProMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeProMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeProMapper = obj.NewInstance ()`
- `vtkVolumeProMapper = obj.SafeDownCast (vtkObject o)`
- `obj.Render (vtkRenderer , vtkVolume )` - Set the blend mode
- `obj.SetBlendMode (int )` - Set the blend mode
- `int = obj.GetBlendModeMinValue ()` - Set the blend mode
- `int = obj.GetBlendModeMaxValue ()` - Set the blend mode
- `int = obj.GetBlendMode ()` - Set the blend mode
- `obj.SetBlendModeToComposite ()` - Set the blend mode
- `obj.SetBlendModeToMaximumIntensity ()` - Set the blend mode
- `obj.SetBlendModeToMinimumIntensity ()` - Set the blend mode
- `string = obj.GetBlendModeAsString (void )` - Set the blend mode
- `obj.SetSubVolume (int , int , int , int , int , int )` - Set the subvolume
- `obj.SetSubVolume (int  a[6])` - Set the subvolume
- `int = obj. GetSubVolume ()` - Set the subvolume
- `obj.SetCursor (int )` - Turn the cursor on / off
- `int = obj.GetCursorMinValue ()` - Turn the cursor on / off
- `int = obj.GetCursorMaxValue ()` - Turn the cursor on / off
- `int = obj.GetCursor ()` - Turn the cursor on / off
- `obj.CursorOn ()` - Turn the cursor on / off
- `obj.CursorOff ()` - Turn the cursor on / off

- `obj.SetCursorType (int )` - Set the type of the cursor
- `int = obj.GetCursorTypeMinValue ()` - Set the type of the cursor
- `int = obj.GetCursorTypeMaxValue ()` - Set the type of the cursor
- `int = obj.GetCursorType ()` - Set the type of the cursor
- `obj.SetCursorTypeToCrossHair ()` - Set the type of the cursor
- `obj.SetCursorTypeToPlane ()` - Set the type of the cursor
- `string = obj.GetCursorTypeAsString (void )` - Set the type of the cursor
- `obj.SetCursorPosition (double , double , double )` - Set/Get the cursor position
- `obj.SetCursorPosition (double  a[3])` - Set/Get the cursor position
- `double = obj. GetCursorPosition ()` - Set/Get the cursor position
- `obj.SetCursorXAxisColor (double , double , double )` - Set/Get the cursor color
- `obj.SetCursorXAxisColor (double  a[3])` - Set/Get the cursor color
- `double = obj. GetCursorXAxisColor ()` - Set/Get the cursor color
- `obj.SetCursorYAxisColor (double , double , double )` - Set/Get the cursor color
- `obj.SetCursorYAxisColor (double  a[3])` - Set/Get the cursor color
- `double = obj. GetCursorYAxisColor ()` - Set/Get the cursor color
- `obj.SetCursorZAxisColor (double , double , double )` - Set/Get the cursor color
- `obj.SetCursorZAxisColor (double  a[3])` - Set/Get the cursor color
- `double = obj. GetCursorZAxisColor ()` - Set/Get the cursor color
- `obj.SetSuperSampling (int )` - Turn supersampling on/off
- `int = obj.GetSuperSamplingMinValue ()` - Turn supersampling on/off
- `int = obj.GetSuperSamplingMaxValue ()` - Turn supersampling on/off
- `int = obj.GetSuperSampling ()` - Turn supersampling on/off
- `obj.SuperSamplingOn ()` - Turn supersampling on/off
- `obj.SuperSamplingOff ()` - Turn supersampling on/off
- `obj.SetSuperSamplingFactor (double x, double y, double z)` - Set the supersampling factors
- `obj.SetSuperSamplingFactor (double f[3])` - Set the supersampling factors
- `double = obj. GetSuperSamplingFactor ()` - Set the supersampling factors
- `obj.SetCutPlane (int )` - Turn on / off the cut plane
- `int = obj.GetCutPlaneMinValue ()` - Turn on / off the cut plane
- `int = obj.GetCutPlaneMaxValue ()` - Turn on / off the cut plane
- `int = obj.GetCutPlane ()` - Turn on / off the cut plane
- `obj.CutPlaneOn ()` - Turn on / off the cut plane

- `obj.CutPlaneOff ()` - Turn on / off the cut plane
- `obj.SetCutPlaneEquation (double , double , double , double )` - Set/Get the cut plane equation
- `obj.SetCutPlaneEquation (double a[4])` - Set/Get the cut plane equation
- `double = obj. GetCutPlaneEquation ()` - Set/Get the cut plane equation
- `obj.SetCutPlaneThickness (double )` - Set / Get the cut plane thickness
- `double = obj.GetCutPlaneThicknessMinValue ()` - Set / Get the cut plane thickness
- `double = obj.GetCutPlaneThicknessMaxValue ()` - Set / Get the cut plane thickness
- `double = obj.GetCutPlaneThickness ()` - Set / Get the cut plane thickness
- `obj.SetCutPlaneFallOffDistance (int )` - Set / Get the cut plane falloff value for intensities
- `int = obj.GetCutPlaneFallOffDistanceMinValue ()` - Set / Get the cut plane falloff value for intensities
- `int = obj.GetCutPlaneFallOffDistanceMaxValue ()` - Set / Get the cut plane falloff value for intensities
- `int = obj.GetCutPlaneFallOffDistance ()` - Set / Get the cut plane falloff value for intensities
- `obj.SetGradientOpacityModulation (int )` - Set/Get the gradient magnitude opacity modulation
- `int = obj.GetGradientOpacityModulationMinValue ()` - Set/Get the gradient magnitude opacity modulation
- `int = obj.GetGradientOpacityModulationMaxValue ()` - Set/Get the gradient magnitude opacity modulation
- `int = obj.GetGradientOpacityModulation ()` - Set/Get the gradient magnitude opacity modulation
- `obj.GradientOpacityModulationOn ()` - Set/Get the gradient magnitude opacity modulation
- `obj.GradientOpacityModulationOff ()` - Set/Get the gradient magnitude opacity modulation
- `obj.SetGradientDiffuseModulation (int )` - Set/Get the gradient magnitude diffuse modulation
- `int = obj.GetGradientDiffuseModulationMinValue ()` - Set/Get the gradient magnitude diffuse modulation
- `int = obj.GetGradientDiffuseModulationMaxValue ()` - Set/Get the gradient magnitude diffuse modulation
- `int = obj.GetGradientDiffuseModulation ()` - Set/Get the gradient magnitude diffuse modulation
- `obj.GradientDiffuseModulationOn ()` - Set/Get the gradient magnitude diffuse modulation
- `obj.GradientDiffuseModulationOff ()` - Set/Get the gradient magnitude diffuse modulation
- `obj.SetGradientSpecularModulation (int )` - Set/Get the gradient magnitude specular modulation
- `int = obj.GetGradientSpecularModulationMinValue ()` - Set/Get the gradient magnitude specular modulation
- `int = obj.GetGradientSpecularModulationMaxValue ()` - Set/Get the gradient magnitude specular modulation

- `int = obj.GetGradientSpecularModulation ()` - Set/Get the gradient magnitude specular modulation
- `obj.GradientSpecularModulationOn ()` - Set/Get the gradient magnitude specular modulation
- `obj.GradientSpecularModulationOff ()` - Set/Get the gradient magnitude specular modulation
- `int = obj.GetNoHardware ()` - Conveniece methods for debugging
- `int = obj.GetWrongVLIVersion ()` - Conveniece methods for debugging
- `int = obj.GetNumberOfBoards ()` - Access methods for some board info
- `int = obj.GetMajorBoardVersion ()` - Access methods for some board info
- `int = obj.GetMinorBoardVersion ()` - Access methods for some board info
- `int = obj.GetAvailableBoardMemory ()` - Access methods for some board info
- `obj.GetLockSizesForBoardMemory (int , int , int , int )` - Access methods for some board info
- `obj.SetIntermixIntersectingGeometry (int )` - Specify whether any geometry intersects the volume.
- `int = obj.GetIntermixIntersectingGeometryMinValue ()` - Specify whether any geometry intersects the volume.
- `int = obj.GetIntermixIntersectingGeometryMaxValue ()` - Specify whether any geometry intersects the volume.
- `int = obj.GetIntermixIntersectingGeometry ()` - Specify whether any geometry intersects the volume.
- `obj.IntermixIntersectingGeometryOn ()` - Specify whether any geometry intersects the volume.
- `obj.IntermixIntersectingGeometryOff ()` - Specify whether any geometry intersects the volume.
- `obj.SetAutoAdjustMipmapLevels (int )` - If set to 1, this mapper will select a mipmap level to use based on the AllocatedRenderTime of the volume and the amount of time used by the previous render.
- `int = obj.GetAutoAdjustMipmapLevelsMinValue ()` - If set to 1, this mapper will select a mipmap level to use based on the AllocatedRenderTime of the volume and the amount of time used by the previous render.
- `int = obj.GetAutoAdjustMipmapLevelsMaxValue ()` - If set to 1, this mapper will select a mipmap level to use based on the AllocatedRenderTime of the volume and the amount of time used by the previous render.
- `int = obj.GetAutoAdjustMipmapLevels ()` - If set to 1, this mapper will select a mipmap level to use based on the AllocatedRenderTime of the volume and the amount of time used by the previous render.
- `obj.AutoAdjustMipmapLevelsOn ()` - If set to 1, this mapper will select a mipmap level to use based on the AllocatedRenderTime of the volume and the amount of time used by the previous render.
- `obj.AutoAdjustMipmapLevelsOff ()` - If set to 1, this mapper will select a mipmap level to use based on the AllocatedRenderTime of the volume and the amount of time used by the previous render.
- `obj.SetMinimumMipmapLevel (int )` - Specify the minimum mipmap level to use – the highest resolution. Defaults to 0. This is the mipmap level that is used when interaction stops.

- `int = obj.GetMinimumMipmapLevelMinValue ()` - Specify the minimum mipmap level to use – the highest resolution. Defaults to 0. This is the mipmap level that is used when interaction stops.

- `int = obj.GetMinimumMipmapLevelMaxValue ()` - Specify the minimum mipmap level to use – the highest resolution. Defaults to 0. This is the mipmap level that is used when interaction stops.

- `int = obj.GetMinimumMipmapLevel ()` - Specify the minimum mipmap level to use – the highest resolution. Defaults to 0. This is the mipmap level that is used when interaction stops.

- `obj.SetMaximumMipmapLevel (int )` - Specify the maximum mipmap level to use – the lowest resolution. Defaults to 4. It will not help to set the level larger than this unless your volume is very large because for each successive mipmap level, the number of voxels along each axis is cut in half.

- `int = obj.GetMaximumMipmapLevelMinValue ()` - Specify the maximum mipmap level to use – the lowest resolution. Defaults to 4. It will not help to set the level larger than this unless your volume is very large because for each successive mipmap level, the number of voxels along each axis is cut in half.

- `int = obj.GetMaximumMipmapLevelMaxValue ()` - Specify the maximum mipmap level to use – the lowest resolution. Defaults to 4. It will not help to set the level larger than this unless your volume is very large because for each successive mipmap level, the number of voxels along each axis is cut in half.

- `int = obj.GetMaximumMipmapLevel ()` - Specify the maximum mipmap level to use – the lowest resolution. Defaults to 4. It will not help to set the level larger than this unless your volume is very large because for each successive mipmap level, the number of voxels along each axis is cut in half.

- `obj.SetMipmapLevel (int )` - Choose a mipmap level. If AutoAdjustMipmapLevels is off, then this specifies the mipmap level to use during interaction. If AutoAdjustMipmapLevels is on, then this specifies the initial mipmap level to use.

- `int = obj.GetMipmapLevelMinValue ()` - Choose a mipmap level. If AutoAdjustMipmapLevels is off, then this specifies the mipmap level to use during interaction. If AutoAdjustMipmapLevels is on, then this specifies the initial mipmap level to use.

- `int = obj.GetMipmapLevelMaxValue ()` - Choose a mipmap level. If AutoAdjustMipmapLevels is off, then this specifies the mipmap level to use during interaction. If AutoAdjustMipmapLevels is on, then this specifies the initial mipmap level to use.

- `int = obj.GetMipmapLevel ()` - Choose a mipmap level. If AutoAdjustMipmapLevels is off, then this specifies the mipmap level to use during interaction. If AutoAdjustMipmapLevels is on, then this specifies the initial mipmap level to use.

41.39 vtkVolumeRayCastCompositeFunction

41.39.1 Usage

vtkVolumeRayCastCompositeFunction is a ray function that can be used within a vtkVolumeRayCastMapper. This function performs compositing along the ray according to the properties stored in the vtkVolumeProperty for the volume.

To create an instance of class vtkVolumeRayCastCompositeFunction, simply invoke its constructor as follows

```
obj = vtkVolumeRayCastCompositeFunction
```

41.39.2 Methods

The class vtkVolumeRayCastCompositeFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeRayCastCompositeFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeRayCastCompositeFunction = obj.NewInstance ()`
- `vtkVolumeRayCastCompositeFunction = obj.SafeDownCast (vtkObject o)`
- `obj.SetCompositeMethod (int )` - Set the CompositeMethod to either Classify First or Interpolate First
- `int = obj.GetCompositeMethodMinValue ()` - Set the CompositeMethod to either Classify First or Interpolate First
- `int = obj.GetCompositeMethodMaxValue ()` - Set the CompositeMethod to either Classify First or Interpolate First
- `int = obj.GetCompositeMethod ()` - Set the CompositeMethod to either Classify First or Interpolate First
- `obj.SetCompositeMethodToInterpolateFirst ()` - Set the CompositeMethod to either Classify First or Interpolate First
- `obj.SetCompositeMethodToClassifyFirst ()` - Set the CompositeMethod to either Classify First or Interpolate First
- `string = obj.GetCompositeMethodAsString (void )` - Set the CompositeMethod to either Classify First or Interpolate First

41.40 vtkVolumeRayCastFunction

41.40.1 Usage

vtkVolumeRayCastFunction is a superclass for ray casting functions that can be used within a vtkVolumeRayCastMapper. This includes for example, vtkVolumeRayCastCompositeFunction, vtkVolumeRayCastMIPFunction, and vtkVolumeRayCastIsosurfaceFunction.

To create an instance of class vtkVolumeRayCastFunction, simply invoke its constructor as follows

```
obj = vtkVolumeRayCastFunction
```

41.40.2 Methods

The class vtkVolumeRayCastFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeRayCastFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeRayCastFunction = obj.NewInstance ()`

- `vtkVolumeRayCastFunction = obj.SafeDownCast (vtkObject o)`
- `float = obj.GetZeroOpacityThreshold (vtkVolume vol)` - Get the value below which all scalar values are considered to have 0 opacity.

41.41 vtkVolumeRayCastIsosurfaceFunction

41.41.1 Usage

vtkVolumeRayCastIsosurfaceFunction is a volume ray cast function that intersects a ray with an analytic isosurface in a scalar field. The color and shading parameters are defined in the vtkVolumeProperty of the vtkVolume, as well as the interpolation type to use when locating the surface (either a nearest neighbor approach or a tri-linear interpolation approach)

To create an instance of class vtkVolumeRayCastIsosurfaceFunction, simply invoke its constructor as follows

```
obj = vtkVolumeRayCastIsosurfaceFunction
```

41.41.2 Methods

The class vtkVolumeRayCastIsosurfaceFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeRayCastIsosurfaceFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeRayCastIsosurfaceFunction = obj.NewInstance ()`
- `vtkVolumeRayCastIsosurfaceFunction = obj.SafeDownCast (vtkObject o)`
- `float = obj.GetZeroOpacityThreshold (vtkVolume vol)` - Get the scalar value below which all scalar values have 0 opacity
- `obj.SetIsoValue (double )` - Set/Get the value of IsoValue.
- `double = obj.GetIsoValue ()` - Set/Get the value of IsoValue.

41.42 vtkVolumeRayCastMapper

41.42.1 Usage

This is a software ray caster for rendering volumes in vtkImageData.

To create an instance of class vtkVolumeRayCastMapper, simply invoke its constructor as follows

```
obj = vtkVolumeRayCastMapper
```

41.42.2 Methods

The class vtkVolumeRayCastMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeRayCastMapper class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkVolumeRayCastMapper = obj.NewInstance ()`
- `vtkVolumeRayCastMapper = obj.SafeDownCast (vtkObject o)`
- `obj.SetSampleDistance (double )` - Set/Get the distance between samples. This variable is only used for sampling ray casting methods. Methods that compute a ray value by stepping cell-by-cell are not affected by this value.
- `double = obj.GetSampleDistance ()` - Set/Get the distance between samples. This variable is only used for sampling ray casting methods. Methods that compute a ray value by stepping cell-by-cell are not affected by this value.
- `obj.SetVolumeRayCastFunction (vtkVolumeRayCastFunction )` - Get / Set the volume ray cast function. This is used to process values found along the ray to compute a final pixel value.
- `vtkVolumeRayCastFunction = obj.GetVolumeRayCastFunction ()` - Get / Set the volume ray cast function. This is used to process values found along the ray to compute a final pixel value.
- `obj.SetGradientEstimator (vtkEncodedGradientEstimator gradest)` - Set / Get the gradient estimator used to estimate normals
- `vtkEncodedGradientEstimator = obj.GetGradientEstimator ()` - Set / Get the gradient estimator used to estimate normals
- `vtkEncodedGradientShader = obj.GetGradientShader ()` - Get the gradient shader.
- `obj.SetImageSampleDistance (double )` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `double = obj.GetImageSampleDistanceMinValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `double = obj.GetImageSampleDistanceMaxValue ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `double = obj.GetImageSampleDistance ()` - Sampling distance in the XY image dimensions. Default value of 1 meaning 1 ray cast per pixel. If set to 0.5, 4 rays will be cast per pixel. If set to 2.0, 1 ray will be cast for every 4 (2 by 2) pixels.
- `obj.SetMinimumImageSampleDistance (double )` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `double = obj.GetMinimumImageSampleDistanceMinValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `double = obj.GetMinimumImageSampleDistanceMaxValue ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `double = obj.GetMinimumImageSampleDistance ()` - This is the minimum image sample distance allow when the image sample distance is being automatically adjusted
- `obj.SetMaximumImageSampleDistance (double )` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted
- `double = obj.GetMaximumImageSampleDistanceMinValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted

- `double = obj.GetMaximumImageSampleDistanceMaxValue ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted

- `double = obj.GetMaximumImageSampleDistance ()` - This is the maximum image sample distance allow when the image sample distance is being automatically adjusted

- `obj.SetAutoAdjustSampleDistances (int )` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).

- `int = obj.GetAutoAdjustSampleDistancesMinValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).

- `int = obj.GetAutoAdjustSampleDistancesMaxValue ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).

- `int = obj.GetAutoAdjustSampleDistances ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).

- `obj.AutoAdjustSampleDistancesOn ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).

- `obj.AutoAdjustSampleDistancesOff ()` - If AutoAdjustSampleDistances is on, the the ImageSampleDistance will be varied to achieve the allocated render time of this prop (controlled by the desired update rate and any culling in use).

- `obj.SetNumberOfThreads (int num)` - Set/Get the number of threads to use. This by default is equal to the number of available processors detected.

- `int = obj.GetNumberOfThreads ()` - Set/Get the number of threads to use. This by default is equal to the number of available processors detected.

- `obj.SetIntermixIntersectingGeometry (int )` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.

- `int = obj.GetIntermixIntersectingGeometryMinValue ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.

- `int = obj.GetIntermixIntersectingGeometryMaxValue ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.

- `int = obj.GetIntermixIntersectingGeometry ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.

- `obj.IntermixIntersectingGeometryOn ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.

- `obj.IntermixIntersectingGeometryOff ()` - If IntermixIntersectingGeometry is turned on, the zbuffer will be captured and used to limit the traversal of the rays.

41.43 vtkVolumeRayCastMIPFunction

41.43.1 Usage

vtkVolumeRayCastMIPFunction is a volume ray cast function that computes the maximum value encountered along the ray. This is either the maximum scalar value, or the maximum opacity, as defined by the MaximizeMethod. The color and opacity returned by this function is based on the color, scalar opacity, and gradient opacity transfer functions defined in the vtkVolumeProperty of the vtkVolume.

To create an instance of class vtkVolumeRayCastMIPFunction, simply invoke its constructor as follows

```
obj = vtkVolumeRayCastMIPFunction
```

41.43.2 Methods

The class vtkVolumeRayCastMIPFunction has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeRayCastMIPFunction class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeRayCastMIPFunction = obj.NewInstance ()`
- `vtkVolumeRayCastMIPFunction = obj.SafeDownCast (vtkObject o)`
- `float = obj.GetZeroOpacityThreshold (vtkVolume vol)` - Get the scalar value below which all scalar values have zero opacity.
- `obj.SetMaximizeMethod (int )` - Set the MaximizeMethod to either ScalarValue or Opacity.
- `int = obj.GetMaximizeMethodMinValue ()` - Set the MaximizeMethod to either ScalarValue or Opacity.
- `int = obj.GetMaximizeMethodMaxValue ()` - Set the MaximizeMethod to either ScalarValue or Opacity.
- `int = obj.GetMaximizeMethod ()` - Set the MaximizeMethod to either ScalarValue or Opacity.
- `obj.SetMaximizeMethodToScalarValue ()` - Set the MaximizeMethod to either ScalarValue or Opacity.
- `obj.SetMaximizeMethodToOpacity ()` - Set the MaximizeMethod to either ScalarValue or Opacity.
- `string = obj.GetMaximizeMethodAsString (void )` - Set the MaximizeMethod to either ScalarValue or Opacity.

41.44 vtkVolumeRenderingFactory

41.44.1 Usage

To create an instance of class vtkVolumeRenderingFactory, simply invoke its constructor as follows

```
obj = vtkVolumeRenderingFactory
```

41.44.2 Methods

The class vtkVolumeRenderingFactory has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeRenderingFactory class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeRenderingFactory = obj.NewInstance ()`
- `vtkVolumeRenderingFactory = obj.SafeDownCast (vtkObject o)`

41.45 vtkVolumeTextureMapper

41.45.1 Usage

vtkVolumeTextureMapper is the abstract definition of a volume mapper that uses a texture mapping approach.

To create an instance of class vtkVolumeTextureMapper, simply invoke its constructor as follows

```
obj = vtkVolumeTextureMapper
```

41.45.2 Methods

The class vtkVolumeTextureMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeTextureMapper class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeTextureMapper = obj.NewInstance ()`
- `vtkVolumeTextureMapper = obj.SafeDownCast (vtkObject o)`
- `obj.Update ()` - Update the volume rendering pipeline by updating the scalar input
- `obj.SetGradientEstimator (vtkEncodedGradientEstimator gradest)` - Set / Get the gradient estimator used to estimate normals
- `vtkEncodedGradientEstimator = obj.GetGradientEstimator ()` - Set / Get the gradient estimator used to estimate normals
- `vtkEncodedGradientShader = obj.GetGradientShader ()` - Get the gradient shader.

41.46 vtkVolumeTextureMapper2D

41.46.1 Usage

vtkVolumeTextureMapper2D renders a volume using 2D texture mapping.

To create an instance of class vtkVolumeTextureMapper2D, simply invoke its constructor as follows

```
obj = vtkVolumeTextureMapper2D
```

41.46.2 Methods

The class vtkVolumeTextureMapper2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeTextureMapper2D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeTextureMapper2D = obj.NewInstance ()`
- `vtkVolumeTextureMapper2D = obj.SafeDownCast (vtkObject o)`
- `obj.SetTargetTextureSize (int , int )` - Target size in pixels of each size of the texture for downloading. Default is 512x512 - so a 512x512 texture will be tiled with as many slices of the volume as possible, then all the quads will be rendered. This can be set to optimize for a particular architecture. This must be set with numbers that are a power of two.
- `obj.SetTargetTextureSize (int a[2])` - Target size in pixels of each size of the texture for downloading. Default is 512x512 - so a 512x512 texture will be tiled with as many slices of the volume as possible, then all the quads will be rendered. This can be set to optimize for a particular architecture. This must be set with numbers that are a power of two.
- `int = obj. GetTargetTextureSize ()` - Target size in pixels of each size of the texture for downloading. Default is 512x512 - so a 512x512 texture will be tiled with as many slices of the volume as possible, then all the quads will be rendered. This can be set to optimize for a particular architecture. This must be set with numbers that are a power of two.
- `obj.SetMaximumNumberOfPlanes (int )` - This is the maximum number of planes that will be created for texture mapping the volume. If the volume has more voxels than this along the viewing direction, then planes of the volume will be skipped to ensure that this maximum is not violated. A skip factor is used, and is incremented until the maximum condition is satisfied.
- `int = obj.GetMaximumNumberOfPlanes ()` - This is the maximum number of planes that will be created for texture mapping the volume. If the volume has more voxels than this along the viewing direction, then planes of the volume will be skipped to ensure that this maximum is not violated. A skip factor is used, and is incremented until the maximum condition is satisfied.
- `obj.SetMaximumStorageSize (int )` - This is the maximum size of saved textures in bytes. If this size is large enough to hold the RGBA textures for all three directions (XxYxZx3x4 is the approximate value - it is actually a bit larger due to wasted space in the textures) then the textures will be saved.
- `int = obj.GetMaximumStorageSize ()` - This is the maximum size of saved textures in bytes. If this size is large enough to hold the RGBA textures for all three directions (XxYxZx3x4 is the approximate value - it is actually a bit larger due to wasted space in the textures) then the textures will be saved.

41.47 vtkVolumeTextureMapper3D

41.47.1 Usage

vtkVolumeTextureMapper3D renders a volume using 3D texture mapping. This class is actually an abstract superclass - with all the actual work done by vtkOpenGLVolumeTextureMapper3D.

This mappers currently supports:

- any data type as input - one component, or two or four non-independent components - composite blending - intermixed opaque geometry - multiple volumes can be rendered if they can be sorted into back-to-front order (use the vtkFrustumCoverageCuller)

This mapper does not support: - more than one independent component - maximum intensity projection

Internally, this mapper will potentially change the resolution of the input data. The data will be resampled to be a power of two in each direction, and also no greater than 128*256*256 voxels (any aspect) for one or two component data, or 128*128*256 voxels (any aspect) for four component data. The limits are currently hardcoded after a check using the GL_PROXY_TEXTURE3D because some graphics drivers were always responding "yes" to the proxy call despite not being able to allocate that much texture memory.

Currently, calculations are computed using 8 bits per RGBA channel. In the future this should be expanded to handle newer boards that can support 15 bit float compositing.

This mapper supports two main families of graphics hardware: nvidia and ATI. There are two different implementations of 3D texture mapping used - one based on nvidia's GL_NV_texture_shader2 and GL_NV_register_combiners2 extension, and one based on ATI's GL_ATI_fragment_shader (supported also by some nvidia boards) To use this class in an application that will run on various hardware configurations, you should have a back-up volume rendering method. You should create a vtkVolumeTextureMapper3D, assign its input, make sure you have a current OpenGL context (you've rendered at least once), then call IsRenderSupported with a vtkVolumeProperty as an argument. This method will return 0 if the input has more than one independent component, or if the graphics hardware does not support the set of required extensions for using at least one of the two implemented methods (nvidia or ati)

.SECTION Thanks Thanks to Alexandre Gouaillard at the Megason Lab, Department of Systems Biology, Harvard Medical School https://wiki.med.harvard.edu/SysBio/Megason/ for the idea and initial patch to speed-up rendering with compressed textures.

To create an instance of class vtkVolumeTextureMapper3D, simply invoke its constructor as follows

```
obj = vtkVolumeTextureMapper3D
```

41.47.2 Methods

The class vtkVolumeTextureMapper3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkVolumeTextureMapper3D class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkVolumeTextureMapper3D = obj.NewInstance ()`
- `vtkVolumeTextureMapper3D = obj.SafeDownCast (vtkObject o)`
- `obj.SetSampleDistance (float )` - The distance at which to space sampling planes. This may not be honored for interactive renders. An interactive render is defined as one that has less than 1 second of allocated render time.
- `float = obj.GetSampleDistance ()` - The distance at which to space sampling planes. This may not be honored for interactive renders. An interactive render is defined as one that has less than 1 second of allocated render time.
- `int = obj. GetVolumeDimensions ()` - These are the dimensions of the 3D texture
- `float = obj. GetVolumeSpacing ()` - This is the spacing of the 3D texture
- `int = obj.IsRenderSupported (vtkVolumeProperty )` - Based on hardware and properties, we may or may not be able to render using 3D texture mapping. This indicates if 3D texture mapping is supported by the hardware, and if the other extensions necessary to support the specific properties are available.
- `int = obj.GetNumberOfPolygons ()` - Allow access to the number of polygons used for the rendering.

- `float = obj.GetActualSampleDistance ()` - Allow access to the actual sample distance used to render the image.
- `obj.SetPreferredRenderMethod (int )` - Set the preferred render method. If it is supported, this one will be used. Don't allow ATI_METHOD - it is not actually supported.
- `int = obj.GetPreferredRenderMethodMinValue ()` - Set the preferred render method. If it is supported, this one will be used. Don't allow ATI_METHOD - it is not actually supported.
- `int = obj.GetPreferredRenderMethodMaxValue ()` - Set the preferred render method. If it is supported, this one will be used. Don't allow ATI_METHOD - it is not actually supported.
- `obj.SetPreferredMethodToFragmentProgram ()` - Set the preferred render method. If it is supported, this one will be used. Don't allow ATI_METHOD - it is not actually supported.
- `obj.SetPreferredMethodToNVidia ()` - Set the preferred render method. If it is supported, this one will be used. Don't allow ATI_METHOD - it is not actually supported.
- `int = obj.GetPreferredRenderMethod ()` - Set the preferred render method. If it is supported, this one will be used. Don't allow ATI_METHOD - it is not actually supported.
- `obj.SetUseCompressedTexture (bool )` - Set/Get if the mapper use compressed textures (if supported by the hardware). Initial value is false. There are two reasons to use compressed textures: 1. rendering can be 4 times faster. 2. It saves some VRAM. There is one reason to not use compressed textures: quality may be lower than with uncompressed textures.
- `bool = obj.GetUseCompressedTexture ()` - Set/Get if the mapper use compressed textures (if supported by the hardware). Initial value is false. There are two reasons to use compressed textures: 1. rendering can be 4 times faster. 2. It saves some VRAM. There is one reason to not use compressed textures: quality may be lower than with uncompressed textures.

Chapter 42

Visualization Toolkit Widget Classes

42.1 vtk3DWidget

42.1.1 Usage

vtk3DWidget is an abstract superclass for 3D interactor observers. These 3D widgets represent themselves in the scene, and have special callbacks associated with them that allows interactive manipulation of the widget. Inparticular, the difference between a vtk3DWidget and its abstract superclass vtkInteractorObserver is that vtk3DWidgets are "placed" in 3D space. vtkInteractorObservers have no notion of where they are placed, and may not exist in 3D space at all. 3D widgets also provide auxiliary functions like producing a transformation, creating polydata (for seeding streamlines, probes, etc.) or creating implicit functions. See the concrete subclasses for particulars.

Typically the widget is used by specifying a vtkProp3D or VTK dataset as input, and then invoking the "On" method to activate it. (You can also specify a bounding box to help position the widget.) Prior to invoking the On() method, the user may also wish to use the PlaceWidget() to initially position it. The 'i' (for "interactor") keypresses also can be used to turn the widgets on and off (methods exist to change the key value and enable keypress activiation).

To support interactive manipulation of objects, this class (and subclasses) invoke the events StartInteractionEvent, InteractionEvent, and EndInteractionEvent. These events are invoked when the vtk3DWidget enters a state where rapid response is desired: mouse motion, etc. The events can be used, for example, to set the desired update frame rate (StartInteractionEvent), operate on the vtkProp3D or other object (InteractionEvent), and set the desired frame rate back to normal values (EndInteractionEvent).

Note that the Priority attribute inherited from vtkInteractorObserver has a new default value which is now 0.5 so that all 3D widgets have a higher priority than the usual interactor styles.

To create an instance of class vtk3DWidget, simply invoke its constructor as follows

```
obj = vtk3DWidget
```

42.1.2 Methods

The class vtk3DWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtk3DWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtk3DWidget = obj.NewInstance ()`
- `vtk3DWidget = obj.SafeDownCast (vtkObject o)`

- `obj.PlaceWidget (double bounds[6])` - This method is used to initially place the widget. The placement of the widget depends on whether a Prop3D or input dataset is provided. If one of these two is provided, they will be used to obtain a bounding box, around which the widget is placed. Otherwise, you can manually specify a bounds with the PlaceWidget(bounds) method. Note: PlaceWidget(bounds) is required by all subclasses; the other methods are provided as convenience methods.

- `obj.PlaceWidget ()` - This method is used to initially place the widget. The placement of the widget depends on whether a Prop3D or input dataset is provided. If one of these two is provided, they will be used to obtain a bounding box, around which the widget is placed. Otherwise, you can manually specify a bounds with the PlaceWidget(bounds) method. Note: PlaceWidget(bounds) is required by all subclasses; the other methods are provided as convenience methods.

- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - This method is used to initially place the widget. The placement of the widget depends on whether a Prop3D or input dataset is provided. If one of these two is provided, they will be used to obtain a bounding box, around which the widget is placed. Otherwise, you can manually specify a bounds with the PlaceWidget(bounds) method. Note: PlaceWidget(bounds) is required by all subclasses; the other methods are provided as convenience methods.

- `obj.SetProp3D (vtkProp3D )` - Specify a vtkProp3D around which to place the widget. This is not required, but if supplied, it is used to initially position the widget.

- `vtkProp3D = obj.GetProp3D ()` - Specify a vtkProp3D around which to place the widget. This is not required, but if supplied, it is used to initially position the widget.

- `obj.SetInput (vtkDataSet )` - Specify the input dataset. This is not required, but if supplied, and no vtkProp3D is specified, it is used to initially position the widget.

- `vtkDataSet = obj.GetInput ()` - Specify the input dataset. This is not required, but if supplied, and no vtkProp3D is specified, it is used to initially position the widget.

- `obj.SetPlaceFactor (double )` - Set/Get a factor representing the scaling of the widget upon placement (via the PlaceWidget() method). Normally the widget is placed so that it just fits within the bounding box defined in PlaceWidget(bounds). The PlaceFactor will make the widget larger (PlaceFactor ¿ 1) or smaller (PlaceFactor ¡ 1). By default, PlaceFactor is set to 0.5.

- `double = obj.GetPlaceFactorMinValue ()` - Set/Get a factor representing the scaling of the widget upon placement (via the PlaceWidget() method). Normally the widget is placed so that it just fits within the bounding box defined in PlaceWidget(bounds). The PlaceFactor will make the widget larger (PlaceFactor ¿ 1) or smaller (PlaceFactor ¡ 1). By default, PlaceFactor is set to 0.5.

- `double = obj.GetPlaceFactorMaxValue ()` - Set/Get a factor representing the scaling of the widget upon placement (via the PlaceWidget() method). Normally the widget is placed so that it just fits within the bounding box defined in PlaceWidget(bounds). The PlaceFactor will make the widget larger (PlaceFactor ¿ 1) or smaller (PlaceFactor ¡ 1). By default, PlaceFactor is set to 0.5.

- `double = obj.GetPlaceFactor ()` - Set/Get a factor representing the scaling of the widget upon placement (via the PlaceWidget() method). Normally the widget is placed so that it just fits within the bounding box defined in PlaceWidget(bounds). The PlaceFactor will make the widget larger (PlaceFactor ¿ 1) or smaller (PlaceFactor ¡ 1). By default, PlaceFactor is set to 0.5.

- `obj.SetHandleSize (double )` - Set/Get the factor that controls the size of the handles that appear as part of the widget. These handles (like spheres, etc.) are used to manipulate the widget, and are sized as a fraction of the screen diagonal.

- `double = obj.GetHandleSizeMinValue ()` - Set/Get the factor that controls the size of the handles that appear as part of the widget. These handles (like spheres, etc.) are used to manipulate the widget, and are sized as a fraction of the screen diagonal.

- `double = obj.GetHandleSizeMaxValue ()` - Set/Get the factor that controls the size of the handles that appear as part of the widget. These handles (like spheres, etc.) are used to manipulate the widget, and are sized as a fraction of the screen diagonal.
- `double = obj.GetHandleSize ()` - Set/Get the factor that controls the size of the handles that appear as part of the widget. These handles (like spheres, etc.) are used to manipulate the widget, and are sized as a fraction of the screen diagonal.

42.2 vtkAbstractPolygonalHandleRepresentation3D

42.2.1 Usage

This class serves as the geometrical representation of a vtkHandleWidget. The handle can be represented by an arbitrary polygonal data (vtkPolyData), set via SetHandle(vtkPolyData *). The actual position of the handle will be initially assumed to be (0,0,0). You can specify an offset from this position if desired. This class differs from vtkPolygonalHandleRepresentation3D in that the handle will always remain front facing, ie it maintains a fixed orientation with respect to the camera. This is done by using vtkFollowers internally to render the actors.

To create an instance of class vtkAbstractPolygonalHandleRepresentation3D, simply invoke its constructor as follows

```
obj = vtkAbstractPolygonalHandleRepresentation3D
```

42.2.2 Methods

The class vtkAbstractPolygonalHandleRepresentation3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractPolygonalHandleRepresentation3D class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkAbstractPolygonalHandleRepresentation3D = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkAbstractPolygonalHandleRepresentation3D = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetWorldPosition (double p[3])` - Set the position of the point in world and display coordinates.
- `obj.SetDisplayPosition (double p[3])` - Set the position of the point in world and display coordinates.
- `obj.SetHandle (vtkPolyData )` - Set/get the handle polydata.
- `vtkPolyData = obj.GetHandle ()` - Set/get the handle polydata.
- `obj.SetProperty (vtkProperty )` - Set/Get the handle properties when unselected and selected.
- `obj.SetSelectedProperty (vtkProperty )` - Set/Get the handle properties when unselected and selected.
- `vtkProperty = obj.GetProperty ()` - Set/Get the handle properties when unselected and selected.
- `vtkProperty = obj.GetSelectedProperty ()` - Set/Get the handle properties when unselected and selected.

- `vtkAbstractTransform = obj.GetTransform ()` - Get the transform used to transform the generic handle polydata before placing it in the render window
- `obj.BuildRepresentation ()` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.StartWidgetInteraction (double eventPos[2])` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.WidgetInteraction (double eventPos[2])` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.ShallowCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp.
- `obj.DeepCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp.
- `obj.GetActors (vtkPropCollection )` - Methods to make this class behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods to make this class behave as a vtkProp.
- `obj.SetLabelVisibility (int )` - A label may be associated with the seed. The string can be set via SetLabelText. The visibility of the label can be turned on / off.
- `int = obj.GetLabelVisibility ()` - A label may be associated with the seed. The string can be set via SetLabelText. The visibility of the label can be turned on / off.
- `obj.LabelVisibilityOn ()` - A label may be associated with the seed. The string can be set via SetLabelText. The visibility of the label can be turned on / off.
- `obj.LabelVisibilityOff ()` - A label may be associated with the seed. The string can be set via SetLabelText. The visibility of the label can be turned on / off.
- `obj.SetLabelText (string label)` - A label may be associated with the seed. The string can be set via SetLabelText. The visibility of the label can be turned on / off.
- `string = obj.GetLabelText ()` - A label may be associated with the seed. The string can be set via SetLabelText. The visibility of the label can be turned on / off.
- `obj.SetLabelTextScale (double scale[3])` - Scale text (font size along each dimension).
- `vtkFollower = obj.GetLabelTextActor ()` - Get the label text actor
- `obj.SetUniformScale (double scale)` - The handle may be scaled uniformly in all three dimensions using this API. The handle can also be scaled interactively using the right mouse button.
- `obj.SetHandleVisibility (int )` - Toogle the visibility of the handle on and off
- `int = obj.GetHandleVisibility ()` - Toogle the visibility of the handle on and off
- `obj.HandleVisibilityOn ()` - Toogle the visibility of the handle on and off
- `obj.HandleVisibilityOff ()` - Toogle the visibility of the handle on and off

42.3 vtkAbstractWidget

42.3.1 Usage

The vtkAbstractWidget defines an API and implements methods common to all widgets using the interaction/representation design. In this design, the term interaction means that part of the widget that performs event handling, while the representation corresponds to a vtkProp (or the subclass vtkWidgetRepresentation) used to represent the widget. vtkAbstractWidget also implements some methods common to all subclasses.

Note that vtkAbstractWidget provides access to the vtkWidgetEventTranslator. This class is responsible for translating VTK events (defined in vtkCommand.h) into widget events (defined in vtkWidgetEvent.h). This class can be manipulated so that different VTK events can be mapped into widget events, thereby allowing the modification of event bindings. Each subclass of vtkAbstractWidget defines the events to which it responds.

To create an instance of class vtkAbstractWidget, simply invoke its constructor as follows

```
obj = vtkAbstractWidget
```

42.3.2 Methods

The class vtkAbstractWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAbstractWidget class.

- `string = obj.GetClassName ()` - Standard macros implementing standard VTK methods.
- `int = obj.IsA (string name)` - Standard macros implementing standard VTK methods.
- `vtkAbstractWidget = obj.NewInstance ()` - Standard macros implementing standard VTK methods.
- `vtkAbstractWidget = obj.SafeDownCast (vtkObject o)` - Standard macros implementing standard VTK methods.
- `obj.SetEnabled (int )` - Methods for activating this widget. Note that the widget representation must be specified or the widget will not appear. ProcessEvents (On by default) must be On for Enabled widget to respond to interaction. If ProcessEvents is Off, enabling/disabling a widget merely affects the visibility of the representation.
- `obj.SetProcessEvents (int )` - Methods to change the whether the widget responds to interaction. Set this to Off to disable interaction. On by default. Subclasses must overide SetProcessEvents() to make sure that they pass on the flag to all component widgets.
- `int = obj.GetProcessEventsMinValue ()` - Methods to change the whether the widget responds to interaction. Set this to Off to disable interaction. On by default. Subclasses must overide SetProcessEvents() to make sure that they pass on the flag to all component widgets.
- `int = obj.GetProcessEventsMaxValue ()` - Methods to change the whether the widget responds to interaction. Set this to Off to disable interaction. On by default. Subclasses must overide SetProcessEvents() to make sure that they pass on the flag to all component widgets.
- `int = obj.GetProcessEvents ()` - Methods to change the whether the widget responds to interaction. Set this to Off to disable interaction. On by default. Subclasses must overide SetProcessEvents() to make sure that they pass on the flag to all component widgets.
- `obj.ProcessEventsOn ()` - Methods to change the whether the widget responds to interaction. Set this to Off to disable interaction. On by default. Subclasses must overide SetProcessEvents() to make sure that they pass on the flag to all component widgets.

- `obj.ProcessEventsOff ()` - Methods to change the whether the widget responds to interaction. Set this to Off to disable interaction. On by default. Subclasses must overide SetProcessEvents() to make sure that they pass on the flag to all component widgets.
- `vtkWidgetEventTranslator = obj.GetEventTranslator ()` - Create the default widget representation if one is not set. The representation defines the geometry of the widget (i.e., how it appears) as well as providing special methods for manipulting the state and appearance of the widget.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set. The representation defines the geometry of the widget (i.e., how it appears) as well as providing special methods for manipulting the state and appearance of the widget.
- `obj.Render ()` - This method is called by subclasses when a render method is to be invoked on the vtkRenderWindowInteractor. This method should be called (instead of vtkRenderWindow::Render() because it has built into it optimizations for minimizing renders and/or speeding renders.
- `obj.SetParent (vtkAbstractWidget parent)` - Specifying a parent to this widget is used when creating composite widgets. It is an internal method not meant to be used by the public. When a widget has a parent, it defers the rendering to the parent. It may also defer managing the cursor (see ManagesCursor ivar).
- `vtkAbstractWidget = obj.GetParent ()` - Specifying a parent to this widget is used when creating composite widgets. It is an internal method not meant to be used by the public. When a widget has a parent, it defers the rendering to the parent. It may also defer managing the cursor (see ManagesCursor ivar).
- `vtkWidgetRepresentation = obj.GetRepresentation ()` - Turn on or off the management of the cursor. Cursor management is typically disabled for subclasses when composite widgets are created. For example, vtkHandleWidgets are often used to create composite widgets, and the parent widget takes over the cursor management.
- `obj.SetManagesCursor (int )` - Turn on or off the management of the cursor. Cursor management is typically disabled for subclasses when composite widgets are created. For example, vtkHandleWidgets are often used to create composite widgets, and the parent widget takes over the cursor management.
- `int = obj.GetManagesCursor ()` - Turn on or off the management of the cursor. Cursor management is typically disabled for subclasses when composite widgets are created. For example, vtkHandleWidgets are often used to create composite widgets, and the parent widget takes over the cursor management.
- `obj.ManagesCursorOn ()` - Turn on or off the management of the cursor. Cursor management is typically disabled for subclasses when composite widgets are created. For example, vtkHandleWidgets are often used to create composite widgets, and the parent widget takes over the cursor management.
- `obj.ManagesCursorOff ()` - Turn on or off the management of the cursor. Cursor management is typically disabled for subclasses when composite widgets are created. For example, vtkHandleWidgets are often used to create composite widgets, and the parent widget takes over the cursor management.
- `obj.SetPriority (float )` - Override the superclass method. This will automatically change the priority of the widget. Unlike the superclass documentation, no methods such as SetInteractor to null and reset it etc. are necessary

42.4 vtkAffineRepresentation

42.4.1 Usage

This class defines an API for affine transformation widget representations. These representations interact with vtkAffineWidget. The basic functionality of the affine representation is to maintain a transformation matrix.

This class may be subclassed so that alternative representations can be created. The class defines an API and a default implementation that the vtkAffineWidget interacts with to render itself in the scene.

To create an instance of class vtkAffineRepresentation, simply invoke its constructor as follows

```
obj = vtkAffineRepresentation
```

42.4.2 Methods

The class vtkAffineRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAffineRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkAffineRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkAffineRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.GetTransform (vtkTransform t)` - Retrieve a linear transform characterizing the affine transformation generated by this widget. This method copies its internal transform into the transform provided. The transform is relative to the initial placement of the representation (i.e., when PlaceWidget() is invoked).
- `obj.SetTolerance (int )` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the widget to be active.
- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the widget to be active.
- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the widget to be active.
- `int = obj.GetTolerance ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the widget to be active.
- `obj.ShallowCopy (vtkProp prop)` - Methods to make this class properly act like a vtkWidgetRepresentation.

42.5 vtkAffineRepresentation2D

42.5.1 Usage

This class is used to represent a vtkAffineWidget. This representation consists of three parts: a box, a circle, and a cross. The box is used for scaling and shearing, the circle for rotation, and the cross for translation. These parts are drawn in the overlay plane and maintain a constant size (width and height) specified in terms of normalized viewport coordinates.

The representation maintains an internal transformation matrix (see superclass' GetTransform() method). The transformations generated by this widget assume that the representation lies in the x-y plane. If this is not the case, the user is responsible for transforming this representation's matrix into the correct coordinate space (by judicious matrix multiplication). Note that the transformation matrix returned by GetTransform() is relative to the last PlaceWidget() invocation. (The PlaceWidget() sets the origin around which rotation and scaling occurs; the origin is the center point of the bounding box provided.)

To create an instance of class vtkAffineRepresentation2D, simply invoke its constructor as follows

```
obj = vtkAffineRepresentation2D
```

42.5.2 Methods

The class vtkAffineRepresentation2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAffineRepresentation2D class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkAffineRepresentation2D = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkAffineRepresentation2D = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetBoxWidth (int )` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.
- `int = obj.GetBoxWidthMinValue ()` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.
- `int = obj.GetBoxWidthMaxValue ()` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.
- `int = obj.GetBoxWidth ()` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.
- `obj.SetCircleWidth (int )` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.
- `int = obj.GetCircleWidthMinValue ()` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.
- `int = obj.GetCircleWidthMaxValue ()` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.
- `int = obj.GetCircleWidth ()` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.
- `obj.SetAxesWidth (int )` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.

- `int = obj.GetAxesWidthMinValue ()` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.

- `int = obj.GetAxesWidthMaxValue ()` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.

- `int = obj.GetAxesWidth ()` - Specify the width of the various parts of the representation (in pixels). The three parts are of the representation are the translation axes, the rotation circle, and the scale/shear box. Note that since the widget resizes itself so that the width and height are always the same, only the width needs to be specified.

- `obj.SetOrigin (double o[3])` - Specify the origin of the widget (in world coordinates). The origin is the point where the widget places itself. Note that rotations and scaling occurs around the origin.

- `obj.SetOrigin (double ox, double oy, double oz)` - Specify the origin of the widget (in world coordinates). The origin is the point where the widget places itself. Note that rotations and scaling occurs around the origin.

- `double = obj. GetOrigin ()` - Specify the origin of the widget (in world coordinates). The origin is the point where the widget places itself. Note that rotations and scaling occurs around the origin.

- `obj.GetTransform (vtkTransform t)` - Retrieve a linear transform characterizing the affine transformation generated by this widget. This method copies its internal transform into the transform provided. Note that the PlaceWidget() method initializes the internal matrix to identity. All subsequent widget operations (i.e., scale, translate, rotate, shear) are concatenated with the internal transform.

- `obj.SetProperty (vtkProperty2D )` - Set/Get the properties when unselected and selected.

- `obj.SetSelectedProperty (vtkProperty2D )` - Set/Get the properties when unselected and selected.

- `obj.SetTextProperty (vtkTextProperty )` - Set/Get the properties when unselected and selected.

- `vtkProperty2D = obj.GetProperty ()` - Set/Get the properties when unselected and selected.

- `vtkProperty2D = obj.GetSelectedProperty ()` - Set/Get the properties when unselected and selected.

- `vtkTextProperty = obj.GetTextProperty ()` - Set/Get the properties when unselected and selected.

- `obj.SetDisplayText (int )` - Enable the display of text with numeric values characterizing the transformation. Rotation and shear are expressed in degrees; translation the distance in world coordinates; and scale normalized (sx,sy) values.

- `int = obj.GetDisplayText ()` - Enable the display of text with numeric values characterizing the transformation. Rotation and shear are expressed in degrees; translation the distance in world coordinates; and scale normalized (sx,sy) values.

- `obj.DisplayTextOn ()` - Enable the display of text with numeric values characterizing the transformation. Rotation and shear are expressed in degrees; translation the distance in world coordinates; and scale normalized (sx,sy) values.

- `obj.DisplayTextOff ()` - Enable the display of text with numeric values characterizing the transformation. Rotation and shear are expressed in degrees; translation the distance in world coordinates; and scale normalized (sx,sy) values.

- `obj.PlaceWidget (double bounds[6])` - Subclasses of vtkAffineRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other. Note: PlaceWidget() reinitializes the transformation matrix (i.e., sets it to identity). It also sets the origin for scaling and rotation.
- `obj.StartWidgetInteraction (double eventPos[2])` - Subclasses of vtkAffineRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other. Note: PlaceWidget() reinitializes the transformation matrix (i.e., sets it to identity). It also sets the origin for scaling and rotation.
- `obj.WidgetInteraction (double eventPos[2])` - Subclasses of vtkAffineRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other. Note: PlaceWidget() reinitializes the transformation matrix (i.e., sets it to identity). It also sets the origin for scaling and rotation.
- `obj.EndWidgetInteraction (double eventPos[2])` - Subclasses of vtkAffineRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other. Note: PlaceWidget() reinitializes the transformation matrix (i.e., sets it to identity). It also sets the origin for scaling and rotation.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Subclasses of vtkAffineRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other. Note: PlaceWidget() reinitializes the transformation matrix (i.e., sets it to identity). It also sets the origin for scaling and rotation.
- `obj.BuildRepresentation ()` - Subclasses of vtkAffineRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other. Note: PlaceWidget() reinitializes the transformation matrix (i.e., sets it to identity). It also sets the origin for scaling and rotation.
- `obj.ShallowCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp.
- `obj.GetActors2D (vtkPropCollection )` - Methods to make this class behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.

42.6 vtkAffineWidget

42.6.1 Usage

The vtkAffineWidget is used to perform affine transformations on objects. (Affine transformations are transformations that keep parallel lines parallel. Affine transformations include translation, scaling, rotation, and shearing.)

To use this widget, set the widget representation. The representation maintains a transformation matrix and other instance variables consistent with the transformations applied by this widget.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
LeftButtonPressEvent - select widget: depending on which part is selected
                       translation, rotation, scaling, or shearing may follow.
LeftButtonReleaseEvent - end selection of widget.
MouseMoveEvent - interactive movement across widget
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkAffineWidget's widget events:

```
vtkWidgetEvent::Select -- focal point is being selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Move -- a request for widget motion
```

In turn, when these widget events are processed, the vtkAffineWidget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkAffineWidget, simply invoke its constructor as follows

```
obj = vtkAffineWidget
```

42.6.2 Methods

The class vtkAffineWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAffineWidget class.

- `string = obj.GetClassName ()` - Standard VTK class macros.
- `int = obj.IsA (string name)` - Standard VTK class macros.
- `vtkAffineWidget = obj.NewInstance ()` - Standard VTK class macros.
- `vtkAffineWidget = obj.SafeDownCast (vtkObject o)` - Standard VTK class macros.
- `obj.SetRepresentation (vtkAffineRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `obj.SetEnabled (int )` - Methods for activiating this widget. This implementation extends the superclasses' in order to resize the widget handles due to a render start event.

42.7 vtkAngleRepresentation

42.7.1 Usage

The vtkAngleRepresentation is a superclass for classes representing the vtkAngleWidget. This representation consists of two rays and three vtkHandleRepresentations to place and manipulate the three points defining the angle representation. (Note: the three points are referred to as Point1, Center, and Point2, at the two end points (Point1 and Point2) and Center (around which the angle is measured).

To create an instance of class vtkAngleRepresentation, simply invoke its constructor as follows

```
obj = vtkAngleRepresentation
```

42.7.2 Methods

The class vtkAngleRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAngleRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkAngleRepresentation = obj.NewInstance ()` - Standard VTK methods.
- `vtkAngleRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `double = obj.GetAngle ()` - This representation and all subclasses must keep an angle (in degrees) consistent with the state of the widget.
- `obj.GetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the three points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetCenterWorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the three points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the three points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the three points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetCenterDisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the three points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the three points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the three points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetCenterDisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the three points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the three points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetHandleRepresentation (vtkHandleRepresentation handle)` - This method is used to specify the type of handle representation to use for the three internal vtkHandleWidgets within vtkAngleRepresentation. To use this method, create a dummy vtkHandleRepresentation (or subclass), and then invoke this method with this dummy. Then the vtkAngleRepresentation uses this dummy to clone three vtkHandleRepresentations of the same type. Make sure you set the handle representation before the widget is enabled. (The method InstantiateHandleRepresentation() is invoked by the vtkAngle widget.)

- `obj.InstantiateHandleRepresentation ()` - This method is used to specify the type of handle representation to use for the three internal vtkHandleWidgets within vtkAngleRepresentation. To use this method, create a dummy vtkHandleRepresentation (or subclass), and then invoke this method with this dummy. Then the vtkAngleRepresentation uses this dummy to clone three vtkHandleRepresentations of the same type. Make sure you set the handle representation before the widget is enabled. (The method InstantiateHandleRepresentation() is invoked by the vtkAngle widget.)

- `vtkHandleRepresentation = obj.GetPoint1Representation ()` - Set/Get the handle representations used for the vtkAngleRepresentation.

- `vtkHandleRepresentation = obj.GetCenterRepresentation ()` - Set/Get the handle representations used for the vtkAngleRepresentation.

- `vtkHandleRepresentation = obj.GetPoint2Representation ()` - Set/Get the handle representations used for the vtkAngleRepresentation.

- `obj.SetTolerance (int )` - The tolerance representing the distance to the representation (in pixels) in which the cursor is considered near enough to the end points of the representation to be active.

- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the representation (in pixels) in which the cursor is considered near enough to the end points of the representation to be active.

- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the representation (in pixels) in which the cursor is considered near enough to the end points of the representation to be active.

- `int = obj.GetTolerance ()` - The tolerance representing the distance to the representation (in pixels) in which the cursor is considered near enough to the end points of the representation to be active.

- `obj.SetLabelFormat (string )` - Specify the format to use for labelling the angle. Note that an empty string results in no label, or a format string without a "will not print the angle value.

- `string = obj.GetLabelFormat ()` - Specify the format to use for labelling the angle. Note that an empty string results in no label, or a format string without a "will not print the angle value.

- `obj.SetRay1Visibility (int )` - Special methods for turning off the rays and arc that define the cone and arc of the angle.

- `int = obj.GetRay1Visibility ()` - Special methods for turning off the rays and arc that define the cone and arc of the angle.

- `obj.Ray1VisibilityOn ()` - Special methods for turning off the rays and arc that define the cone and arc of the angle.

- `obj.Ray1VisibilityOff ()` - Special methods for turning off the rays and arc that define the cone and arc of the angle.

- `obj.SetRay2Visibility (int )` - Special methods for turning off the rays and arc that define the cone and arc of the angle.

- `int = obj.GetRay2Visibility ()` - Special methods for turning off the rays and arc that define the cone and arc of the angle.

- `obj.Ray2VisibilityOn ()` - Special methods for turning off the rays and arc that define the cone and arc of the angle.

- `obj.Ray2VisibilityOff ()` - Special methods for turning off the rays and arc that define the cone and arc of the angle.

- `obj.SetArcVisibility (int )` - Special methods for turning off the rays and arc that define the cone and arc of the angle.
- `int = obj.GetArcVisibility ()` - Special methods for turning off the rays and arc that define the cone and arc of the angle.
- `obj.ArcVisibilityOn ()` - Special methods for turning off the rays and arc that define the cone and arc of the angle.
- `obj.ArcVisibilityOff ()` - Special methods for turning off the rays and arc that define the cone and arc of the angle.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.StartWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.CenterWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.

42.8 vtkAngleRepresentation2D

42.8.1 Usage

The vtkAngleRepresentation2D is a representation for the vtkAngleWidget. This representation consists of two rays and three vtkHandleRepresentations to place and manipulate the three points defining the angle representation. (Note: the three points are referred to as Point1, Center, and Point2, at the two end points (Point1 and Point2) and Center (around which the angle is measured). This particular implementation is a 2D representation, meaning that it draws in the overlay plane.

To create an instance of class vtkAngleRepresentation2D, simply invoke its constructor as follows

```
obj = vtkAngleRepresentation2D
```

42.8.2 Methods

The class vtkAngleRepresentation2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAngleRepresentation2D class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkAngleRepresentation2D = obj.NewInstance ()` - Standard VTK methods.
- `vtkAngleRepresentation2D = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `double = obj.GetAngle ()` - Satisfy the superclasses API.
- `obj.GetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.GetCenterWorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetCenterDisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetCenterDisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `vtkLeaderActor2D = obj.GetRay1 ()` - Set/Get the three leaders used to create this representation. By obtaining these leaders the user can set the appropriate properties, etc.
- `vtkLeaderActor2D = obj.GetRay2 ()` - Set/Get the three leaders used to create this representation. By obtaining these leaders the user can set the appropriate properties, etc.
- `vtkLeaderActor2D = obj.GetArc ()` - Set/Get the three leaders used to create this representation. By obtaining these leaders the user can set the appropriate properties, etc.
- `obj.BuildRepresentation ()` - Method defined by vtkWidgetRepresentation superclass and needed here.
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Methods required by vtkProp superclass.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods required by vtkProp superclass.

42.9 vtkAngleRepresentation3D

42.9.1 Usage

The vtkAngleRepresentation3D is a representation for the vtkAngleWidget. This representation consists of two rays and three vtkHandleRepresentations to place and manipulate the three points defining the angle representation. (Note: the three points are referred to as Point1, Center, and Point2, at the two end points (Point1 and Point2) and Center (around which the angle is measured). This particular implementation is a 3D representation, meaning that it draws in the overlay plane.

To create an instance of class vtkAngleRepresentation3D, simply invoke its constructor as follows

```
obj = vtkAngleRepresentation3D
```

42.9.2 Methods

The class vtkAngleRepresentation3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAngleRepresentation3D class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkAngleRepresentation3D = obj.NewInstance ()` - Standard VTK methods.
- `vtkAngleRepresentation3D = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `double = obj.GetAngle ()` - Satisfy the superclasses API. Angle returned is in radians.
- `obj.GetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetCenterWorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetCenterWorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetCenterDisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetCenterDisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.GetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `vtkActor = obj.GetRay1 ()` - Set/Get the three leaders used to create this representation. By obtaining these leaders the user can set the appropriate properties, etc.
- `vtkActor = obj.GetRay2 ()` - Set/Get the three leaders used to create this representation. By obtaining these leaders the user can set the appropriate properties, etc.
- `vtkActor = obj.GetArc ()` - Set/Get the three leaders used to create this representation. By obtaining these leaders the user can set the appropriate properties, etc.
- `vtkFollower = obj.GetTextActor ()` - Set/Get the three leaders used to create this representation. By obtaining these leaders the user can set the appropriate properties, etc.
- `obj.SetTextActorScale (double scale[3])` - Scale text.
- `obj.BuildRepresentation ()` - Method defined by vtkWidgetRepresentation superclass and needed here.
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Methods required by vtkProp superclass.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Methods required by vtkProp superclass.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Methods required by vtkProp superclass.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods required by vtkProp superclass.

42.10 vtkAngleWidget

42.10.1 Usage

The vtkAngleWidget is used to measure the angle between two rays (defined by three points). The three points (two end points and a center) can be positioned independently, and when they are released, a special PlacePointEvent is invoked so that special operations may be take to reposition the point (snap to grid, etc.) The widget has two different modes of interaction: when initially defined (i.e., placing the three points) and then a manipulate mode (adjusting the position of the three points).

To use this widget, specify an instance of vtkAngleWidget and a representation (a subclass of vtkAngleRepresentation). The widget is implemented using three instances of vtkHandleWidget which are used to position the three points. The representations for these handle widgets are provided by the vtkAngleRepresentation.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
LeftButtonPressEvent - add a point or select a handle
MouseMoveEvent - position the second or third point, or move a handle
LeftButtonReleaseEvent - release the selected handle
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkAngleWidget's widget events:

```
vtkWidgetEvent::AddPoint -- add one point; depending on the state
                            it may the first, second or third point
                            added. Or, if near a handle, select the handle.
vtkWidgetEvent::Move -- position the second or third point, or move the
```

```
                          handle depending on the state.
vtkWidgetEvent::EndSelect -- the handle manipulation process has completed.
```

This widget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (beginning to interact)
vtkCommand::EndInteractionEvent (completing interaction)
vtkCommand::InteractionEvent (moving a handle)
vtkCommand::PlacePointEvent (after a point is positioned;
                             call data includes handle id (0,1,2))
```

To create an instance of class vtkAngleWidget, simply invoke its constructor as follows

```
obj = vtkAngleWidget
```

42.10.2 Methods

The class vtkAngleWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkAngleWidget class.

- `string = obj.GetClassName ()` - Standard methods for a VTK class.
- `int = obj.IsA (string name)` - Standard methods for a VTK class.
- `vtkAngleWidget = obj.NewInstance ()` - Standard methods for a VTK class.
- `vtkAngleWidget = obj.SafeDownCast (vtkObject o)` - Standard methods for a VTK class.
- `obj.SetEnabled (int )` - The method for activiating and deactiviating this widget. This method must be overridden because it is a composite widget and does more than its superclasses' vtkAbstractWidget::SetEnabled() method.
- `obj.SetRepresentation (vtkAngleRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `int = obj.IsAngleValid ()` - A flag indicates whether the angle is valid. The angle value only becomes valid after two of the three points are placed.
- `obj.SetProcessEvents (int )` - Methods to change the whether the widget responds to interaction. Overridden to pass the state to component widgets.

42.11 vtkBalloonRepresentation

42.11.1 Usage

The vtkBalloonRepresentation is used to represent the vtkBalloonWidget. This representation is defined by two items: a text string and an image. At least one of these two items must be defined, but it is allowable to specify both, or just an image or just text. If both the text and image are specified, then methods are available for positioning the text and image with respect to each other.

The balloon representation consists of three parts: text, a rectangular frame behind the text, and an image placed next to the frame and sized to match the frame.

The size of the balloon is ultimately controlled by the text properties (i.e., font size). This representation uses a layout policy as follows.

If there is just text and no image, then the text properties and padding are used to control the size of the balloon.

If there is just an image and no text, then the ImageSize[2] member is used to control the image size. (The image will fit into this rectangle, but will not necessarily fill the whole rectangle, i.e., the image is not stretched).

If there is text and an image, the following approach ia used. First, based on the font size and other related properties (e.g., padding), determine the size of the frame. Second, depending on the layout of the image and text frame, control the size of the neighboring image (since the frame and image share a common edge). However, if this results in an image that is smaller than ImageSize[2], then the image size will be set to ImageSize[2] and the frame will be adjusted accordingly. The text is always placed in the center of the frame if the frame is resized.

To create an instance of class vtkBalloonRepresentation, simply invoke its constructor as follows

```
obj = vtkBalloonRepresentation
```

42.11.2 Methods

The class vtkBalloonRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBalloonRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkBalloonRepresentation = obj.NewInstance ()` - Standard VTK methods.
- `vtkBalloonRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `obj.SetBalloonImage (vtkImageData img)` - Specify/retrieve the image to display in the balloon.
- `vtkImageData = obj.GetBalloonImage ()` - Specify/retrieve the image to display in the balloon.
- `string = obj.GetBalloonText ()` - Specify/retrieve the text to display in the balloon.
- `obj.SetBalloonText (string )` - Specify/retrieve the text to display in the balloon.
- `obj.SetImageSize (int , int )` - Specify the minimum size for the image. Note that this is a bounding rectangle, the image will fit inside of it. However, if the balloon consists of text plus an image, then the image may be bigger than ImageSize[2] to fit into the balloon frame.
- `obj.SetImageSize (int  a[2])` - Specify the minimum size for the image. Note that this is a bounding rectangle, the image will fit inside of it. However, if the balloon consists of text plus an image, then the image may be bigger than ImageSize[2] to fit into the balloon frame.
- `int = obj. GetImageSize ()` - Specify the minimum size for the image. Note that this is a bounding rectangle, the image will fit inside of it. However, if the balloon consists of text plus an image, then the image may be bigger than ImageSize[2] to fit into the balloon frame.
- `obj.SetTextProperty (vtkTextProperty p)` - Set/get the text property (relevant only if text is shown).
- `vtkTextProperty = obj.GetTextProperty ()` - Set/get the text property (relevant only if text is shown).
- `obj.SetFrameProperty (vtkProperty2D p)` - Set/get the frame property (relevant only if text is shown). The frame lies behind the text.

- `vtkProperty2D = obj.GetFrameProperty ()` - Set/get the frame property (relevant only if text is shown). The frame lies behind the text.

- `obj.SetImageProperty (vtkProperty2D p)` - Set/get the image property (relevant only if an image is shown).

- `vtkProperty2D = obj.GetImageProperty ()` - Set/get the image property (relevant only if an image is shown).

- `obj.SetBalloonLayout (int )` - Specify the layout of the image and text within the balloon. Note that there are reduncies in these methods, for example SetBalloonLayoutToImageLeft() results in the same effect as SetBalloonLayoutToTextRight(). If only text is specified, or only an image is specified, then it doesn't matter how the layout is specified.

- `int = obj.GetBalloonLayout ()` - Specify the layout of the image and text within the balloon. Note that there are reduncies in these methods, for example SetBalloonLayoutToImageLeft() results in the same effect as SetBalloonLayoutToTextRight(). If only text is specified, or only an image is specified, then it doesn't matter how the layout is specified.

- `obj.SetBalloonLayoutToImageLeft ()` - Specify the layout of the image and text within the balloon. Note that there are reduncies in these methods, for example SetBalloonLayoutToImageLeft() results in the same effect as SetBalloonLayoutToTextRight(). If only text is specified, or only an image is specified, then it doesn't matter how the layout is specified.

- `obj.SetBalloonLayoutToImageRight ()` - Specify the layout of the image and text within the balloon. Note that there are reduncies in these methods, for example SetBalloonLayoutToImageLeft() results in the same effect as SetBalloonLayoutToTextRight(). If only text is specified, or only an image is specified, then it doesn't matter how the layout is specified.

- `obj.SetBalloonLayoutToImageBottom ()` - Specify the layout of the image and text within the balloon. Note that there are reduncies in these methods, for example SetBalloonLayoutToImageLeft() results in the same effect as SetBalloonLayoutToTextRight(). If only text is specified, or only an image is specified, then it doesn't matter how the layout is specified.

- `obj.SetBalloonLayoutToImageTop ()` - Specify the layout of the image and text within the balloon. Note that there are reduncies in these methods, for example SetBalloonLayoutToImageLeft() results in the same effect as SetBalloonLayoutToTextRight(). If only text is specified, or only an image is specified, then it doesn't matter how the layout is specified.

- `obj.SetBalloonLayoutToTextLeft ()` - Specify the layout of the image and text within the balloon. Note that there are reduncies in these methods, for example SetBalloonLayoutToImageLeft() results in the same effect as SetBalloonLayoutToTextRight(). If only text is specified, or only an image is specified, then it doesn't matter how the layout is specified.

- `obj.SetBalloonLayoutToTextRight ()` - Specify the layout of the image and text within the balloon. Note that there are reduncies in these methods, for example SetBalloonLayoutToImageLeft() results in the same effect as SetBalloonLayoutToTextRight(). If only text is specified, or only an image is specified, then it doesn't matter how the layout is specified.

- `obj.SetBalloonLayoutToTextTop ()` - Specify the layout of the image and text within the balloon. Note that there are reduncies in these methods, for example SetBalloonLayoutToImageLeft() results in the same effect as SetBalloonLayoutToTextRight(). If only text is specified, or only an image is specified, then it doesn't matter how the layout is specified.

- `obj.SetBalloonLayoutToTextBottom ()` - Set/Get the offset from the mouse pointer from which to place the balloon. The representation will try and honor this offset unless there is a collision with the side of the renderer, in which case the balloon will be repositioned to lie within the rendering window.

- `obj.SetOffset (int , int )` - Set/Get the offset from the mouse pointer from which to place the balloon. The representation will try and honor this offset unless there is a collision with the side of the renderer, in which case the balloon will be repositioned to lie within the rendering window.
- `obj.SetOffset (int a[2])` - Set/Get the offset from the mouse pointer from which to place the balloon. The representation will try and honor this offset unless there is a collision with the side of the renderer, in which case the balloon will be repositioned to lie within the rendering window.
- `int = obj. GetOffset ()` - Set/Get the offset from the mouse pointer from which to place the balloon. The representation will try and honor this offset unless there is a collision with the side of the renderer, in which case the balloon will be repositioned to lie within the rendering window.
- `obj.SetPadding (int )` - Set/Get the padding (in pixels) that is used between the text and the frame.
- `int = obj.GetPaddingMinValue ()` - Set/Get the padding (in pixels) that is used between the text and the frame.
- `int = obj.GetPaddingMaxValue ()` - Set/Get the padding (in pixels) that is used between the text and the frame.
- `int = obj.GetPadding ()` - Set/Get the padding (in pixels) that is used between the text and the frame.
- `obj.StartWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.EndWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Methods required by vtkProp superclass.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods required by vtkProp superclass.

42.12 vtkBalloonWidget

42.12.1 Usage

The vtkBalloonWidget is used to popup text and/or an image when the mouse hovers over an instance of vtkProp. The widget keeps track of (vtkProp,vtkBalloon) pairs (where the internal vtkBalloon class is defined by a pair of vtkStdString and vtkImageData), and when the mouse stops moving for a user-specified period of time over the vtkProp, then the vtkBalloon is drawn nearby the vtkProp. Note that an instance of vtkBalloonRepresentation is used to draw the balloon.

To use this widget, specify an instance of vtkBalloonWidget and a representation (e.g., vtkBalloonRepresentation). Then list all instances of vtkProp, a text string, and/or an instance of vtkImageData to be associated with each vtkProp. (Note that you can specify both text and an image, or just one or the other.) You may also wish to specify the hover delay (i.e., set in the superclass vtkHoverWidget).

.SECTION Event Bindings By default, the widget observes the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
MouseMoveEvent - occurs when mouse is moved in render window.
TimerEvent - occurs when the time between events (e.g., mouse move)
             is greater than TimerDuration.
KeyPressEvent - when the ''Enter'' key is pressed after the balloon appears,
                a callback is activited (e.g., WidgetActivateEvent).
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkBalloonWidget's widget events:

```
vtkWidgetEvent::Move -- start the timer
vtkWidgetEvent::TimedOut -- when hovering occurs,
vtkWidgetEvent::SelectAction -- activate any callbacks associated
                                with the balloon.
```

This widget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::TimerEvent (when hovering is determined to occur)
vtkCommand::EndInteractionEvent (after a hover has occured and the
                                 mouse begins moving again).
vtkCommand::WidgetActivateEvent (when the balloon is selected with a
                                 keypress).
```

To create an instance of class vtkBalloonWidget, simply invoke its constructor as follows

```
obj = vtkBalloonWidget
```

42.12.2 Methods

The class vtkBalloonWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBalloonWidget class.

- `string = obj.GetClassName ()` - Standard methods for a VTK class.
- `int = obj.IsA (string name)` - Standard methods for a VTK class.
- `vtkBalloonWidget = obj.NewInstance ()` - Standard methods for a VTK class.
- `vtkBalloonWidget = obj.SafeDownCast (vtkObject o)` - Standard methods for a VTK class.
- `obj.SetEnabled (int )` - The method for activiating and deactiviating this widget. This method must be overridden because it performs special timer-related operations.
- `obj.SetRepresentation (vtkBalloonRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `obj.AddBalloon (vtkProp prop, string str, vtkImageData img)` - Add and remove text and/or an image to be associated with a vtkProp. You may add one or both of them.
- `obj.AddBalloon (vtkProp prop, string str)` - Add and remove text and/or an image to be associated with a vtkProp. You may add one or both of them.
- `obj.RemoveBalloon (vtkProp prop)` - Add and remove text and/or an image to be associated with a vtkProp. You may add one or both of them.
- `string = obj.GetBalloonString (vtkProp prop)` - Methods to retrieve the information associated with each vtkProp (i.e., the information that makes up each balloon). A NULL will be returned if the vtkProp does not exist, or if a string or image have not been associated with the specified vtkProp.
- `vtkImageData = obj.GetBalloonImage (vtkProp prop)` - Methods to retrieve the information associated with each vtkProp (i.e., the information that makes up each balloon). A NULL will be returned if the vtkProp does not exist, or if a string or image have not been associated with the specified vtkProp.

- `vtkProp = obj.GetCurrentProp ()` - Set/Get the object used to perform pick operations. Since the vtkBalloonWidget operates on vtkProps, the picker must be a subclass of vtkAbstractPropPicker. (Note: if not specified, an instance of vtkPropPicker is used.)

- `obj.SetPicker (vtkAbstractPropPicker )` - Set/Get the object used to perform pick operations. Since the vtkBalloonWidget operates on vtkProps, the picker must be a subclass of vtkAbstractPropPicker. (Note: if not specified, an instance of vtkPropPicker is used.)

- `vtkAbstractPropPicker = obj.GetPicker ()` - Set/Get the object used to perform pick operations. Since the vtkBalloonWidget operates on vtkProps, the picker must be a subclass of vtkAbstractPropPicker. (Note: if not specified, an instance of vtkPropPicker is used.)

42.13 vtkBezierContourLineInterpolator

42.13.1 Usage

The line interpolator interpolates supplied nodes (see InterpolateLine) with bezier line segments. The finess of the curve may be controlled using SetMaximumCurveError and SetMaximumNumberOfLineSegments.

To create an instance of class vtkBezierContourLineInterpolator, simply invoke its constructor as follows

```
obj = vtkBezierContourLineInterpolator
```

42.13.2 Methods

The class vtkBezierContourLineInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBezierContourLineInterpolator class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.

- `int = obj.IsA (string name)` - Standard methods for instances of this class.

- `vtkBezierContourLineInterpolator = obj.NewInstance ()` - Standard methods for instances of this class.

- `vtkBezierContourLineInterpolator = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.

- `int = obj.InterpolateLine (vtkRenderer ren, vtkContourRepresentation rep, int idx1, int idx2)`

- `obj.SetMaximumCurveError (double )` - The difference between a line segment connecting two points and the curve connecting the same points. In the limit of the length of the curve dx -¿ 0, the two values will be the same. The smaller this number, the finer the bezier curve will be interpolated. Default is 0.005

- `double = obj.GetMaximumCurveErrorMinValue ()` - The difference between a line segment connecting two points and the curve connecting the same points. In the limit of the length of the curve dx -¿ 0, the two values will be the same. The smaller this number, the finer the bezier curve will be interpolated. Default is 0.005

- `double = obj.GetMaximumCurveErrorMaxValue ()` - The difference between a line segment connecting two points and the curve connecting the same points. In the limit of the length of the curve dx -¿ 0, the two values will be the same. The smaller this number, the finer the bezier curve will be interpolated. Default is 0.005

- `double = obj.GetMaximumCurveError ()` - The difference between a line segment connecting two points and the curve connecting the same points. In the limit of the length of the curve dx -¿ 0, the two values will be the same. The smaller this number, the finer the bezier curve will be interpolated. Default is 0.005

- `obj.SetMaximumCurveLineSegments (int )` - Maximum number of bezier line segments between two nodes. Larger values create a finer interpolation. Default is 100.

- `int = obj.GetMaximumCurveLineSegmentsMinValue ()` - Maximum number of bezier line segments between two nodes. Larger values create a finer interpolation. Default is 100.

- `int = obj.GetMaximumCurveLineSegmentsMaxValue ()` - Maximum number of bezier line segments between two nodes. Larger values create a finer interpolation. Default is 100.

- `int = obj.GetMaximumCurveLineSegments ()` - Maximum number of bezier line segments between two nodes. Larger values create a finer interpolation. Default is 100.

- `obj.GetSpan (int nodeIndex, vtkIntArray nodeIndices, vtkContourRepresentation rep)` - Span of the interpolator. ie. the number of control points its supposed to interpolate given a node.

  The first argument is the current nodeIndex. ie, you'd be trying to interpolate between nodes "nodeIndex" and "nodeIndex-1", unless you're closing the contour in which case, you're trying to interpolate "nodeIndex" and "Node=0". The node span is returned in a vtkIntArray.

  The node span is returned in a vtkIntArray. The node span returned by this interpolator will be a 2-tuple with a span of 4.

42.14 vtkBiDimensionalRepresentation2D

42.14.1 Usage

The vtkBiDimensionalRepresentation2D is used to represent the bi-dimensional measure in a 2D (overlay) context. This representation consists of two perpendicular lines defined by four vtkHandleRepresentations. The four handles can be independently manipulated consistent with the orthogonal constraint on the lines. (Note: the four points are referred to as Point1, Point2, Point3 and Point4. Point1 and Point2 define the first line; and Point3 and Point4 define the second orthogonal line.)

To create this widget, you click to place the first two points. The third point is mirrored with the fourth point; when you place the third point (which is orthogonal to the lined defined by the first two points), the fourth point is dropped as well. After definition, the four points can be moved (in constrained fashion, preserving orthogonality). Further, the entire widget can be translated by grabbing the center point of the widget; each line can be moved along the other line; and the entire widget can be rotated around its center point.

To create an instance of class vtkBiDimensionalRepresentation2D, simply invoke its constructor as follows

```
obj = vtkBiDimensionalRepresentation2D
```

42.14.2 Methods

The class vtkBiDimensionalRepresentation2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBiDimensionalRepresentation2D class.

- `string = obj.GetClassName ()` - Standard VTK methods.

- `int = obj.IsA (string name)` - Standard VTK methods.

- `vtkBiDimensionalRepresentation2D = obj.NewInstance ()` - Standard VTK methods.

- `vtkBiDimensionalRepresentation2D = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `obj.SetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint3WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint4WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint3WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint4WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint3DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint4DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.GetPoint3DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint4DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the four points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetLine1Visibility (int )` - Special methods for turning off the lines that define the bi-dimensional measure. Generally these methods are used by the vtkBiDimensionalWidget to control the appearance of the widget. Note: turning off Line1 actually turns off Line1 and Line2.
- `int = obj.GetLine1Visibility ()` - Special methods for turning off the lines that define the bi-dimensional measure. Generally these methods are used by the vtkBiDimensionalWidget to control the appearance of the widget. Note: turning off Line1 actually turns off Line1 and Line2.
- `obj.Line1VisibilityOn ()` - Special methods for turning off the lines that define the bi-dimensional measure. Generally these methods are used by the vtkBiDimensionalWidget to control the appearance of the widget. Note: turning off Line1 actually turns off Line1 and Line2.
- `obj.Line1VisibilityOff ()` - Special methods for turning off the lines that define the bi-dimensional measure. Generally these methods are used by the vtkBiDimensionalWidget to control the appearance of the widget. Note: turning off Line1 actually turns off Line1 and Line2.
- `obj.SetLine2Visibility (int )` - Special methods for turning off the lines that define the bi-dimensional measure. Generally these methods are used by the vtkBiDimensionalWidget to control the appearance of the widget. Note: turning off Line1 actually turns off Line1 and Line2.
- `int = obj.GetLine2Visibility ()` - Special methods for turning off the lines that define the bi-dimensional measure. Generally these methods are used by the vtkBiDimensionalWidget to control the appearance of the widget. Note: turning off Line1 actually turns off Line1 and Line2.
- `obj.Line2VisibilityOn ()` - Special methods for turning off the lines that define the bi-dimensional measure. Generally these methods are used by the vtkBiDimensionalWidget to control the appearance of the widget. Note: turning off Line1 actually turns off Line1 and Line2.
- `obj.Line2VisibilityOff ()` - Special methods for turning off the lines that define the bi-dimensional measure. Generally these methods are used by the vtkBiDimensionalWidget to control the appearance of the widget. Note: turning off Line1 actually turns off Line1 and Line2.
- `obj.SetHandleRepresentation (vtkHandleRepresentation handle)` - This method is used to specify the type of handle representation to use for the four internal vtkHandleRepresentations within vtkBiDimensionalRepresentation2D. To use this method, create a dummy vtkHandleRepresentation (or subclass), and then invoke this method with this dummy. Then the vtkBiDimensionalRepresentation2D uses this dummy to clone four vtkHandleRepresentations of the same type. Make sure you set the handle representation before the widget is enabled. (The method InstantiateHandleRepresentation() is invoked by the vtkBiDimensionalWidget for the purposes of cloning.)
- `obj.InstantiateHandleRepresentation ()` - This method is used to specify the type of handle representation to use for the four internal vtkHandleRepresentations within vtkBiDimensionalRepresentation2D. To use this method, create a dummy vtkHandleRepresentation (or subclass), and then invoke this method with this dummy. Then the vtkBiDimensionalRepresentation2D uses this dummy to clone four vtkHandleRepresentations of the same type. Make sure you set the handle representation before the widget is enabled. (The method InstantiateHandleRepresentation() is invoked by the vtkBiDimensionalWidget for the purposes of cloning.)
- `vtkHandleRepresentation = obj.GetPoint1Representation ()` - Set/Get the handle representations used within the vtkBiDimensionalRepresentation2D. (Note: properties can be set by grabbing these representations and setting the properties appropriately.)

- `vtkHandleRepresentation = obj.GetPoint2Representation ()` - Set/Get the handle representations used within the vtkBiDimensionalRepresentation2D. (Note: properties can be set by grabbing these representations and setting the properties appropriately.)
- `vtkHandleRepresentation = obj.GetPoint3Representation ()` - Set/Get the handle representations used within the vtkBiDimensionalRepresentation2D. (Note: properties can be set by grabbing these representations and setting the properties appropriately.)
- `vtkHandleRepresentation = obj.GetPoint4Representation ()` - Set/Get the handle representations used within the vtkBiDimensionalRepresentation2D. (Note: properties can be set by grabbing these representations and setting the properties appropriately.)
- `vtkProperty2D = obj.GetLineProperty ()` - Retrieve the property used to control the appearance of the two orthogonal lines.
- `vtkProperty2D = obj.GetSelectedLineProperty ()` - Retrieve the property used to control the appearance of the two orthogonal lines.
- `vtkTextProperty = obj.GetTextProperty ()` - Retrieve the property used to control the appearance of the text labels.
- `obj.SetTolerance (int )` - The tolerance representing the distance to the representation (in pixels) in which the cursor is considered near enough to the representation to be active.
- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the representation (in pixels) in which the cursor is considered near enough to the representation to be active.
- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the representation (in pixels) in which the cursor is considered near enough to the representation to be active.
- `int = obj.GetTolerance ()` - The tolerance representing the distance to the representation (in pixels) in which the cursor is considered near enough to the representation to be active.
- `double = obj.GetLength1 ()` - Return the length of the line defined by (Point1,Point2). This is the distance in the world coordinate system.
- `double = obj.GetLength2 ()` - Return the length of the line defined by (Point3,Point4). This is the distance in the world coordinate system.
- `obj.SetLabelFormat (string )` - Specify the format to use for labelling the distance. Note that an empty string results in no label, or a format string without a "will not print the distance value.
- `string = obj.GetLabelFormat ()` - Specify the format to use for labelling the distance. Note that an empty string results in no label, or a format string without a "will not print the distance value.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.StartWidgetDefinition (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.Point2WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.Point3WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.StartWidgetManipulation (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.

- `obj.WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.Highlight (int highlightOn)` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Methods required by vtkProp superclass.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods required by vtkProp superclass.
- `obj.SetShowLabelAboveWidget (int )` - Toggle whether to display the label above or below the widget. Defaults to 1.
- `int = obj.GetShowLabelAboveWidget ()` - Toggle whether to display the label above or below the widget. Defaults to 1.
- `obj.ShowLabelAboveWidgetOn ()` - Toggle whether to display the label above or below the widget. Defaults to 1.
- `obj.ShowLabelAboveWidgetOff ()` - Toggle whether to display the label above or below the widget. Defaults to 1.
- `obj.SetID (long id)` - Set/get the id to display in the label.
- `long = obj.GetID ()` - Set/get the id to display in the label.
- `string = obj.GetLabelText ()` - Get the text shown in the widget's label.
- `obj.GetLabelPosition (double pos[3])` - Get the position of the widget's label in display coordinates.
- `obj.GetWorldLabelPosition (double pos[3])` - Get the position of the widget's label in display coordinates.

42.15 vtkBiDimensionalWidget

42.15.1 Usage

The vtkBiDimensionalWidget is used to measure the bi-dimensional length of an object. The bi-dimensional measure is defined by two finite, orthogonal lines that intersect within the finite extent of both lines. The lengths of these two lines gives the bi-dimensional measure. Each line is defined by two handle widgets at the end points of each line.

The orthognal constraint on the two lines limits how the four end points can be positioned. The first two points can be placed arbitrarily to define the first line (similar to vtkDistanceWidget). The placement of the third point is limited by the finite extent of the first line. As the third point is placed, the fourth point is placed on the opposite side of the first line. Once the third point is placed, the second line is defined since the fourth point is defined at the same time, but the fourth point can be moved along the second line (i.e., maintaining the orthogonal relationship between the two lines). Onced defined, any of the four points can be moved along their constraint lines. Also, each line can be translated along the other line (in an orthogonal direction), and the whole bi-dimensional widget can be rotated about its center point (see the description of the event bindings). Finally, selecting the point where the two orthogonal axes intersect, the entire widget can be translated in any direction.

Placement of any point results in a special PlacePointEvent invocation so that special operations may be performed to reposition the point. Motion of any point, moving the lines, or rotating the widget cause InteractionEvents to be invoked. Note that the widget has two fundamental modes: a define mode (when initially placing the points) and a manipulate mode (after the points are placed). Line translation and rotation are only possible in manipulate mode.

To use this widget, specify an instance of vtkBiDimensionalWidget and a representation (e.g., vtkBiDimensionalRepresentation2D). The widget is implemented using four instances of vtkHandleWidget which are used to position the end points of the two intersecting lines. The representations for these handle widgets are provided by the vtkBiDimensionalRepresentation2D class.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
LeftButtonPressEvent - define a point or manipulate a handle, line,
                       perform rotation or translate the widget.
MouseMoveEvent - position the points, move a line, rotate or translate the widget
LeftButtonReleaseEvent - release the selected handle and end interaction
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkBiDimensionalWidget's widget events:

```
vtkWidgetEvent::AddPoint -- (In Define mode:) Add one point; depending on the
                            state it may the first, second, third or fourth
                            point added. (In Manipulate mode:) If near a handle,
                            select the handle. Or if near a line, select the line.
vtkWidgetEvent::Move -- (In Define mode:) Position the second, third or fourth
                        point. (In Manipulate mode:) Move the handle, line or widget.
vtkWidgetEvent::EndSelect -- the manipulation process has completed.
```

This widget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (beginning to interact)
vtkCommand::EndInteractionEvent (completing interaction)
vtkCommand::InteractionEvent (moving a handle, line or performing rotation)
vtkCommand::PlacePointEvent (after a point is positioned;
                             call data includes handle id (0,1,2,4))
```

To create an instance of class vtkBiDimensionalWidget, simply invoke its constructor as follows

```
obj = vtkBiDimensionalWidget
```

42.15.2 Methods

The class vtkBiDimensionalWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBiDimensionalWidget class.

- `string = obj.GetClassName ()` - Standard methods for a VTK class.
- `int = obj.IsA (string name)` - Standard methods for a VTK class.
- `vtkBiDimensionalWidget = obj.NewInstance ()` - Standard methods for a VTK class.
- `vtkBiDimensionalWidget = obj.SafeDownCast (vtkObject o)` - Standard methods for a VTK class.
- `obj.SetEnabled (int )` - The method for activiating and deactiviating this widget. This method must be overridden because it is a composite widget and does more than its superclasses' vtkAbstractWidget::SetEnabled() method.
- `obj.SetRepresentation (vtkBiDimensionalRepresentation2D r)` - Create the default widget representation if one is not set.

- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `int = obj.IsMeasureValid ()` - A flag indicates whether the bi-dimensional measure is valid. The widget becomes valid after two of the four points are placed.
- `obj.SetProcessEvents (int )` - Methods to change the whether the widget responds to interaction. Overridden to pass the state to component widgets.

42.16 vtkBorderRepresentation

42.16.1 Usage

This class is used to represent and render a vtBorderWidget. To use this class, you need to specify the two corners of a rectangular region.

The class is typically subclassed so that specialized representations can be created. The class defines an API and a default implementation that the vtkBorderRepresentation interacts with to render itself in the scene.

To create an instance of class vtkBorderRepresentation, simply invoke its constructor as follows

```
obj = vtkBorderRepresentation
```

42.16.2 Methods

The class vtkBorderRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBorderRepresentation class.

- `string = obj.GetClassName ()` - Define standard methods.
- `int = obj.IsA (string name)` - Define standard methods.
- `vtkBorderRepresentation = obj.NewInstance ()` - Define standard methods.
- `vtkBorderRepresentation = obj.SafeDownCast (vtkObject o)` - Define standard methods.
- `vtkCoordinate = obj.GetPositionCoordinate ()` - Specify opposite corners of the box defining the boundary of the widget. By default, these coordinates are in the normalized viewport coordinate system, with Position the lower left of the outline, and Position2 relative to Position. Note that using these methods are affected by the ProportionalResize flag. That is, if the aspect ratio of the representation is to be preserved (e.g., ProportionalResize is on), then the rectangle (Position,Position2) is a bounding rectangle. Also,
- `obj.SetPosition (double, double)` - Specify opposite corners of the box defining the boundary of the widget. By default, these coordinates are in the normalized viewport coordinate system, with Position the lower left of the outline, and Position2 relative to Position. Note that using these methods are affected by the ProportionalResize flag. That is, if the aspect ratio of the representation is to be preserved (e.g., ProportionalResize is on), then the rectangle (Position,Position2) is a bounding rectangle. Also,
- `obj.SetPosition (double a[2])` - Specify opposite corners of the box defining the boundary of the widget. By default, these coordinates are in the normalized viewport coordinate system, with Position the lower left of the outline, and Position2 relative to Position. Note that using these methods are affected by the ProportionalResize flag. That is, if the aspect ratio of the representation is to be preserved (e.g., ProportionalResize is on), then the rectangle (Position,Position2) is a bounding rectangle. Also,

- `double = obj.GetPosition ()` - Specify opposite corners of the box defining the boundary of the widget. By default, these coordinates are in the normalized viewport coordinate system, with Position the lower left of the outline, and Position2 relative to Position. Note that using these methods are affected by the ProportionalResize flag. That is, if the aspect ratio of the representation is to be preserved (e.g., ProportionalResize is on), then the rectangle (Position,Position2) is a bounding rectangle. Also,

- `vtkCoordinate = obj.GetPosition2Coordinate ()` - Specify opposite corners of the box defining the boundary of the widget. By default, these coordinates are in the normalized viewport coordinate system, with Position the lower left of the outline, and Position2 relative to Position. Note that using these methods are affected by the ProportionalResize flag. That is, if the aspect ratio of the representation is to be preserved (e.g., ProportionalResize is on), then the rectangle (Position,Position2) is a bounding rectangle. Also,

- `obj.SetPosition2 (double, double)` - Specify opposite corners of the box defining the boundary of the widget. By default, these coordinates are in the normalized viewport coordinate system, with Position the lower left of the outline, and Position2 relative to Position. Note that using these methods are affected by the ProportionalResize flag. That is, if the aspect ratio of the representation is to be preserved (e.g., ProportionalResize is on), then the rectangle (Position,Position2) is a bounding rectangle. Also,

- `obj.SetPosition2 (double a[2])` - Specify opposite corners of the box defining the boundary of the widget. By default, these coordinates are in the normalized viewport coordinate system, with Position the lower left of the outline, and Position2 relative to Position. Note that using these methods are affected by the ProportionalResize flag. That is, if the aspect ratio of the representation is to be preserved (e.g., ProportionalResize is on), then the rectangle (Position,Position2) is a bounding rectangle. Also,

- `double = obj.GetPosition2 ()` - Specify opposite corners of the box defining the boundary of the widget. By default, these coordinates are in the normalized viewport coordinate system, with Position the lower left of the outline, and Position2 relative to Position. Note that using these methods are affected by the ProportionalResize flag. That is, if the aspect ratio of the representation is to be preserved (e.g., ProportionalResize is on), then the rectangle (Position,Position2) is a bounding rectangle. Also,

- `obj.SetShowBorder (int )` - Specify when and if the border should appear. If ShowBorder is "on", then the border will always appear. If ShowBorder is "off" then the border will never appear. If ShowBorder is "active" then the border will appear when the mouse pointer enters the region bounded by the border widget.

- `int = obj.GetShowBorderMinValue ()` - Specify when and if the border should appear. If ShowBorder is "on", then the border will always appear. If ShowBorder is "off" then the border will never appear. If ShowBorder is "active" then the border will appear when the mouse pointer enters the region bounded by the border widget.

- `int = obj.GetShowBorderMaxValue ()` - Specify when and if the border should appear. If ShowBorder is "on", then the border will always appear. If ShowBorder is "off" then the border will never appear. If ShowBorder is "active" then the border will appear when the mouse pointer enters the region bounded by the border widget.

- `int = obj.GetShowBorder ()` - Specify when and if the border should appear. If ShowBorder is "on", then the border will always appear. If ShowBorder is "off" then the border will never appear. If ShowBorder is "active" then the border will appear when the mouse pointer enters the region bounded by the border widget.

- `obj.SetShowBorderToOff ()` - Specify when and if the border should appear. If ShowBorder is "on", then the border will always appear. If ShowBorder is "off" then the border will never appear. If

ShowBorder is "active" then the border will appear when the mouse pointer enters the region bounded by the border widget.

- `obj.SetShowBorderToOn ()` - Specify when and if the border should appear. If ShowBorder is "on", then the border will always appear. If ShowBorder is "off" then the border will never appear. If ShowBorder is "active" then the border will appear when the mouse pointer enters the region bounded by the border widget.
- `obj.SetShowBorderToActive ()` - Specify the properties of the border.
- `vtkProperty2D = obj.GetBorderProperty ()` - Specify the properties of the border.
- `obj.SetProportionalResize (int )` - Indicate whether resizing operations should keep the x-y directions proportional to one another. Also, if ProportionalResize is on, then the rectangle (Position,Position2) is a bounding rectangle, and the representation will be placed in the rectangle in such a way as to preserve the aspect ratio of the representation.
- `int = obj.GetProportionalResize ()` - Indicate whether resizing operations should keep the x-y directions proportional to one another. Also, if ProportionalResize is on, then the rectangle (Position,Position2) is a bounding rectangle, and the representation will be placed in the rectangle in such a way as to preserve the aspect ratio of the representation.
- `obj.ProportionalResizeOn ()` - Indicate whether resizing operations should keep the x-y directions proportional to one another. Also, if ProportionalResize is on, then the rectangle (Position,Position2) is a bounding rectangle, and the representation will be placed in the rectangle in such a way as to preserve the aspect ratio of the representation.
- `obj.ProportionalResizeOff ()` - Indicate whether resizing operations should keep the x-y directions proportional to one another. Also, if ProportionalResize is on, then the rectangle (Position,Position2) is a bounding rectangle, and the representation will be placed in the rectangle in such a way as to preserve the aspect ratio of the representation.
- `obj.SetMinimumSize (int , int )` - Specify a minimum and/or maximum size (in pixels) that this representation can take. These methods require two values: size values in the x and y directions, respectively.
- `obj.SetMinimumSize (int a[2])` - Specify a minimum and/or maximum size (in pixels) that this representation can take. These methods require two values: size values in the x and y directions, respectively.
- `int = obj. GetMinimumSize ()` - Specify a minimum and/or maximum size (in pixels) that this representation can take. These methods require two values: size values in the x and y directions, respectively.
- `obj.SetMaximumSize (int , int )` - Specify a minimum and/or maximum size (in pixels) that this representation can take. These methods require two values: size values in the x and y directions, respectively.
- `obj.SetMaximumSize (int a[2])` - Specify a minimum and/or maximum size (in pixels) that this representation can take. These methods require two values: size values in the x and y directions, respectively.
- `int = obj. GetMaximumSize ()` - Specify a minimum and/or maximum size (in pixels) that this representation can take. These methods require two values: size values in the x and y directions, respectively.
- `obj.SetTolerance (int )` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).

- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).
- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).
- `int = obj.GetTolerance ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).
- `double = obj. GetSelectionPoint ()` - After a selection event within the region interior to the border; the normalized selection coordinates may be obtained.
- `obj.SetMoving (int )` - This is a modifier of the interaction state. When set, widget interaction allows the border (and stuff inside of it) to be translated with mouse motion.
- `int = obj.GetMoving ()` - This is a modifier of the interaction state. When set, widget interaction allows the border (and stuff inside of it) to be translated with mouse motion.
- `obj.MovingOn ()` - This is a modifier of the interaction state. When set, widget interaction allows the border (and stuff inside of it) to be translated with mouse motion.
- `obj.MovingOff ()` - This is a modifier of the interaction state. When set, widget interaction allows the border (and stuff inside of it) to be translated with mouse motion.
- `obj.BuildRepresentation ()` - Subclasses should implement these methods. See the superclasses' documentation for more information.
- `obj.StartWidgetInteraction (double eventPos[2])` - Subclasses should implement these methods. See the superclasses' documentation for more information.
- `obj.WidgetInteraction (double eventPos[2])` - Subclasses should implement these methods. See the superclasses' documentation for more information.
- `obj.GetSize (double size[2])` - Subclasses should implement these methods. See the superclasses' documentation for more information.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Subclasses should implement these methods. See the superclasses' documentation for more information.
- `obj.GetActors2D (vtkPropCollection )` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - These methods are necessary to make this representation behave as a vtkProp.

42.17 vtkBorderWidget

42.17.1 Usage

This class is a superclass for 2D widgets that may require a rectangular border. Besides drawing a border, the widget provides methods for resizing and moving the rectangular region (and associated border). The widget provides methods and internal data members so that subclasses can take advantage of this widgets capabilities, requiring only that the subclass defines a "representation", i.e., some combination of props or actors that can be managed in the 2D rectangular region.

The class defines basic positioning functionality, including the ability to size the widget with locked x/y proportions. The area within the border may be made "selectable" as well, meaning that a selection event interior to the widget invokes a virtual SelectRegion() method, which can be used to pick objects or otherwise manipulate data interior to the widget.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
On the boundary of the widget:
  LeftButtonPressEvent - select boundary
  LeftButtonReleaseEvent - deselect boundary
  MouseMoveEvent - move/resize widget depending on which portion of the
                   boundary was selected.
On the interior of the widget:
  LeftButtonPressEvent - invoke SelectButton() callback (if the ivar
                         Selectable is on)
Anywhere on the widget:
  MiddleButtonPressEvent - move the widget
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkBorderWidget's widget events:

```
vtkWidgetEvent::Select -- some part of the widget has been selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Translate -- the widget is to be translated
vtkWidgetEvent::Move -- a request for slider motion has been invoked
```

In turn, when these widget events are processed, this widget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkBorderWidget, simply invoke its constructor as follows

```
obj = vtkBorderWidget
```

42.17.2 Methods

The class vtkBorderWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBorderWidget class.

- `string = obj.GetClassName ()`

- `int = obj.IsA (string name)`
- `vtkBorderWidget = obj.NewInstance ()`
- `vtkBorderWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetSelectable (int )` - Indicate whether the interior region of the widget can be selected or not. If not, then events (such as left mouse down) allow the user to "move" the widget, and no selection is possible. Otherwise the SelectRegion() method is invoked.
- `int = obj.GetSelectable ()` - Indicate whether the interior region of the widget can be selected or not. If not, then events (such as left mouse down) allow the user to "move" the widget, and no selection is possible. Otherwise the SelectRegion() method is invoked.
- `obj.SelectableOn ()` - Indicate whether the interior region of the widget can be selected or not. If not, then events (such as left mouse down) allow the user to "move" the widget, and no selection is possible. Otherwise the SelectRegion() method is invoked.
- `obj.SelectableOff ()` - Indicate whether the interior region of the widget can be selected or not. If not, then events (such as left mouse down) allow the user to "move" the widget, and no selection is possible. Otherwise the SelectRegion() method is invoked.
- `obj.SetResizable (int )` - Indicate whether the boundary of the widget can be resized. If not, the cursor will not change to "resize" type when mouse over the boundary.
- `int = obj.GetResizable ()` - Indicate whether the boundary of the widget can be resized. If not, the cursor will not change to "resize" type when mouse over the boundary.
- `obj.ResizableOn ()` - Indicate whether the boundary of the widget can be resized. If not, the cursor will not change to "resize" type when mouse over the boundary.
- `obj.ResizableOff ()` - Indicate whether the boundary of the widget can be resized. If not, the cursor will not change to "resize" type when mouse over the boundary.
- `obj.SetRepresentation (vtkBorderRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.18 vtkBoundedPlanePointPlacer

42.18.1 Usage

vtkBoundedPlanePointPlacer is a type of point placer that constrains its points to a finite (i.e., bounded) plance.

To create an instance of class vtkBoundedPlanePointPlacer, simply invoke its constructor as follows

```
obj = vtkBoundedPlanePointPlacer
```

42.18.2 Methods

The class vtkBoundedPlanePointPlacer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBoundedPlanePointPlacer class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.

- `vtkBoundedPlanePointPlacer = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkBoundedPlanePointPlacer = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetProjectionNormal (int )` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `int = obj.GetProjectionNormalMinValue ()` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `int = obj.GetProjectionNormalMaxValue ()` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `int = obj.GetProjectionNormal ()` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `obj.SetProjectionNormalToXAxis ()` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `obj.SetProjectionNormalToYAxis ()` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `obj.SetProjectionNormalToZAxis ()` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `obj.SetProjectionNormalToOblique ()` - If the ProjectionNormal is set to Oblique, then this is the oblique plane used to constrain the handle position.
- `obj.SetObliquePlane (vtkPlane )` - If the ProjectionNormal is set to Oblique, then this is the oblique plane used to constrain the handle position.
- `obj.SetProjectionPosition (double position)` - The position of the bounding plane from the origin along the normal. The origin and normal are defined in the oblique plane when the ProjectionNormal is oblique. For the X, Y, and Z axes projection normals, the normal is the axis direction, and the origin is (0,0,0).
- `double = obj.GetProjectionPosition ()` - The position of the bounding plane from the origin along the normal. The origin and normal are defined in the oblique plane when the ProjectionNormal is oblique. For the X, Y, and Z axes projection normals, the normal is the axis direction, and the origin is (0,0,0).
- `obj.AddBoundingPlane (vtkPlane plane)` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.
- `obj.RemoveBoundingPlane (vtkPlane plane)` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.
- `obj.RemoveAllBoundingPlanes ()` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `obj.SetBoundingPlanes (vtkPlaneCollection )` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `vtkPlaneCollection = obj.GetBoundingPlanes ()` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `obj.SetBoundingPlanes (vtkPlanes planes)` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double worldPos[3], double w` - Given a renderer and a display position, compute the world position and world orientation for this point. A plane is defined by a combination of the ProjectionNormal, ProjectionOrigin, and Oblique-Plane ivars. The display position is projected onto this plane to determine a world position, and the orientation is set to the normal of the plane. If the point cannot project onto the plane or if it falls outside the bounds imposed by the BoundingPlanes, then 0 is returned, otherwise 1 is returned to indicate a valid return position and orientation.

- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double refWorldPos[3], doubl` - Given a renderer, a display position, and a reference world position, compute the new world position and orientation of this point. This method is typically used by the representation to move the point.

- `int = obj.ValidateWorldPosition (double worldPos[3])` - Give a world position check if it is valid - does it lie on the plane and within the bounds? Returns 1 if it is valid, 0 otherwise.

- `int = obj.ValidateWorldPosition (double worldPos[3], double worldOrient[9])`

- `int = obj.UpdateWorldPosition (vtkRenderer ren, double worldPos[3], double worldOrient[9])` - If the constraints on this placer are changed, then this method will be called by the representation on each of its points. For this placer, the world position will be converted to a display position, then ComputeWorldPosition will be used to update the point.

42.19 vtkBoxRepresentation

42.19.1 Usage

This class is a concrete representation for the vtkBoxWidget2. It represents a box with seven handles: one on each of the six faces, plus a center handle. Through interaction with the widget, the box representation can be arbitrarily positioned in the 3D space.

To use this representation, you normally use the PlaceWidget() method to position the widget at a specified region in space.

To create an instance of class vtkBoxRepresentation, simply invoke its constructor as follows

```
obj = vtkBoxRepresentation
```

42.19.2 Methods

The class vtkBoxRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBoxRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkBoxRepresentation = obj.NewInstance ()` - Standard methods for the class.
- `vtkBoxRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.
- `obj.GetPlanes (vtkPlanes planes)` - Get the planes describing the implicit function defined by the box widget. The user must provide the instance of the class vtkPlanes. Note that vtkPlanes is a subclass of vtkImplicitFunction, meaning that it can be used by a variety of filters to perform clipping, cutting, and selection of data. (The direction of the normals of the planes can be reversed enabling the InsideOut flag.)
- `obj.SetInsideOut (int )` - Set/Get the InsideOut flag. This data memeber is used in conjunction with the GetPlanes() method. When off, the normals point out of the box. When on, the normals point into the hexahedron. InsideOut is off by default.
- `int = obj.GetInsideOut ()` - Set/Get the InsideOut flag. This data memeber is used in conjunction with the GetPlanes() method. When off, the normals point out of the box. When on, the normals point into the hexahedron. InsideOut is off by default.
- `obj.InsideOutOn ()` - Set/Get the InsideOut flag. This data memeber is used in conjunction with the GetPlanes() method. When off, the normals point out of the box. When on, the normals point into the hexahedron. InsideOut is off by default.
- `obj.InsideOutOff ()` - Set/Get the InsideOut flag. This data memeber is used in conjunction with the GetPlanes() method. When off, the normals point out of the box. When on, the normals point into the hexahedron. InsideOut is off by default.
- `obj.GetTransform (vtkTransform t)` - Retrieve a linear transform characterizing the transformation of the box. Note that the transformation is relative to where PlaceWidget() was initially called. This method modifies the transform provided. The transform can be used to control the position of vtkProp3D's, as well as other transformation operations (e.g., vtkTranformPolyData).
- `obj.SetTransform (vtkTransform t)` - Set the position, scale and orientation of the box widget using the transform specified. Note that the transformation is relative to where PlaceWidget() was initially called (i.e., the original bounding box).
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that define the box widget. The polydata consists of 6 quadrilateral faces and 15 points. The first eight points define the eight corner vertices; the next six define the -x,+x, -y,+y, -z,+z face points; and the final point (the 15th out of 15 points) defines the center of the box. These point values are guaranteed to be up-to-date when either the widget's corresponding InteractionEvent or EndInteractionEvent events are invoked. The user provides the vtkPolyData and the points and cells are added to it.
- `vtkProperty = obj.GetHandleProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles, when selected or normal, can be specified.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles, when selected or normal, can be specified.

- `vtkProperty = obj.GetFaceProperty ()` - Get the face properties (the faces of the box). The properties of the face when selected and normal can be set.
- `vtkProperty = obj.GetSelectedFaceProperty ()` - Get the face properties (the faces of the box). The properties of the face when selected and normal can be set.
- `vtkProperty = obj.GetOutlineProperty ()` - Get the outline properties (the outline of the box). The properties of the outline when selected and normal can be set.
- `vtkProperty = obj.GetSelectedOutlineProperty ()` - Get the outline properties (the outline of the box). The properties of the outline when selected and normal can be set.
- `obj.SetOutlineFaceWires (int )` - Control the representation of the outline. This flag enables face wires. By default face wires are off.
- `int = obj.GetOutlineFaceWires ()` - Control the representation of the outline. This flag enables face wires. By default face wires are off.
- `obj.OutlineFaceWiresOn ()` - Control the representation of the outline. This flag enables face wires. By default face wires are off.
- `obj.OutlineFaceWiresOff ()` - Control the representation of the outline. This flag enables the cursor lines running between the handles. By default cursor wires are on.
- `obj.SetOutlineCursorWires (int )` - Control the representation of the outline. This flag enables the cursor lines running between the handles. By default cursor wires are on.
- `int = obj.GetOutlineCursorWires ()` - Control the representation of the outline. This flag enables the cursor lines running between the handles. By default cursor wires are on.
- `obj.OutlineCursorWiresOn ()` - Control the representation of the outline. This flag enables the cursor lines running between the handles. By default cursor wires are on.
- `obj.OutlineCursorWiresOff ()` - Switches handles (the spheres) on or off by manipulating the underlying actor visibility.
- `obj.HandlesOn ()` - Switches handles (the spheres) on or off by manipulating the underlying actor visibility.
- `obj.HandlesOff ()` - Switches handles (the spheres) on or off by manipulating the underlying actor visibility.
- `obj.PlaceWidget (double bounds[6])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.StartWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `double = obj.GetBounds ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods supporting, and required by, the rendering process.

- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Methods supporting, and required by, the rendering process.

- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Methods supporting, and required by, the rendering process.

- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods supporting, and required by, the rendering process.

- `obj.SetInteractionState (int state)` - The interaction state may be set from a widget (e.g., vtkBoxWidget2) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking process with the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.

42.20 vtkBoxWidget

42.20.1 Usage

This 3D widget defines a region of interest that is represented by an arbitrarily oriented hexahedron with interior face angles of 90 degrees (orthogonal faces). The object creates 7 handles that can be moused on and manipulated. The first six correspond to the six faces, the seventh is in the center of the hexahedron. In addition, a bounding box outline is shown, the "faces" of which can be selected for object rotation or scaling. A nice feature of the object is that the vtkBoxWidget, like any 3D widget, will work with the current interactor style. That is, if vtkBoxWidget does not handle an event, then all other registered observers (including the interactor style) have an opportunity to process the event. Otherwise, the vtkBoxWidget will terminate the processing of the event that it handles.

To use this object, just invoke SetInteractor() with the argument of the method a vtkRenderWindowInteractor. You may also wish to invoke "PlaceWidget()" to initially position the widget. The interactor will act normally until the "i" key (for "interactor") is pressed, at which point the vtkBoxWidget will appear. (See superclass documentation for information about changing this behavior.) By grabbing the six face handles (use the left mouse button), faces can be moved. By grabbing the center handle (with the left mouse button), the entire hexahedron can be translated. (Translation can also be employed by using the "shift-left-mouse-button" combination inside of the widget.) Scaling is achieved by using the right mouse button "up" the render window (makes the widget bigger) or "down" the render window (makes the widget smaller). To rotate vtkBoxWidget, pick a face (but not a face handle) and move the left mouse. (Note: the mouse button must be held down during manipulation.) Events that occur outside of the widget (i.e., no part of the widget is picked) are propagated to any other registered obsevers (such as the interaction style). Turn off the widget by pressing the "i" key again. (See the superclass documentation on key press activiation.)

The vtkBoxWidget is very flexible. It can be used to select, cut, clip, or perform any other operation that depends on an implicit function (use the GetPlanes() method); or it can be used to transform objects using a linear transformation (use the GetTransform() method). Typical usage of the widget is to make use of the StartInteractionEvent, InteractionEvent, and EndInteractionEvent events. The InteractionEvent is called on mouse motion; the other two events are called on button down and button up (either left or right button).

Some additional features of this class include the ability to control the rendered properties of the widget. You can set the properties of the selected and unselected representations of the parts of the widget. For example, you can set the property for the handles, faces, and outline in their normal and selected states.

To create an instance of class vtkBoxWidget, simply invoke its constructor as follows

```
obj = vtkBoxWidget
```

42.20.2 Methods

The class vtkBoxWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBoxWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkBoxWidget = obj.NewInstance ()`
- `vtkBoxWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double bounds[6])` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget ()` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Get the planes describing the implicit function defined by the box widget. The user must provide the instance of the class vtkPlanes. Note that vtkPlanes is a subclass of vtkImplicitFunction, meaning that it can be used by a variety of filters to perform clipping, cutting, and selection of data. (The direction of the normals of the planes can be reversed enabling the InsideOut flag.)
- `obj.GetPlanes (vtkPlanes planes)` - Get the planes describing the implicit function defined by the box widget. The user must provide the instance of the class vtkPlanes. Note that vtkPlanes is a subclass of vtkImplicitFunction, meaning that it can be used by a variety of filters to perform clipping, cutting, and selection of data. (The direction of the normals of the planes can be reversed enabling the InsideOut flag.)
- `obj.SetInsideOut (int )` - Set/Get the InsideOut flag. When off, the normals point out of the box. When on, the normals point into the hexahedron. InsideOut is off by default.
- `int = obj.GetInsideOut ()` - Set/Get the InsideOut flag. When off, the normals point out of the box. When on, the normals point into the hexahedron. InsideOut is off by default.
- `obj.InsideOutOn ()` - Set/Get the InsideOut flag. When off, the normals point out of the box. When on, the normals point into the hexahedron. InsideOut is off by default.
- `obj.InsideOutOff ()` - Set/Get the InsideOut flag. When off, the normals point out of the box. When on, the normals point into the hexahedron. InsideOut is off by default.
- `obj.GetTransform (vtkTransform t)` - Retrieve a linear transform characterizing the transformation of the box. Note that the transformation is relative to where PlaceWidget was initially called. This method modifies the transform provided. The transform can be used to control the position of vtkProp3D's, as well as other transformation operations (e.g., vtkTranformPolyData).
- `obj.SetTransform (vtkTransform t)` - Set the position, scale and orientation of the box widget using the transform specified. Note that the transformation is relative to where PlaceWidget was initially called (i.e., the original bounding box).
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that define the box widget. The polydata consists of 6 quadrilateral faces and 15 points. The first eight points define the eight corner vertices; the next six define the -x,+x, -y,+y, -z,+z face points; and the final point (the 15th out of 15 points) defines the center of the hexahedron. These point values are guaranteed to be up-to-date when either the InteractionEvent or EndInteractionEvent events are invoked. The user provides the vtkPolyData and the points and cells are added to it.

- `vtkProperty = obj.GetHandleProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles when selected and normal can be set.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles when selected and normal can be set.
- `obj.HandlesOn ()` - Switches handles (the spheres) on or off by manipulating the actor visibility.
- `obj.HandlesOff ()` - Switches handles (the spheres) on or off by manipulating the actor visibility.
- `vtkProperty = obj.GetFaceProperty ()` - Get the face properties (the faces of the box). The properties of the face when selected and normal can be set.
- `vtkProperty = obj.GetSelectedFaceProperty ()` - Get the face properties (the faces of the box). The properties of the face when selected and normal can be set.
- `vtkProperty = obj.GetOutlineProperty ()` - Get the outline properties (the outline of the box). The properties of the outline when selected and normal can be set.
- `vtkProperty = obj.GetSelectedOutlineProperty ()` - Get the outline properties (the outline of the box). The properties of the outline when selected and normal can be set.
- `obj.SetOutlineFaceWires (int )` - Control the representation of the outline. This flag enables face wires. By default face wires are off.
- `int = obj.GetOutlineFaceWires ()` - Control the representation of the outline. This flag enables face wires. By default face wires are off.
- `obj.OutlineFaceWiresOn ()` - Control the representation of the outline. This flag enables face wires. By default face wires are off.
- `obj.OutlineFaceWiresOff ()` - Control the representation of the outline. This flag enables the cursor lines running between the handles. By default cursor wires are on.
- `obj.SetOutlineCursorWires (int )` - Control the representation of the outline. This flag enables the cursor lines running between the handles. By default cursor wires are on.
- `int = obj.GetOutlineCursorWires ()` - Control the representation of the outline. This flag enables the cursor lines running between the handles. By default cursor wires are on.
- `obj.OutlineCursorWiresOn ()` - Control the representation of the outline. This flag enables the cursor lines running between the handles. By default cursor wires are on.
- `obj.OutlineCursorWiresOff ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `obj.SetTranslationEnabled (int )` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `int = obj.GetTranslationEnabled ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `obj.TranslationEnabledOn ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `obj.TranslationEnabledOff ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `obj.SetScalingEnabled (int )` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.

- `int = obj.GetScalingEnabled ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `obj.ScalingEnabledOn ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `obj.ScalingEnabledOff ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `obj.SetRotationEnabled (int )` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `int = obj.GetRotationEnabled ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `obj.RotationEnabledOn ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.
- `obj.RotationEnabledOff ()` - Control the behavior of the widget. Translation, rotation, and scaling can all be enabled and disabled.

42.21 vtkBoxWidget2

42.21.1 Usage

This 3D widget interacts with a vtkBoxRepresentation class (i.e., it handles the events that drive its corresponding representation). The representation is assumed to represent a region of interest that is represented by an arbitrarily oriented hexahedron (or box) with interior face angles of 90 degrees (i.e., orthogonal faces). The representation manifests seven handles that can be moused on and manipulated, plus the six faces can also be interacted with. The first six handles are placed on the six faces, the seventh is in the center of the box. In addition, a bounding box outline is shown, the "faces" of which can be selected for object rotation or scaling. A nice feature of vtkBoxWidget2, like any 3D widget, will work with the current interactor style. That is, if vtkBoxWidget2 does not handle an event, then all other registered observers (including the interactor style) have an opportunity to process the event. Otherwise, the vtkBoxWidget will terminate the processing of the event that it handles.

To use this widget, you generally pair it with a vtkBoxRepresentation (or a subclass). Variuos options are available in the representation for controlling how the widget appears, and how the widget functions.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
If one of the seven handles are selected:
  LeftButtonPressEvent - select the appropriate handle
  LeftButtonReleaseEvent - release the currently selected handle
  MouseMoveEvent - move the handle
If one of the faces is selected:
  LeftButtonPressEvent - select a box face
  LeftButtonReleaseEvent - release the box face
  MouseMoveEvent - rotate the box
In all the cases, independent of what is picked, the widget responds to the
following VTK events:
  MiddleButtonPressEvent - translate the widget
  MiddleButtonReleaseEvent - release the widget
  RightButtonPressEvent - scale the widget's representation
  RightButtonReleaseEvent - stop scaling the widget
  MouseMoveEvent - scale (if right button) or move (if middle button) the widget
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkBoxWidget2's widget events:

```
vtkWidgetEvent::Select -- some part of the widget has been selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Scale -- some part of the widget has been selected
vtkWidgetEvent::EndScale -- the selection process has completed
vtkWidgetEvent::Translate -- some part of the widget has been selected
vtkWidgetEvent::EndTranslate -- the selection process has completed
vtkWidgetEvent::Move -- a request for motion has been invoked
```

In turn, when these widget events are processed, the vtkBoxWidget2 invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkBoxWidget2, simply invoke its constructor as follows

```
obj = vtkBoxWidget2
```

42.21.2 Methods

The class vtkBoxWidget2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkBoxWidget2 class.

- `string = obj.GetClassName ()` - Standard class methods for type information and printing.
- `int = obj.IsA (string name)` - Standard class methods for type information and printing.
- `vtkBoxWidget2 = obj.NewInstance ()` - Standard class methods for type information and printing.
- `vtkBoxWidget2 = obj.SafeDownCast (vtkObject o)` - Standard class methods for type information and printing.
- `obj.SetRepresentation (vtkBoxRepresentation r)` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `obj.SetTranslationEnabled (int )` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `int = obj.GetTranslationEnabled ()` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `obj.TranslationEnabledOn ()` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `obj.TranslationEnabledOff ()` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `obj.SetScalingEnabled (int )` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `int = obj.GetScalingEnabled ()` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.

- `obj.ScalingEnabledOn ()` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `obj.ScalingEnabledOff ()` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `obj.SetRotationEnabled (int )` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `int = obj.GetRotationEnabled ()` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `obj.RotationEnabledOn ()` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `obj.RotationEnabledOff ()` - Control the behavior of the widget (i.e., how it processes events). Translation, rotation, and scaling can all be enabled and disabled.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set. By default, this is an instance of the vtkBoxRepresentation class.

42.22 vtkCameraRepresentation

42.22.1 Usage

This class provides support for interactively saving a series of camera views into an interpolated path (using vtkCameraInterpolator). The class typically works in conjunction with vtkCameraWidget. To use this class simply specify the camera to interpolate and use the methods AddCameraToPath(), AnimatePath(), and InitializePath() to add a new camera view, animate the current views, and initialize the interpolation.

To create an instance of class vtkCameraRepresentation, simply invoke its constructor as follows

```
obj = vtkCameraRepresentation
```

42.22.2 Methods

The class vtkCameraRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCameraRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK class methods.
- `int = obj.IsA (string name)` - Standard VTK class methods.
- `vtkCameraRepresentation = obj.NewInstance ()` - Standard VTK class methods.
- `vtkCameraRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK class methods.
- `obj.SetCamera (vtkCamera camera)` - Specify the camera to interpolate. This must be specified by the user.
- `vtkCamera = obj.GetCamera ()` - Specify the camera to interpolate. This must be specified by the user.
- `obj.SetInterpolator (vtkCameraInterpolator camInt)` - Get the vtkCameraInterpolator used to interpolate and save the sequence of camera views. If not defined, one is created automatically. Note that you can access this object to set the interpolation type (linear, spline) and other instance variables.

- `vtkCameraInterpolator = obj.GetInterpolator ()` - Get the vtkCameraInterpolator used to interpolate and save the sequence of camera views. If not defined, one is created automatically. Note that you can access this object to set the interpolation type (linear, spline) and other instance variables.

- `obj.SetNumberOfFrames (int )` - Set the number of frames to generate when playback is initiated.

- `int = obj.GetNumberOfFramesMinValue ()` - Set the number of frames to generate when playback is initiated.

- `int = obj.GetNumberOfFramesMaxValue ()` - Set the number of frames to generate when playback is initiated.

- `int = obj.GetNumberOfFrames ()` - Set the number of frames to generate when playback is initiated.

- `vtkProperty2D = obj.GetProperty ()` - By obtaining this property you can specify the properties of the representation.

- `obj.AddCameraToPath ()` - These methods are used to create interpolated camera paths. The AddCameraToPath() method adds the view defined by the current camera (via SetCamera()) to the interpolated camera path. AnimatePath() interpolates NumberOfFrames along the current path. InitializePath() resets the interpolated path to its initial, empty configuration.

- `obj.AnimatePath (vtkRenderWindowInteractor rwi)` - These methods are used to create interpolated camera paths. The AddCameraToPath() method adds the view defined by the current camera (via SetCamera()) to the interpolated camera path. AnimatePath() interpolates NumberOfFrames along the current path. InitializePath() resets the interpolated path to its initial, empty configuration.

- `obj.InitializePath ()` - These methods are used to create interpolated camera paths. The AddCameraToPath() method adds the view defined by the current camera (via SetCamera()) to the interpolated camera path. AnimatePath() interpolates NumberOfFrames along the current path. InitializePath() resets the interpolated path to its initial, empty configuration.

- `obj.BuildRepresentation ()` - Satisfy the superclasses' API.

- `obj.GetSize (double size[2])` - These methods are necessary to make this representation behave as a vtkProp.

- `obj.GetActors2D (vtkPropCollection )` - These methods are necessary to make this representation behave as a vtkProp.

- `obj.ReleaseGraphicsResources (vtkWindow )` - These methods are necessary to make this representation behave as a vtkProp.

- `int = obj.RenderOverlay (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.

- `int = obj.RenderOpaqueGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.

- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.

- `int = obj.HasTranslucentPolygonalGeometry ()` - These methods are necessary to make this representation behave as a vtkProp.

42.23 vtkCameraWidget

42.23.1 Usage

This class provides support for interactively saving a series of camera views into an interpolated path (using vtkCameraInterpolator). To use the class start by specifying a camera to interpolate, and then simply start recording by hitting the "record" button, manipulate the camera (by using an interactor, direct scripting, or any other means), and then save the camera view. Repeat this process to record a series of views. The user can then play back interpolated camera views using the vtkCameraInterpolator.

To create an instance of class vtkCameraWidget, simply invoke its constructor as follows

```
obj = vtkCameraWidget
```

42.23.2 Methods

The class vtkCameraWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCameraWidget class.

- `string = obj.GetClassName ()` - Standar VTK class methods.
- `int = obj.IsA (string name)` - Standar VTK class methods.
- `vtkCameraWidget = obj.NewInstance ()` - Standar VTK class methods.
- `vtkCameraWidget = obj.SafeDownCast (vtkObject o)` - Standar VTK class methods.
- `obj.SetRepresentation (vtkCameraRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.24 vtkCaptionRepresentation

42.24.1 Usage

This class represents vtkCaptionWidget. A caption is defined by some text with a leader (e.g., arrow) that points from the text to a point in the scene. The caption is defined by an instance of vtkCaptionActor2D. It uses the event bindings of its superclass (vtkBorderWidget) to control the placement of the text, and adds the ability to move the attachment point around. In addition, when the caption text is selected, the widget emits a ActivateEvent that observers can watch for. This is useful for opening GUI dialogoues to adjust font characteristics, etc. (Please see the superclass for a description of event bindings.)

Note that this widget extends the behavior of its superclass vtkBorderRepresentation.

To create an instance of class vtkCaptionRepresentation, simply invoke its constructor as follows

```
obj = vtkCaptionRepresentation
```

42.24.2 Methods

The class vtkCaptionRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCaptionRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK class methods.
- `int = obj.IsA (string name)` - Standard VTK class methods.

- `vtkCaptionRepresentation = obj.NewInstance ()` - Standard VTK class methods.
- `vtkCaptionRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK class methods.
- `obj.SetAnchorPosition (double pos[3])` - Specify the position of the anchor (i.e., the point that the caption is anchored to). Note that the position should be specified in world coordinates.
- `obj.GetAnchorPosition (double pos[3])` - Specify the position of the anchor (i.e., the point that the caption is anchored to). Note that the position should be specified in world coordinates.
- `obj.SetCaptionActor2D (vtkCaptionActor2D captionActor)` - Specify the vtkCaptionActor2D to manage. If not specified, then one is automatically created.
- `vtkCaptionActor2D = obj.GetCaptionActor2D ()` - Specify the vtkCaptionActor2D to manage. If not specified, then one is automatically created.
- `obj.SetAnchorRepresentation (vtkPointHandleRepresentation3D )` - Set and get the instances of vtkPointHandleRepresention3D used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `vtkPointHandleRepresentation3D = obj.GetAnchorRepresentation ()` - Set and get the instances of vtkPointHandleRepresention3D used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `obj.BuildRepresentation ()` - Satisfy the superclasses API.
- `obj.GetSize (double size[2])` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.GetActors2D (vtkPropCollection )` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.SetFontFactor (double )` - Set/Get the factor that controls the overall size of the fonts of the caption when the text actor's ScaledText is OFF
- `double = obj.GetFontFactorMinValue ()` - Set/Get the factor that controls the overall size of the fonts of the caption when the text actor's ScaledText is OFF
- `double = obj.GetFontFactorMaxValue ()` - Set/Get the factor that controls the overall size of the fonts of the caption when the text actor's ScaledText is OFF
- `double = obj.GetFontFactor ()` - Set/Get the factor that controls the overall size of the fonts of the caption when the text actor's ScaledText is OFF

42.25 vtkCaptionWidget

42.25.1 Usage

This class provides support for interactively placing a caption on the 2D overlay plane. A caption is defined by some text with a leader (e.g., arrow) that points from the text to a point in the scene. The caption is represented by a vtkCaptionRepresentation. It uses the event bindings of its superclass (vtkBorderWidget) to control the placement of the text, and adds the ability to move the attachment point around. In addition, when the caption text is selected, the widget emits a ActivateEvent that observers can watch for. This is useful for opening GUI dialogoues to adjust font characteristics, etc. (Please see the superclass for a description of event bindings.)

Note that this widget extends the behavior of its superclass vtkBorderWidget. The end point of the leader can be selected and moved around with an internal vtkHandleWidget.

To create an instance of class vtkCaptionWidget, simply invoke its constructor as follows

```
obj = vtkCaptionWidget
```

42.25.2 Methods

The class vtkCaptionWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCaptionWidget class.

- `string = obj.GetClassName ()` - Standard VTK class methods.
- `int = obj.IsA (string name)` - Standard VTK class methods.
- `vtkCaptionWidget = obj.NewInstance ()` - Standard VTK class methods.
- `vtkCaptionWidget = obj.SafeDownCast (vtkObject o)` - Standard VTK class methods.
- `obj.SetEnabled (int enabling)` - Override superclasses' SetEnabled() method because the caption leader has its own dedicated widget.
- `obj.SetRepresentation (vtkCaptionRepresentation r)` - Specify a vtkCaptionActor2D to manage. This is convenient, alternative method to SetRepresentation(). It internally create a vtkCaptionRepresentation and then invokes vtkCaptionRepresentation::SetCaptionActor2D().
- `obj.SetCaptionActor2D (vtkCaptionActor2D capActor)` - Specify a vtkCaptionActor2D to manage. This is convenient, alternative method to SetRepresentation(). It internally create a vtkCaptionRepresentation and then invokes vtkCaptionRepresentation::SetCaptionActor2D().
- `vtkCaptionActor2D = obj.GetCaptionActor2D ()` - Specify a vtkCaptionActor2D to manage. This is convenient, alternative method to SetRepresentation(). It internally create a vtkCaptionRepresentation and then invokes vtkCaptionRepresentation::SetCaptionActor2D().
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.26 vtkCenteredSliderRepresentation

42.26.1 Usage

This class is used to represent and render a vtkCenteredSliderWidget. To use this class, you must at a minimum specify the end points of the slider. Optional instance variable can be used to modify the appearance of the widget.

To create an instance of class vtkCenteredSliderRepresentation, simply invoke its constructor as follows

```
obj = vtkCenteredSliderRepresentation
```

42.26.2 Methods

The class vtkCenteredSliderRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCenteredSliderRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkCenteredSliderRepresentation = obj.NewInstance ()` - Standard methods for the class.
- `vtkCenteredSliderRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.
- `vtkCoordinate = obj.GetPoint1Coordinate ()` - Position the first end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point1Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.
- `vtkCoordinate = obj.GetPoint2Coordinate ()` - Position the second end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point2Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.
- `obj.SetTitleText (string )` - Specify the label text for this widget. If the value is not set, or set to the empty string "", then the label text is not displayed.
- `string = obj.GetTitleText ()` - Specify the label text for this widget. If the value is not set, or set to the empty string "", then the label text is not displayed.
- `vtkProperty2D = obj.GetTubeProperty ()` - Get the properties for the tube and slider
- `vtkProperty2D = obj.GetSliderProperty ()` - Get the properties for the tube and slider
- `vtkProperty2D = obj.GetSelectedProperty ()` - Get the selection property. This property is used to modify the appearance of selected objects (e.g., the slider).
- `vtkTextProperty = obj.GetLabelProperty ()` - Set/Get the properties for the label and title text.
- `obj.PlaceWidget (double bounds[6])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.BuildRepresentation ()` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.StartWidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.

- `obj.WidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.Highlight (int )` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.GetActors (vtkPropCollection )`
- `obj.ReleaseGraphicsResources (vtkWindow )`
- `int = obj.RenderOverlay (vtkViewport )`
- `int = obj.RenderOpaqueGeometry (vtkViewport )`

42.27 vtkCenteredSliderWidget

42.27.1 Usage

The vtkCenteredSliderWidget is used to adjust a scalar value in an application. This class measures deviations form the center point on the slider. Moving the slider modifies the value of the widget, which can be used to set parameters on other objects. Note that the actual appearance of the widget depends on the specific representation for the widget.

To use this widget, set the widget representation. The representation is assumed to consist of a tube, two end caps, and a slider (the details may vary depending on the particulars of the representation). Then in the representation you will typically set minimum and maximum value, as well as the current value. The position of the slider must also be set, as well as various properties.

Note that the value should be obtain from the widget, not from the representation. Also note that Minimum and Maximum values are in terms of value per second. The value you get from this widget's GetValue method is multiplied by time.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
If the slider bead is selected:
  LeftButtonPressEvent - select slider (if on slider)
  LeftButtonReleaseEvent - release slider (if selected)
  MouseMoveEvent - move slider
If the end caps or slider tube are selected:
  LeftButtonPressEvent - move (or animate) to cap or point on tube;
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkCenteredSliderWidget's widget events:

```
vtkWidgetEvent::Select -- some part of the widget has been selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Move -- a request for slider motion has been invoked
```

In turn, when these widget events are processed, the vtkCenteredSliderWidget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkCenteredSliderWidget, simply invoke its constructor as follows

```
obj = vtkCenteredSliderWidget
```

42.27.2 Methods

The class vtkCenteredSliderWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCenteredSliderWidget class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkCenteredSliderWidget = obj.NewInstance ()` - Standard macros.
- `vtkCenteredSliderWidget = obj.SafeDownCast (vtkObject o)` - Standard macros.
- `obj.SetRepresentation (vtkSliderRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `double = obj.GetValue ()` - Get the value fo this widget.

42.28 vtkCheckerboardRepresentation

42.28.1 Usage

The vtkCheckerboardRepresentation is used to implement the representation of the vtkCheckerboardWidget. The user can adjust the number of divisions in each of the i-j directions in a 2D image. A frame appears around the vtkImageActor with sliders along each side of the frame. The user can interactively adjust the sliders to the desired number of checkerboard subdivisions. The representation uses four instances of vtkSliderRepresentation3D to implement itself.

To create an instance of class vtkCheckerboardRepresentation, simply invoke its constructor as follows

```
obj = vtkCheckerboardRepresentation
```

42.28.2 Methods

The class vtkCheckerboardRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCheckerboardRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkCheckerboardRepresentation = obj.NewInstance ()` - Standard VTK methods.
- `vtkCheckerboardRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `obj.SetCheckerboard (vtkImageCheckerboard chkrbrd)` - Specify an instance of vtkImageCheckerboard to manipulate.
- `vtkImageCheckerboard = obj.GetCheckerboard ()` - Specify an instance of vtkImageCheckerboard to manipulate.
- `obj.SetImageActor (vtkImageActor imageActor)` - Specify an instance of vtkImageActor to decorate.
- `vtkImageActor = obj.GetImageActor ()` - Specify an instance of vtkImageActor to decorate.

- `obj.SetCornerOffset (double )` - Specify the offset of the ends of the sliders (on the boundary edges of the image) from the corner of the image. The offset is expressed as a normalized fraction of the border edges.
- `double = obj.GetCornerOffsetMinValue ()` - Specify the offset of the ends of the sliders (on the boundary edges of the image) from the corner of the image. The offset is expressed as a normalized fraction of the border edges.
- `double = obj.GetCornerOffsetMaxValue ()` - Specify the offset of the ends of the sliders (on the boundary edges of the image) from the corner of the image. The offset is expressed as a normalized fraction of the border edges.
- `double = obj.GetCornerOffset ()` - Specify the offset of the ends of the sliders (on the boundary edges of the image) from the corner of the image. The offset is expressed as a normalized fraction of the border edges.
- `obj.SliderValueChanged (int sliderNum)` - This method is invoked by the vtkCheckerboardWidget() when a value of some slider has changed.
- `obj.SetTopRepresentation (vtkSliderRepresentation3D )` - Set and get the instances of vtkSliderRepresention used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `obj.SetRightRepresentation (vtkSliderRepresentation3D )` - Set and get the instances of vtkSliderRepresention used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `obj.SetBottomRepresentation (vtkSliderRepresentation3D )` - Set and get the instances of vtkSliderRepresention used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `obj.SetLeftRepresentation (vtkSliderRepresentation3D )` - Set and get the instances of vtkSliderRepresention used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `vtkSliderRepresentation3D = obj.GetTopRepresentation ()` - Set and get the instances of vtkSliderRepresention used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `vtkSliderRepresentation3D = obj.GetRightRepresentation ()` - Set and get the instances of vtkSliderRepresention used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `vtkSliderRepresentation3D = obj.GetBottomRepresentation ()` - Set and get the instances of vtkSliderRepresention used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `vtkSliderRepresentation3D = obj.GetLeftRepresentation ()` - Set and get the instances of vtkSliderRepresention used to implement this representation. Normally default representations are created, but you can specify the ones you want to use.
- `obj.BuildRepresentation ()` - Methods required by superclass.
- `obj.GetActors (vtkPropCollection )` - Methods required by superclass.
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Methods required by superclass.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods required by superclass.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods required by superclass.

- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Methods required by superclass.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods required by superclass.

42.29 vtkCheckerboardWidget

42.29.1 Usage

The vtkCheckerboardWidget is used to interactively control an instance of vtkImageCheckerboard (and an associated vtkImageActor used to display the checkerboard). The user can adjust the number of divisions in each of the i-j directions in a 2D image. A frame appears around the vtkImageActor with sliders along each side of the frame. The user can interactively adjust the sliders to the desired number of checkerboard subdivisions.

To use this widget, specify an instance of vtkImageCheckerboard and an instance of vtkImageActor. By default, the widget responds to the following events:

```
If the slider bead is selected:
  LeftButtonPressEvent - select slider (if on slider)
  LeftButtonReleaseEvent - release slider
  MouseMoveEvent - move slider
If the end caps or slider tube of a slider are selected:
  LeftButtonPressEvent - jump (or animate) to cap or point on tube;
```

It is possible to change these event bindings. Please refer to the documentation for vtkSliderWidget for more information. Advanced users may directly access and manipulate the sliders by obtaining the instances of vtkSliderWidget composing the vtkCheckerboard widget.

To create an instance of class vtkCheckerboardWidget, simply invoke its constructor as follows

```
obj = vtkCheckerboardWidget
```

42.29.2 Methods

The class vtkCheckerboardWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkCheckerboardWidget class.

- `string = obj.GetClassName ()` - Standard methods for a VTK class.
- `int = obj.IsA (string name)` - Standard methods for a VTK class.
- `vtkCheckerboardWidget = obj.NewInstance ()` - Standard methods for a VTK class.
- `vtkCheckerboardWidget = obj.SafeDownCast (vtkObject o)` - Standard methods for a VTK class.
- `obj.SetEnabled (int )` - The method for activiating and deactiviating this widget. This method must be overridden because it is a composite widget and does more than its superclasses' vtkAbstractWidget::SetEnabled() method.
- `obj.SetRepresentation (vtkCheckerboardRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.30 vtkClosedSurfacePointPlacer

42.30.1 Usage

This placer takes a set of boudning planes and constraints the validity within the supplied convex planes. It is used by the ParallelopPipedRepresentation to place constraints on the motion the handles within the parallelopiped.

To create an instance of class vtkClosedSurfacePointPlacer, simply invoke its constructor as follows

```
obj = vtkClosedSurfacePointPlacer
```

42.30.2 Methods

The class vtkClosedSurfacePointPlacer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkClosedSurfacePointPlacer class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkClosedSurfacePointPlacer = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkClosedSurfacePointPlacer = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.AddBoundingPlane (vtkPlane plane)` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.
- `obj.RemoveBoundingPlane (vtkPlane plane)` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.
- `obj.RemoveAllBoundingPlanes ()` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.
- `obj.SetBoundingPlanes (vtkPlaneCollection )` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.
- `vtkPlaneCollection = obj.GetBoundingPlanes ()` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `obj.SetBoundingPlanes (vtkPlanes planes)` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double worldPos[3], double w` - Given a renderer and a display position, compute the world position and world orientation for this point. A plane is defined by a combination of the ProjectionNormal, ProjectionOrigin, and Oblique-Plane ivars. The display position is projected onto this plane to determine a world position, and the orientation is set to the normal of the plane. If the point cannot project onto the plane or if it falls outside the bounds imposed by the BoundingPlanes, then 0 is returned, otherwise 1 is returned to indicate a valid return position and orientation.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double refWorldPos[2], doubl` - Given a renderer, a display position and a reference position, "worldPos" is calculated as : Consider the line "L" that passes through the supplied "displayPos" and is parallel to the direction of projection of the camera. Clip this line segment with the parallelopiped, let's call it "L_segment". The computed world position, "worldPos" will be the point on "L_segment" that is closest to refWorldPos. NOTE: Note that a set of bounding planes must be supplied. The Oblique plane, if supplied is ignored.
- `int = obj.ValidateWorldPosition (double worldPos[3])` - Give a world position check if it is valid - does it lie on the plane and within the bounds? Returns 1 if it is valid, 0 otherwise.
- `int = obj.ValidateWorldPosition (double worldPos[3], double worldOrient[9])`
- `obj.SetMinimumDistance (double )`
- `double = obj.GetMinimumDistanceMinValue ()`
- `double = obj.GetMinimumDistanceMaxValue ()`
- `double = obj.GetMinimumDistance ()`

42.31 vtkConstrainedPointHandleRepresentation

42.31.1 Usage

This class is used to represent a vtkHandleWidget. It represents a position in 3D world coordinates that is constrained to a specified plane. The default look is to draw a white point when this widget is not selected or active, a thin green circle when it is highlighted, and a thicker cyan circle when it is active (being positioned). Defaults can be adjusted - but take care to define cursor geometry that makes sense for this widget. The geometry will be aligned on the constraining plane, with the plane normal aligned with the X axis of the geometry (similar behavior to vtkGlyph3D).

TODO: still need to work on 1) translation when mouse is outside bounding planes 2) size of the widget

To create an instance of class vtkConstrainedPointHandleRepresentation, simply invoke its constructor as follows

```
obj = vtkConstrainedPointHandleRepresentation
```

42.31.2 Methods

The class vtkConstrainedPointHandleRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkConstrainedPointHandleRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.

- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkConstrainedPointHandleRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkConstrainedPointHandleRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetCursorShape (vtkPolyData cursorShape)` - Specify the cursor shape. Keep in mind that the shape will be aligned with the constraining plane by orienting it such that the x axis of the geometry lies along the normal of the plane.
- `vtkPolyData = obj.GetCursorShape ()` - Specify the cursor shape. Keep in mind that the shape will be aligned with the constraining plane by orienting it such that the x axis of the geometry lies along the normal of the plane.
- `obj.SetActiveCursorShape (vtkPolyData activeShape)` - Specify the shape of the cursor (handle) when it is active. This is the geometry that will be used when the mouse is close to the handle or if the user is manipulating the handle.
- `vtkPolyData = obj.GetActiveCursorShape ()` - Specify the shape of the cursor (handle) when it is active. This is the geometry that will be used when the mouse is close to the handle or if the user is manipulating the handle.
- `obj.SetProjectionNormal (int )` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `int = obj.GetProjectionNormalMinValue ()` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `int = obj.GetProjectionNormalMaxValue ()` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `int = obj.GetProjectionNormal ()` - Set the projection normal to lie along the x, y, or z axis, or to be oblique. If it is oblique, then the plane is defined in the ObliquePlane ivar.
- `obj.SetProjectionNormalToXAxis ()`
- `obj.SetProjectionNormalToYAxis ()`
- `obj.SetProjectionNormalToZAxis ()`
- `obj.SetProjectionNormalToOblique ()` - If the ProjectionNormal is set to Oblique, then this is the oblique plane used to constrain the handle position
- `obj.SetObliquePlane (vtkPlane )` - If the ProjectionNormal is set to Oblique, then this is the oblique plane used to constrain the handle position
- `vtkPlane = obj.GetObliquePlane ()` - If the ProjectionNormal is set to Oblique, then this is the oblique plane used to constrain the handle position
- `obj.SetProjectionPosition (double position)` - The position of the bounding plane from the origin along the normal. The origin and normal are defined in the oblique plane when the ProjectionNormal is Oblique. For the X, Y, and Z axes projection normals, the normal is the axis direction, and the origin is (0,0,0).
- `double = obj.GetProjectionPosition ()` - The position of the bounding plane from the origin along the normal. The origin and normal are defined in the oblique plane when the ProjectionNormal is Oblique. For the X, Y, and Z axes projection normals, the normal is the axis direction, and the origin is (0,0,0).

- `obj.AddBoundingPlane (vtkPlane plane)` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `obj.RemoveBoundingPlane (vtkPlane plane)` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `obj.RemoveAllBoundingPlanes ()` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `obj.SetBoundingPlanes (vtkPlaneCollection )` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `vtkPlaneCollection = obj.GetBoundingPlanes ()` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `obj.SetBoundingPlanes (vtkPlanes planes)` - A collection of plane equations used to bound the position of the point. This is in addition to confining the point to a plane - these contraints are meant to, for example, keep a point within the extent of an image. Using a set of plane equations allows for more complex bounds (such as bounding a point to an oblique reliced image that has hexagonal shape) than a simple extent.

- `int = obj.CheckConstraint (vtkRenderer renderer, double pos[2])` - Overridden from the base class. It converts the display co-ordinates to world co-ordinates. It returns 1 if the point lies within the constrained region, otherwise return 0

- `obj.SetPosition (double x, double y, double z)` - Set/Get the position of the point in display coordinates. These are convenience methods that extend the superclasses' GetHandlePosition() method. Note that only the x-y coordinate values are used

- `obj.SetPosition (double xyz[3])` - Set/Get the position of the point in display coordinates. These are convenience methods that extend the superclasses' GetHandlePosition() method. Note that only the x-y coordinate values are used

- `obj.GetPosition (double xyz[3])` - Set/Get the position of the point in display coordinates. These are convenience methods that extend the superclasses' GetHandlePosition() method. Note that only the x-y coordinate values are used

- `vtkProperty = obj.GetProperty ()` - This is the property used when the handle is not active (the mouse is not near the handle)

- `vtkProperty = obj.GetSelectedProperty ()` - This is the property used when the mouse is near the handle (but the user is not yet interacting with it)

- `vtkProperty = obj.GetActiveProperty ()` - This is the property used when the user is interacting with the handle.
- `obj.SetRenderer (vtkRenderer ren)` - Subclasses of vtkConstrainedPointHandleRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.BuildRepresentation ()` - Subclasses of vtkConstrainedPointHandleRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.StartWidgetInteraction (double eventPos[2])` - Subclasses of vtkConstrainedPointHandleRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.WidgetInteraction (double eventPos[2])` - Subclasses of vtkConstrainedPointHandleRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Subclasses of vtkConstrainedPointHandleRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.SetDisplayPosition (double pos[3])` - Method overridden from Superclass. computes the world co-ordinates using GetIntersectionPosition()
- `obj.GetActors (vtkPropCollection )` - Methods to make this class behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods to make this class behave as a vtkProp.
- `obj.ShallowCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp.

42.32 vtkContinuousValueWidget

42.32.1 Usage

The vtkContinuousValueWidget is used to adjust a scalar value in an application. Note that the actual appearance of the widget depends on the specific representation for the widget.

To use this widget, set the widget representation. (the details may vary depending on the particulars of the representation).

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
If the slider bead is selected:
  LeftButtonPressEvent - select slider
  LeftButtonReleaseEvent - release slider
  MouseMoveEvent - move slider
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkContinuousValueWidget's widget events:

```
vtkWidgetEvent::Select -- some part of the widget has been selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Move -- a request for slider motion has been invoked
```

In turn, when these widget events are processed, the vtkContinuousValueWidget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkContinuousValueWidget, simply invoke its constructor as follows

```
obj = vtkContinuousValueWidget
```

42.32.2 Methods

The class vtkContinuousValueWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkContinuousValueWidget class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkContinuousValueWidget = obj.NewInstance ()` - Standard macros.
- `vtkContinuousValueWidget = obj.SafeDownCast (vtkObject o)` - Standard macros.
- `obj.SetRepresentation (vtkContinuousValueWidgetRepresentation r)` - Get the value for this widget.
- `double = obj.GetValue ()` - Get the value for this widget.
- `obj.SetValue (double v)` - Get the value for this widget.

42.33 vtkContinuousValueWidgetRepresentation

42.33.1 Usage

This class is used mainly as a superclass for continuous value widgets

To create an instance of class vtkContinuousValueWidgetRepresentation, simply invoke its constructor as follows

```
obj = vtkContinuousValueWidgetRepresentation
```

42.33.2 Methods

The class vtkContinuousValueWidgetRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkContinuousValueWidgetRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkContinuousValueWidgetRepresentation = obj.NewInstance ()` - Standard methods for the class.
- `vtkContinuousValueWidgetRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.
- `obj.PlaceWidget (double bounds[6])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.BuildRepresentation ()` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.StartWidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.WidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.
- `obj.SetValue (double value)`
- `double = obj.GetValue ()`

42.34 vtkContourLineInterpolator

42.34.1 Usage

vtkContourLineInterpolator is an abstract base class for interpolators that work are used by the contour representation class to interpolate and/or modify nodes in a contour. Subclasses must override the virtual method: ĮnterpolateLine. This is used by the contour representation to give the interpolator a chance to define an interpolation scheme between nodes. See vtkBezierContourLineInterpolator for a concrete implementation. Subclasses may also override, ǪpdateNode. This provides a way for the representation to give the interpolator a chance to modify the nodes, as the user constructs the contours. For instance a sticky contour widget may be implemented that moves nodes to nearby regions of high gradient, to be used in contour guided segmentation.

To create an instance of class vtkContourLineInterpolator, simply invoke its constructor as follows

```
obj = vtkContourLineInterpolator
```

42.34.2 Methods

The class vtkContourLineInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkContourLineInterpolator class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkContourLineInterpolator = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkContourLineInterpolator = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `int = obj.InterpolateLine (vtkRenderer ren, vtkContourRepresentation rep, int idx1, int idx2)` - Subclasses that wish to interpolate a line segment must implement this. For instance vtkBezierContourLineInterpolator adds nodes between idx1 and idx2, that allow the contour to adhere to a bezier curve.
- `int = obj.UpdateNode (vtkRenderer , vtkContourRepresentation , double , int )` - The interpolator is given a chance to update the node. For instance, the vtkImageContourLineInterpolator updates the idx'th node in the contour, so it automatically sticks to edges in the vicinity as the user constructs the contour. Returns 0 if the node (world position) is unchanged.
- `obj.GetSpan (int nodeIndex, vtkIntArray nodeIndices, vtkContourRepresentation rep)` - Span of the interpolator. ie. the number of control points its supposed to interpolate given a node.

  The first argument is the current nodeIndex. ie, you'd be trying to interpolate between nodes "nodeIndex" and "nodeIndex-1", unless you're closing the contour in which case, you're trying to interpolate "nodeIndex" and "Node=0".

  The node span is returned in a vtkIntArray. The default node span is 1 (ie. nodeIndices is a 2 tuple (nodeIndex, nodeIndex-1)). However, it need not always be 1. For instance, cubic spline interpolators, which have a span of 3 control points, it can be larger. See vtkBezierContourLineInterpolator for instance.

42.35 vtkContourRepresentation

42.35.1 Usage

The vtkContourRepresentation is a superclass for various types of representations for the vtkContourWidget.

.SECTION Managing contour points The classes vtkContourRepresentationNode, vtkContourRepresentationInternals, vtkContourRepresentationPoint manage the data structure used to represent nodes and points on a contour. A contour may contain several nodes and several more points. Nodes are usually the result of user clicked points on the contour. Additional points are created between nodes to generate a smooth curve using some Interpolator. See the method ŞetLineInterpolator. ar The data structure stores both the world and display positions for every point. (This may seem like a duplication.) The default behaviour of this class is to use the WorldPosition to do all the math. Typically a point is added at a given display position. Its corresponding world position is computed using the point placer and stored. Any query of the display position of a stored point is done via the Renderer, which computes the display position given a world position.

ar So why maintain the display position ? Consider drawing a contour on a volume widget. You might want the contour to be located at a certain world position in the volume or you might want to be overlayed over the window like an Actor2D. The default behaviour of this class is to provide the former behaviour.

ar To achieve the latter behaviour override the methods that return the display position (to return the set display position instead of computing it from the world positions) and the method ḂuildLines() to interpolate lines using their display positions intead of world positions.

To create an instance of class vtkContourRepresentation, simply invoke its constructor as follows

```
obj = vtkContourRepresentation
```

42.35.2 Methods

The class vtkContourRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkContourRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkContourRepresentation = obj.NewInstance ()` - Standard VTK methods.
- `vtkContourRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `int = obj.AddNodeAtWorldPosition (double x, double y, double z)` - Add a node at a specific world position. Returns 0 if the node could not be added, 1 otherwise.
- `int = obj.AddNodeAtWorldPosition (double worldPos[3])` - Add a node at a specific world position. Returns 0 if the node could not be added, 1 otherwise.
- `int = obj.AddNodeAtWorldPosition (double worldPos[3], double worldOrient[9])` - Add a node at a specific world position. Returns 0 if the node could not be added, 1 otherwise.
- `int = obj.AddNodeAtDisplayPosition (double displayPos[2])` - Add a node at a specific display position. This will be converted into a world position according to the current constraints of the point placer. Return 0 if a point could not be added, 1 otherwise.
- `int = obj.AddNodeAtDisplayPosition (int displayPos[2])` - Add a node at a specific display position. This will be converted into a world position according to the current constraints of the point placer. Return 0 if a point could not be added, 1 otherwise.
- `int = obj.AddNodeAtDisplayPosition (int X, int Y)` - Add a node at a specific display position. This will be converted into a world position according to the current constraints of the point placer. Return 0 if a point could not be added, 1 otherwise.
- `int = obj.ActivateNode (double displayPos[2])` - Given a display position, activate a node. The closest node within tolerance will be activated. If a node is activated, 1 will be returned, otherwise 0 will be returned.
- `int = obj.ActivateNode (int displayPos[2])` - Given a display position, activate a node. The closest node within tolerance will be activated. If a node is activated, 1 will be returned, otherwise 0 will be returned.
- `int = obj.ActivateNode (int X, int Y)` - Given a display position, activate a node. The closest node within tolerance will be activated. If a node is activated, 1 will be returned, otherwise 0 will be returned.
- `int = obj.SetActiveNodeToWorldPosition (double pos[3])`
- `int = obj.SetActiveNodeToWorldPosition (double pos[3], double orient[9])`
- `int = obj.SetActiveNodeToDisplayPosition (double pos[2])` - Move the active node based on a specified display position. The display position will be converted into a world position. If the new position is not valid or there is no active node, a 0 will be returned. Otherwise, on success a 1 will be returned.
- `int = obj.SetActiveNodeToDisplayPosition (int pos[2])` - Move the active node based on a specified display position. The display position will be converted into a world position. If the new position is not valid or there is no active node, a 0 will be returned. Otherwise, on success a 1 will be returned.

- `int = obj.SetActiveNodeToDisplayPosition (int X, int Y)` - Move the active node based on a specified display position. The display position will be converted into a world position. If the new position is not valid or there is no active node, a 0 will be returned. Otherwise, on success a 1 will be returned.
- `int = obj.ToggleActiveNodeSelected ()` - Set/Get whether the active or nth node is selected.
- `int = obj.GetActiveNodeSelected ()` - Set/Get whether the active or nth node is selected.
- `int = obj.GetNthNodeSelected (int )` - Set/Get whether the active or nth node is selected.
- `int = obj.SetNthNodeSelected (int )` - Set/Get whether the active or nth node is selected.
- `int = obj.GetActiveNodeWorldPosition (double pos[3])` - Get the world position of the active node. Will return 0 if there is no active node, or 1 otherwise.
- `int = obj.GetActiveNodeWorldOrientation (double orient[9])` - Get the world orientation of the active node. Will return 0 if there is no active node, or 1 otherwise.
- `int = obj.GetActiveNodeDisplayPosition (double pos[2])` - Get the display position of the active node. Will return 0 if there is no active node, or 1 otherwise.
- `int = obj.GetNumberOfNodes ()` - Get the number of nodes.
- `int = obj.GetNthNodeDisplayPosition (int n, double pos[2])` - Get the nth node's display position. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting).
- `int = obj.GetNthNodeWorldPosition (int n, double pos[3])` - Get the nth node's world position. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting).
- `int = obj.GetNthNodeWorldOrientation (int n, double orient[9])` - Get the nth node's world orientation. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting).
- `int = obj.SetNthNodeDisplayPosition (int n, int X, int Y)` - Set the nth node's display position. Display position will be converted into world position according to the constraints of the point placer. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting) or the world position is not valid.
- `int = obj.SetNthNodeDisplayPosition (int n, int pos[2])` - Set the nth node's display position. Display position will be converted into world position according to the constraints of the point placer. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting) or the world position is not valid.
- `int = obj.SetNthNodeDisplayPosition (int n, double pos[2])` - Set the nth node's display position. Display position will be converted into world position according to the constraints of the point placer. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting) or the world position is not valid.
- `int = obj.SetNthNodeWorldPosition (int n, double pos[3])` - Set the nth node's world position. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting) or the world position is not valid according to the point placer.
- `int = obj.SetNthNodeWorldPosition (int n, double pos[3], double orient[9])` - Set the nth node's world position. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting) or the world position is not valid according to the point placer.
- `int = obj.GetNthNodeSlope (int idx, double slope[3])` - Get the nth node's slope. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting).
- `int = obj.GetNumberOfIntermediatePoints (int n)`

- `int = obj.GetIntermediatePointWorldPosition (int n, int idx, double point[3])` - Get the world position of the intermediate point at index idx between nodes n and (n+1) (or n and 0 if n is the last node and the loop is closed). Returns 1 on success or 0 if n or idx are out of range.
- `int = obj.AddIntermediatePointWorldPosition (int n, double point[3])` - Add an intermediate point between node n and n+1 (or n and 0 if n is the last node and the loop is closed). Returns 1 on success or 0 if n is out of range.
- `int = obj.DeleteLastNode ()` - Delete the last node. Returns 1 on success or 0 if there were not any nodes.
- `int = obj.DeleteActiveNode ()` - Delete the active node. Returns 1 on success or 0 if the active node did not indicate a valid node.
- `int = obj.DeleteNthNode (int n)` - Delete the nth node. Return 1 on success or 0 if n is out of range.
- `obj.ClearAllNodes ()` - Delete all nodes.
- `int = obj.AddNodeOnContour (int X, int Y)` - Given a specific X, Y pixel location, add a new node on the contour at this location.
- `obj.SetPixelTolerance (int )` - The tolerance to use when calculations are performed in display coordinates
- `int = obj.GetPixelToleranceMinValue ()` - The tolerance to use when calculations are performed in display coordinates
- `int = obj.GetPixelToleranceMaxValue ()` - The tolerance to use when calculations are performed in display coordinates
- `int = obj.GetPixelTolerance ()` - The tolerance to use when calculations are performed in display coordinates
- `obj.SetWorldTolerance (double )` - The tolerance to use when calculations are performed in world coordinates
- `double = obj.GetWorldToleranceMinValue ()` - The tolerance to use when calculations are performed in world coordinates
- `double = obj.GetWorldToleranceMaxValue ()` - The tolerance to use when calculations are performed in world coordinates
- `double = obj.GetWorldTolerance ()` - The tolerance to use when calculations are performed in world coordinates
- `int = obj.GetCurrentOperation ()` - Set / get the current operation. The widget is either inactive, or it is being translated.
- `obj.SetCurrentOperation (int )` - Set / get the current operation. The widget is either inactive, or it is being translated.
- `int = obj.GetCurrentOperationMinValue ()` - Set / get the current operation. The widget is either inactive, or it is being translated.
- `int = obj.GetCurrentOperationMaxValue ()` - Set / get the current operation. The widget is either inactive, or it is being translated.
- `obj.SetCurrentOperationToInactive ()` - Set / get the current operation. The widget is either inactive, or it is being translated.

- `obj.SetCurrentOperationToTranslate ()` - Set / get the current operation. The widget is either inactive, or it is being translated.
- `obj.SetCurrentOperationToShift ()` - Set / get the current operation. The widget is either inactive, or it is being translated.
- `obj.SetCurrentOperationToScale ()`
- `obj.SetPointPlacer (vtkPointPlacer )`
- `vtkPointPlacer = obj.GetPointPlacer ()`
- `obj.SetLineInterpolator (vtkContourLineInterpolator )` - Set / Get the Line Interpolator. The line interpolator is responsible for generating the line segments connecting nodes.
- `vtkContourLineInterpolator = obj.GetLineInterpolator ()` - Set / Get the Line Interpolator. The line interpolator is responsible for generating the line segments connecting nodes.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `int = obj.ComputeInteractionState (int X, int Y, int modified)` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.StartWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Methods required by vtkProp superclass.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods required by vtkProp superclass.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods required by vtkProp superclass.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Methods required by vtkProp superclass.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods required by vtkProp superclass.
- `obj.SetClosedLoop (int val)` - Set / Get the ClosedLoop value. This ivar indicates whether the contour forms a closed loop.
- `int = obj.GetClosedLoop ()` - Set / Get the ClosedLoop value. This ivar indicates whether the contour forms a closed loop.
- `obj.ClosedLoopOn ()` - Set / Get the ClosedLoop value. This ivar indicates whether the contour forms a closed loop.
- `obj.ClosedLoopOff ()` - Set / Get the ClosedLoop value. This ivar indicates whether the contour forms a closed loop.
- `obj.SetShowSelectedNodes (int )` - A flag to indicate whether to show the Selected nodes Default is to set it to false.
- `int = obj.GetShowSelectedNodes ()` - A flag to indicate whether to show the Selected nodes Default is to set it to false.
- `obj.ShowSelectedNodesOn ()` - A flag to indicate whether to show the Selected nodes Default is to set it to false.

- `obj.ShowSelectedNodesOff ()` - A flag to indicate whether to show the Selected nodes Default is to set it to false.
- `obj.GetNodePolyData (vtkPolyData poly)` - Get the nodes and not the intermediate points in this contour as a vtkPolyData.

42.36 vtkContourWidget

42.36.1 Usage

The vtkContourWidget is used to select a set of points, and draw lines between these points. The contour may be opened or closed, depending on how the last point is added. The widget handles all processing of widget events (that are triggered by VTK events). The vtkContourRepresentation is responsible for all placement of the points, calculation of the lines, and contour manipulation. This is done through two main helper classes: vtkPointPlacer and vtkContourLineInterpolator. The representation is also responsible for drawing the points and lines.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
LeftButtonPressEvent - triggers a Select event
RightButtonPressEvent - triggers a AddFinalPoint event
MouseMoveEvent - triggers a Move event
LeftButtonReleaseEvent - triggers an EndSelect event
Delete key event - triggers a Delete event
Shift + Delete key event - triggers a Reset event
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkContourWidget's widget events:

```
vtkWidgetEvent::Select
     widget state is:
         Start or
         Define: If we already have at least 2 nodes, test
              whether the current (X,Y) location is near an existing
              node. If so, close the contour and change to Manipulate
              state. Otherwise, attempt to add a node at this (X,Y)
              location.
         Manipulate: If this (X,Y) location activates a node, then
              set the current operation to Translate. Otherwise, if
              this location is near the contour, attempt to add a
              new node on the contour at this (X,Y) location.

vtkWidgetEvent::AddFinalPoint
     widget state is:
         Start: Do nothing.
         Define: If we already have at least 2 nodes, test
              whether the current (X,Y) location is near an existing
              node. If so, close the contour and change to Manipulate
              state. Otherwise, attempt to add a node at this (X,Y)
              location. If we do, then leave the contour open and
              change to Manipulate state.
         Manipulate: Do nothing.

vtkWidgetEvent::Move
```

```
        widget state is:
            Start or
            Define: Do nothing.
            Manipulate: If our operation is Translate, then invoke
                  WidgetInteraction() on the representation. If our
                  operation is Inactive, then just attempt to activate
                  a node at this (X,Y) location.

    vtkWidgetEvent::EndSelect
        widget state is:
            Start or
            Define: Do nothing.
            Manipulate: If our operation is not Inactive, set it to
                  Inactive.

    vtkWidgetEvent::Delete
        widget state is:
            Start: Do nothing.
            Define: Remove the last point on the contour.
            Manipulate: Attempt to activate a node at (X,Y). If
                   we do activate a node, delete it. If we now
                   have less than 3 nodes, go back to Define state.

    vtkWidgetEvent::Reset
        widget state is:
            Start: Do nothing.
            Define: Remove all points and line segments of the contour.
                 Essentially calls Intialize(NULL)
            Manipulate: Do nothing.
```

This widget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (beginning to interact)
vtkCommand::EndInteractionEvent (completing interaction)
vtkCommand::InteractionEvent (moving after selecting something)
vtkCommand::PlacePointEvent (after point is positioned;
                             call data includes handle id (0,1))
vtkCommand::WidgetValueChangedEvent (Invoked when the contour is closed
                                     for the first time. )
```

To create an instance of class vtkContourWidget, simply invoke its constructor as follows

```
obj = vtkContourWidget
```

42.36.2 Methods

The class vtkContourWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkContourWidget class.

- `string = obj.GetClassName ()` - Standard methods for a VTK class.
- `int = obj.IsA (string name)` - Standard methods for a VTK class.

- `vtkContourWidget = obj.NewInstance ()` - Standard methods for a VTK class.
- `vtkContourWidget = obj.SafeDownCast (vtkObject o)` - Standard methods for a VTK class.
- `obj.SetEnabled (int )` - The method for activiating and deactiviating this widget. This method must be overridden because it is a composite widget and does more than its superclasses' vtkAbstractWidget::SetEnabled() method.
- `obj.SetRepresentation (vtkContourRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `obj.CloseLoop ()` - Convenient method to close the contour loop.
- `obj.SetAllowNodePicking (int )` - Set / Get the AllowNodePicking value. This ivar indicates whether the nodes and points between nodes can be picked/un-picked by Ctrl+Click on the node.
- `int = obj.GetAllowNodePicking ()` - Set / Get the AllowNodePicking value. This ivar indicates whether the nodes and points between nodes can be picked/un-picked by Ctrl+Click on the node.
- `obj.AllowNodePickingOn ()` - Set / Get the AllowNodePicking value. This ivar indicates whether the nodes and points between nodes can be picked/un-picked by Ctrl+Click on the node.
- `obj.AllowNodePickingOff ()` - Set / Get the AllowNodePicking value. This ivar indicates whether the nodes and points between nodes can be picked/un-picked by Ctrl+Click on the node.
- `obj.SetFollowCursor (int )` - Follow the cursor ? If this is ON, during definition, the last node of the contour will automatically follow the cursor, without waiting for the the point to be dropped. This may be useful for some interpolators, such as the live-wire interpolator to see the shape of the contour that will be placed as you move the mouse cursor.
- `int = obj.GetFollowCursor ()` - Follow the cursor ? If this is ON, during definition, the last node of the contour will automatically follow the cursor, without waiting for the the point to be dropped. This may be useful for some interpolators, such as the live-wire interpolator to see the shape of the contour that will be placed as you move the mouse cursor.
- `obj.FollowCursorOn ()` - Follow the cursor ? If this is ON, during definition, the last node of the contour will automatically follow the cursor, without waiting for the the point to be dropped. This may be useful for some interpolators, such as the live-wire interpolator to see the shape of the contour that will be placed as you move the mouse cursor.
- `obj.FollowCursorOff ()` - Follow the cursor ? If this is ON, during definition, the last node of the contour will automatically follow the cursor, without waiting for the the point to be dropped. This may be useful for some interpolators, such as the live-wire interpolator to see the shape of the contour that will be placed as you move the mouse cursor.
- `obj.SetContinuousDraw (int )` - Define a contour by continuously drawing with the mouse cursor. Press and hold the left mouse button down to continuously draw. Releasing the left mouse button switches into a snap drawing mode. Terminate the contour by pressing the right mouse button. If you do not want to see the nodes as they are added to the contour, set the opacity to 0 of the representation's property. If you do not want to see the last active node as it is being added, set the opacity to 0 of the representation's active property.
- `int = obj.GetContinuousDraw ()` - Define a contour by continuously drawing with the mouse cursor. Press and hold the left mouse button down to continuously draw. Releasing the left mouse button switches into a snap drawing mode. Terminate the contour by pressing the right mouse button. If you do not want to see the nodes as they are added to the contour, set the opacity to 0 of the representation's property. If you do not want to see the last active node as it is being added, set the opacity to 0 of the representation's active property.

- `obj.ContinuousDrawOn ()` - Define a contour by continuously drawing with the mouse cursor. Press and hold the left mouse button down to continuously draw. Releasing the left mouse button switches into a snap drawing mode. Terminate the contour by pressing the right mouse button. If you do not want to see the nodes as they are added to the contour, set the opacity to 0 of the representation's property. If you do not want to see the last active node as it is being added, set the opacity to 0 of the representation's active property.

- `obj.ContinuousDrawOff ()` - Define a contour by continuously drawing with the mouse cursor. Press and hold the left mouse button down to continuously draw. Releasing the left mouse button switches into a snap drawing mode. Terminate the contour by pressing the right mouse button. If you do not want to see the nodes as they are added to the contour, set the opacity to 0 of the representation's property. If you do not want to see the last active node as it is being added, set the opacity to 0 of the representation's active property.

- `obj.Initialize (vtkPolyData poly, int state)` - Initialize the contour widget from a user supplied set of points. The state of the widget decides if you are still defining the widget, or if you've finished defining (added the last point) are manipulating it. Note that if the polydata supplied is closed, the state will be set to manipulate. State: Define = 0, Manipulate = 1.

- `obj.Initialize ()`

42.37 vtkDijkstraImageContourLineInterpolator

42.37.1 Usage

vtkDijkstraImageContourLineInterpolator interpolates and places contour points on images. The class interpolates nodes by computing a graph lying on the image data. By graph, we mean that the line interpolating the two end points traverses along pixels so as to form a shortest path. A Dijkstra algorithm is used to compute the path.

The class is meant to be used in conjunction with vtkImageActorPointPlacer. One reason for this coupling is a performance issue: both classes need to perform a cell pick, and coupling avoids multiple cell picks (cell picks are slow). Another issue is that the interpolator may need to set the image input to its vtkDijkstraImageGeodesicPath ivar.

To create an instance of class vtkDijkstraImageContourLineInterpolator, simply invoke its constructor as follows

```
obj = vtkDijkstraImageContourLineInterpolator
```

42.37.2 Methods

The class vtkDijkstraImageContourLineInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDijkstraImageContourLineInterpolator class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.

- `int = obj.IsA (string name)` - Standard methods for instances of this class.

- `vtkDijkstraImageContourLineInterpolator = obj.NewInstance ()` - Standard methods for instances of this class.

- `vtkDijkstraImageContourLineInterpolator = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.

- `int = obj.InterpolateLine (vtkRenderer ren, vtkContourRepresentation rep, int idx1, int idx2)` - Subclasses that wish to interpolate a line segment must implement this. For instance vtkBezierContourLineInterpolator adds nodes between idx1 and idx2, that allow the contour to adhere to a bezier curve.

- `obj.SetCostImage (vtkImageData )` - Set the image data for the vtkDijkstraImageGeodesicPath. If not set, the interpolator uses the image data input to the image actor. The image actor is obtained from the expected vtkImageActorPointPlacer.

- `vtkImageData = obj.GetCostImage ()` - Set the image data for the vtkDijkstraImageGeodesicPath. If not set, the interpolator uses the image data input to the image actor. The image actor is obtained from the expected vtkImageActorPointPlacer.

- `vtkDijkstraImageGeodesicPath = obj.GetDijkstraImageGeodesicPath ()` - access to the internal dijkstra path

42.38 vtkDistanceRepresentation

42.38.1 Usage

The vtkDistanceRepresentation is a superclass for various types of representations for the vtkDistanceWidget. Logically subclasses consist of an axis and two handles for placing/manipulating the end points.

To create an instance of class vtkDistanceRepresentation, simply invoke its constructor as follows

```
obj = vtkDistanceRepresentation
```

42.38.2 Methods

The class vtkDistanceRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDistanceRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK methods.

- `int = obj.IsA (string name)` - Standard VTK methods.

- `vtkDistanceRepresentation = obj.NewInstance ()` - Standard VTK methods.

- `vtkDistanceRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.

- `double = obj.GetDistance ()` - This representation and all subclasses must keep a distance consistent with the state of the widget.

- `obj.GetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.GetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `double = obj.GetPoint1WorldPosition ()` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `double = obj.GetPoint2WorldPosition ()` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.GetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.GetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetHandleRepresentation (vtkHandleRepresentation handle)` - This method is used to specify the type of handle representation to use for the two internal vtkHandleWidgets within vtkDistanceWidget. To use this method, create a dummy vtkHandleWidget (or subclass), and then invoke this method with this dummy. Then the vtkDistanceRepresentation uses this dummy to clone two vtkHandleWidgets of the same type. Make sure you set the handle representation before the widget is enabled. (The method InstantiateHandleRepresentation() is invoked by the vtkDistance widget.)

- `obj.InstantiateHandleRepresentation ()` - This method is used to specify the type of handle representation to use for the two internal vtkHandleWidgets within vtkDistanceWidget. To use this method, create a dummy vtkHandleWidget (or subclass), and then invoke this method with this dummy. Then the vtkDistanceRepresentation uses this dummy to clone two vtkHandleWidgets of the same type. Make sure you set the handle representation before the widget is enabled. (The method InstantiateHandleRepresentation() is invoked by the vtkDistance widget.)

- `vtkHandleRepresentation = obj.GetPoint1Representation ()` - Set/Get the two handle representations used for the vtkDistanceWidget. (Note: properties can be set by grabbing these representations and setting the properties appropriately.)

- `vtkHandleRepresentation = obj.GetPoint2Representation ()` - Set/Get the two handle representations used for the vtkDistanceWidget. (Note: properties can be set by grabbing these representations and setting the properties appropriately.)

- `obj.SetTolerance (int )` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the end points of the widget to be active.

- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the end points of the widget to be active.

- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the end points of the widget to be active.

- `int = obj.GetTolerance ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the end points of the widget to be active.

- `obj.SetLabelFormat (string )` - Specify the format to use for labelling the distance. Note that an empty string results in no label, or a format string without a "will not print the distance value.
- `string = obj.GetLabelFormat ()` - Specify the format to use for labelling the distance. Note that an empty string results in no label, or a format string without a "will not print the distance value.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.StartWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.

42.39 vtkDistanceRepresentation2D

42.39.1 Usage

The vtkDistanceRepresentation2D is a representation for the vtkDistanceWidget. This representation consists of a measuring line (axis) and two vtkHandleWidgets to place the end points of the line. Note that this particular widget draws its representation in the overlay plane.

To create an instance of class vtkDistanceRepresentation2D, simply invoke its constructor as follows

```
obj = vtkDistanceRepresentation2D
```

42.39.2 Methods

The class vtkDistanceRepresentation2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDistanceRepresentation2D class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkDistanceRepresentation2D = obj.NewInstance ()` - Standard VTK methods.
- `vtkDistanceRepresentation2D = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `double = obj.GetDistance ()` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.SetPoint1DisplayPosition (double pos[3])`
- `obj.SetPoint2DisplayPosition (double pos[3])`
- `obj.GetPoint1DisplayPosition (double pos[3])`
- `obj.GetPoint2DisplayPosition (double pos[3])`
- `vtkAxisActor2D = obj.GetAxis ()` - Retrieve the vtkAxisActor2D used to draw the measurement axis. With this properties can be set and so on.
- `obj.BuildRepresentation ()` - Method to satisfy superclasses' API.
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Methods required by vtkProp superclass.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods required by vtkProp superclass.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods required by vtkProp superclass.

42.40 vtkDistanceWidget

42.40.1 Usage

The vtkDistanceWidget is used to measure the distance between two points. The two end points can be positioned independently, and when they are released, a special PlacePointEvent is invoked so that special operations may be take to reposition the point (snap to grid, etc.) The widget has two different modes of interaction: when initially defined (i.e., placing the two points) and then a manipulate mode (adjusting the position of the two points).

To use this widget, specify an instance of vtkDistanceWidget and a representation (a subclass of vtkDistanceRepresentation). The widget is implemented using two instances of vtkHandleWidget which are used to position the end points of the line. The representations for these two handle widgets are provided by the vtkDistanceRepresentation.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
LeftButtonPressEvent - add a point or select a handle
MouseMoveEvent - position the second point or move a handle
LeftButtonReleaseEvent - release the handle
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkDistanceWidget's widget events:

```
vtkWidgetEvent::AddPoint -- add one point; depending on the state
                            it may the first or second point added. Or,
                            if near a handle, select the handle.
vtkWidgetEvent::Move -- move the second point or handle depending on the state.
vtkWidgetEvent::EndSelect -- the handle manipulation process has completed.
```

This widget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (beginning to interact)
vtkCommand::EndInteractionEvent (completing interaction)
vtkCommand::InteractionEvent (moving after selecting something)
vtkCommand::PlacePointEvent (after point is positioned;
                             call data includes handle id (0,1))
```

To create an instance of class vtkDistanceWidget, simply invoke its constructor as follows

```
obj = vtkDistanceWidget
```

42.40.2 Methods

The class vtkDistanceWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkDistanceWidget class.

- `string = obj.GetClassName ()` - Standard methods for a VTK class.
- `int = obj.IsA (string name)` - Standard methods for a VTK class.
- `vtkDistanceWidget = obj.NewInstance ()` - Standard methods for a VTK class.
- `vtkDistanceWidget = obj.SafeDownCast (vtkObject o)` - Standard methods for a VTK class.
- `obj.SetEnabled (int )` - The method for activiating and deactiviating this widget. This method must be overridden because it is a composite widget and does more than its superclasses' vtkAbstractWidget::SetEnabled() method.
- `obj.SetRepresentation (vtkDistanceRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `obj.SetProcessEvents (int )` - Methods to change the whether the widget responds to interaction. Overridden to pass the state to component widgets.

42.41 vtkEllipsoidTensorProbeRepresentation

42.41.1 Usage

vtkEllipsoidTensorProbeRepresentation is a concrete implementation of vtkTensorProbeRepresentation. It renders tensors as ellipsoids. Locations between two points when probed have the tensors linearly interpolated from the neighboring locations on the polyline.

To create an instance of class vtkEllipsoidTensorProbeRepresentation, simply invoke its constructor as follows

```
obj = vtkEllipsoidTensorProbeRepresentation
```

42.41.2 Methods

The class vtkEllipsoidTensorProbeRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEllipsoidTensorProbeRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.

- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkEllipsoidTensorProbeRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkEllipsoidTensorProbeRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.BuildRepresentation ()`
- `int = obj.RenderOpaqueGeometry (vtkViewport )`
- `int = obj.SelectProbe (int pos[2])` - Can we pick the tensor glyph at the current cursor pos
- `obj.GetActors (vtkPropCollection )` - See vtkProp for details.
- `obj.ReleaseGraphicsResources (vtkWindow )` - See vtkProp for details.

42.42 vtkEvent

42.42.1 Usage

vtkEvent is a class that fully describes a VTK event. It is used by the widgets to help specify the mapping between VTK events and widget events.

To create an instance of class vtkEvent, simply invoke its constructor as follows

```
obj = vtkEvent
```

42.42.2 Methods

The class vtkEvent has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkEvent class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkEvent = obj.NewInstance ()` - Standard macros.
- `vtkEvent = obj.SafeDownCast (vtkObject o)` - Standard macros.
- `obj.SetEventId (long )` - Set the modifier for the event.
- `long = obj.GetEventId ()` - Set the modifier for the event.
- `obj.SetModifier (int )` - Set the modifier for the event.
- `int = obj.GetModifier ()` - Set the modifier for the event.
- `obj.SetKeyCode (char )` - Set the KeyCode for the event.
- `char = obj.GetKeyCode ()` - Set the KeyCode for the event.
- `obj.SetRepeatCount (int )` - Set the repease count for the event.
- `int = obj.GetRepeatCount ()` - Set the repease count for the event.
- `obj.SetKeySym (string )` - Set the complex key symbol (compound key strokes) for the event.
- `string = obj.GetKeySym ()` - Set the complex key symbol (compound key strokes) for the event.

42.43 vtkFocalPlaneContourRepresentation

42.43.1 Usage

The contour will stay on the focal plane irrespective of camera position/orientation changes. The class was written in order to be able to draw contours on a volume widget and have the contours overlayed on the focal plane in order to do contour segmentation. The superclass, vtkContourRepresentation handles contours that are drawn in actual world position co-ordinates, so they would rotate with the camera position/ orientation changes

To create an instance of class vtkFocalPlaneContourRepresentation, simply invoke its constructor as follows

```
obj = vtkFocalPlaneContourRepresentation
```

42.43.2 Methods

The class vtkFocalPlaneContourRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFocalPlaneContourRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkFocalPlaneContourRepresentation = obj.NewInstance ()` - Standard VTK methods.
- `vtkFocalPlaneContourRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `int = obj.GetIntermediatePointWorldPosition (int n, int idx, double point[3])` - Get the world position of the intermediate point at index idx between nodes n and (n+1) (or n and 0 if n is the last node and the loop is closed). Returns 1 on success or 0 if n or idx are out of range.
- `int = obj.GetIntermediatePointDisplayPosition (int n, int idx, double point[3])` - Get the world position of the intermediate point at index idx between nodes n and (n+1) (or n and 0 if n is the last node and the loop is closed). Returns 1 on success or 0 if n or idx are out of range.
- `int = obj.GetNthNodeDisplayPosition (int n, double pos[2])` - Get the nth node's display position. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting).
- `int = obj.GetNthNodeWorldPosition (int n, double pos[3])` - Get the nth node's world position. Will return 1 on success, or 0 if there are not at least (n+1) nodes (0 based counting).
- `obj.UpdateContourWorldPositionsBasedOnDisplayPositions ()` - The class maintains its true contour locations based on display co-ords This method syncs the world co-ords data structure with the display co-ords.
- `int = obj.UpdateContour ()` - The method must be called whenever the contour needs to be updated, usually from RenderOpaqueGeometry()
- `obj.UpdateLines (int index)`

42.44 vtkFocalPlanePointPlacer

42.44.1 Usage

To create an instance of class vtkFocalPlanePointPlacer, simply invoke its constructor as follows

```
obj = vtkFocalPlanePointPlacer
```

42.44.2 Methods

The class vtkFocalPlanePointPlacer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkFocalPlanePointPlacer class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkFocalPlanePointPlacer = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkFocalPlanePointPlacer = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double worldPos[3], double w`
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double refWorldPos[3], doubl` - Given a renderer, a display position, and a reference world position, compute a new world position. The orientation will be the standard coordinate axes, and the computed world position will be created by projecting the display point onto a plane that is parallel to the focal plane and runs through the reference world position. This method is typically used to move existing points.
- `int = obj.ValidateWorldPosition (double worldPos[3])` - Validate a world position. All world positions are valid so these methods always return 1.
- `int = obj.ValidateWorldPosition (double worldPos[3], double worldOrient[9])` - Validate a world position. All world positions are valid so these methods always return 1.
- `obj.SetOffset (double )` - Optionally specify a signed offset from the focal plane for the points to be placed at. If negative, the constraint plane is offset closer to the camera. If positive, its further away from the camera.
- `double = obj.GetOffset ()` - Optionally specify a signed offset from the focal plane for the points to be placed at. If negative, the constraint plane is offset closer to the camera. If positive, its further away from the camera.
- `obj.SetPointBounds (double , double , double , double , double , double )` - Optionally Restrict the points to a set of bounds. The placer will invalidate points outside these bounds.
- `obj.SetPointBounds (double  a[6])` - Optionally Restrict the points to a set of bounds. The placer will invalidate points outside these bounds.
- `double = obj. GetPointBounds ()` - Optionally Restrict the points to a set of bounds. The placer will invalidate points outside these bounds.

42.45 vtkHandleRepresentation

42.45.1 Usage

This class defines an API for widget handle representations. These representations interact with vtkHandleWidget. Various representations can be used depending on the nature of the handle. The basic functionality of the handle representation is to maintain a position. The position is represented via a vtkCoordinate, meaning that the position can be easily obtained in a variety of coordinate systems.

Optional features for this representation include an active mode (the widget appears only when the mouse pointer is close to it). The active distance is expressed in pixels and represents a circle in display space.

The class may be subclassed so that alternative representations can be created. The class defines an API and a default implementation that the vtkHandleWidget interacts with to render itself in the scene.

To create an instance of class vtkHandleRepresentation, simply invoke its constructor as follows

```
obj = vtkHandleRepresentation
```

42.45.2 Methods

The class vtkHandleRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHandleRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkHandleRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkHandleRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetDisplayPosition (double pos[3])` - Handles usually have their coordinates set in display coordinates (generally by an associated widget) and internally maintain the position in world coordinates. (Using world coordinates insures that handles are rendered in the right position when the camera view changes.) These methods are often subclassed because special constraint operations can be used to control the actual positioning.
- `obj.GetDisplayPosition (double pos[3])` - Handles usually have their coordinates set in display coordinates (generally by an associated widget) and internally maintain the position in world coordinates. (Using world coordinates insures that handles are rendered in the right position when the camera view changes.) These methods are often subclassed because special constraint operations can be used to control the actual positioning.
- `double = obj.GetDisplayPosition ()` - Handles usually have their coordinates set in display coordinates (generally by an associated widget) and internally maintain the position in world coordinates. (Using world coordinates insures that handles are rendered in the right position when the camera view changes.) These methods are often subclassed because special constraint operations can be used to control the actual positioning.
- `obj.SetWorldPosition (double pos[3])` - Handles usually have their coordinates set in display coordinates (generally by an associated widget) and internally maintain the position in world coordinates. (Using world coordinates insures that handles are rendered in the right position when the camera view changes.) These methods are often subclassed because special constraint operations can be used to control the actual positioning.
- `obj.GetWorldPosition (double pos[3])` - Handles usually have their coordinates set in display coordinates (generally by an associated widget) and internally maintain the position in world coordinates. (Using world coordinates insures that handles are rendered in the right position when the camera view changes.) These methods are often subclassed because special constraint operations can be used to control the actual positioning.
- `double = obj.GetWorldPosition ()` - Handles usually have their coordinates set in display coordinates (generally by an associated widget) and internally maintain the position in world coordinates. (Using world coordinates insures that handles are rendered in the right position when the camera view changes.) These methods are often subclassed because special constraint operations can be used to control the actual positioning.
- `obj.SetTolerance (int )` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the widget to be active.
- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the widget to be active.

- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the widget to be active.
- `int = obj.GetTolerance ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the widget to be active.
- `obj.SetActiveRepresentation (int )` - Flag controls whether the widget becomes visible when the mouse pointer moves close to it (i.e., the widget becomes active). By default, ActiveRepresentation is off and the representation is always visible.
- `int = obj.GetActiveRepresentation ()` - Flag controls whether the widget becomes visible when the mouse pointer moves close to it (i.e., the widget becomes active). By default, ActiveRepresentation is off and the representation is always visible.
- `obj.ActiveRepresentationOn ()` - Flag controls whether the widget becomes visible when the mouse pointer moves close to it (i.e., the widget becomes active). By default, ActiveRepresentation is off and the representation is always visible.
- `obj.ActiveRepresentationOff ()` - Flag controls whether the widget becomes visible when the mouse pointer moves close to it (i.e., the widget becomes active). By default, ActiveRepresentation is off and the representation is always visible.
- `obj.SetInteractionState (int )` - The interaction state may be set from a widget (e.g., HandleWidget) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking processwith the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.
- `int = obj.GetInteractionStateMinValue ()` - The interaction state may be set from a widget (e.g., HandleWidget) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking processwith the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.
- `int = obj.GetInteractionStateMaxValue ()` - The interaction state may be set from a widget (e.g., HandleWidget) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking processwith the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.
- `obj.SetConstrained (int )` - Specify whether any motions (such as scale, translate, etc.) are constrained in some way (along an axis, etc.) Widgets can use this to control the resulting motion.
- `int = obj.GetConstrained ()` - Specify whether any motions (such as scale, translate, etc.) are constrained in some way (along an axis, etc.) Widgets can use this to control the resulting motion.
- `obj.ConstrainedOn ()` - Specify whether any motions (such as scale, translate, etc.) are constrained in some way (along an axis, etc.) Widgets can use this to control the resulting motion.
- `obj.ConstrainedOff ()` - Specify whether any motions (such as scale, translate, etc.) are constrained in some way (along an axis, etc.) Widgets can use this to control the resulting motion.
- `int = obj.CheckConstraint (vtkRenderer renderer, double pos[2])` - Method has to be overridden in the subclasses which has constraints on placing the handle (Ex. vtkConstrainedPointHandleRepresentation). It should return 1 if the position is within the constraint, else it should return 0. By default it returns 1.
- `obj.ShallowCopy (vtkProp prop)` - Methods to make this class properly act like a vtkWidgetRepresentation.

- `obj.DeepCopy (vtkProp prop)` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.SetRenderer (vtkRenderer ren)` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `long = obj.GetMTime ()` - Overload the superclasses' GetMTime() because the internal vtkCoordinates are used to keep the state of the representation.
- `obj.SetPointPlacer (vtkPointPlacer )` - Set/Get the point placer. Point placers can be used to dictate constraints on the placement of handles. As an example, see vtkBoundedPlanePointPlacer (constrains the placement of handles to a set of bounded planes) vtkFocalPlanePointPlacer (constrains placement on the focal plane) etc. The default point placer is vtkPointPlacer (which does not apply any constraints, so the handles are free to move anywhere).
- `vtkPointPlacer = obj.GetPointPlacer ()` - Set/Get the point placer. Point placers can be used to dictate constraints on the placement of handles. As an example, see vtkBoundedPlanePointPlacer (constrains the placement of handles to a set of bounded planes) vtkFocalPlanePointPlacer (constrains placement on the focal plane) etc. The default point placer is vtkPointPlacer (which does not apply any constraints, so the handles are free to move anywhere).

42.46 vtkHandleWidget

42.46.1 Usage

The vtkHandleWidget is used to position a handle. A handle is a widget with a position (in display and world space). Various appearances are available depending on its associated representation. The widget provides methods for translation, including constrained translation along coordinate axes. To use this widget, create and associate a representation with the widget.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
LeftButtonPressEvent - select focal point of widget
LeftButtonReleaseEvent - end selection
MiddleButtonPressEvent - translate widget
MiddleButtonReleaseEvent - end translation
RightButtonPressEvent - scale widget
RightButtonReleaseEvent - end scaling
MouseMoveEvent - interactive movement across widget
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkHandleWidget's widget events:

```
vtkWidgetEvent::Select -- focal point is being selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Translate -- translate the widget
vtkWidgetEvent::EndTranslate -- end widget translation
vtkWidgetEvent::Scale -- scale the widget
vtkWidgetEvent::EndScale -- end scaling the widget
vtkWidgetEvent::Move -- a request for widget motion
```

In turn, when these widget events are processed, the vtkHandleWidget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkHandleWidget, simply invoke its constructor as follows

```
obj = vtkHandleWidget
```

42.46.2 Methods

The class vtkHandleWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHandleWidget class.

- `string = obj.GetClassName ()` - Standard VTK class macros.
- `int = obj.IsA (string name)` - Standard VTK class macros.
- `vtkHandleWidget = obj.NewInstance ()` - Standard VTK class macros.
- `vtkHandleWidget = obj.SafeDownCast (vtkObject o)` - Standard VTK class macros.
- `obj.SetRepresentation (vtkHandleRepresentation r)` - Create the default widget representation if one is not set. By default an instance of vtkPointHandleRepresenation3D is created.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set. By default an instance of vtkPointHandleRepresenation3D is created.
- `obj.SetEnableAxisConstraint (int )` - Enable / disable axis constrained motion of the handles. By default the widget responds to the shift modifier to constrain the handle along the axis closest aligned with the motion vector.
- `int = obj.GetEnableAxisConstraint ()` - Enable / disable axis constrained motion of the handles. By default the widget responds to the shift modifier to constrain the handle along the axis closest aligned with the motion vector.
- `obj.EnableAxisConstraintOn ()` - Enable / disable axis constrained motion of the handles. By default the widget responds to the shift modifier to constrain the handle along the axis closest aligned with the motion vector.
- `obj.EnableAxisConstraintOff ()` - Enable / disable axis constrained motion of the handles. By default the widget responds to the shift modifier to constrain the handle along the axis closest aligned with the motion vector.
- `obj.SetAllowHandleResize (int )` - Allow resizing of handles ? By default the right mouse button scales the handle size.
- `int = obj.GetAllowHandleResize ()` - Allow resizing of handles ? By default the right mouse button scales the handle size.
- `obj.AllowHandleResizeOn ()` - Allow resizing of handles ? By default the right mouse button scales the handle size.
- `obj.AllowHandleResizeOff ()` - Allow resizing of handles ? By default the right mouse button scales the handle size.
- `int = obj.GetWidgetState ()` - Get the widget state.

42.47 vtkHoverWidget

42.47.1 Usage

The vtkHoverWidget is used to invoke an event when hovering in a render window. Hovering occurs when mouse motion (in the render window) does not occur for a specified amount of time (i.e., TimerDuration). This class can be used as is (by observing TimerEvents) or for class derivation for those classes wishing to do more with the hover event.

To use this widget, specify an instance of vtkHoverWidget and specify the time (in milliseconds) defining the hover period. Unlike most widgets, this widget does not require a representation (although subclasses like vtkBalloonWidget do require a representation).

.SECTION Event Bindings By default, the widget observes the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
MouseMoveEvent - manages a timer used to determine whether the mouse
                 is hovering.
TimerEvent - when the time between events (e.g., mouse move), then a
             timer event is invoked.
KeyPressEvent - when the ''Enter'' key is pressed after the balloon appears,
                a callback is activited (e.g., WidgetActivateEvent).
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkHoverWidget's widget events:

```
vtkWidgetEvent::Move -- start (or reset) the timer
vtkWidgetEvent::TimedOut -- when enough time is elapsed between defined
                            VTK events the hover event is invoked.
vtkWidgetEvent::SelectAction -- activate any callbacks associated
                                with the balloon.
```

This widget invokes the following VTK events on itself when the widget determines that it is hovering. Note that observers of this widget can listen for these events and take appropriate action.

```
vtkCommand::TimerEvent (when hovering is determined to occur)
vtkCommand::EndInteractionEvent (after a hover has occured and the
                                 mouse begins moving again).
vtkCommand::WidgetActivateEvent (when the balloon is selected with a
                                 keypress).
```

To create an instance of class vtkHoverWidget, simply invoke its constructor as follows

```
obj = vtkHoverWidget
```

42.47.2 Methods

The class vtkHoverWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkHoverWidget class.

- `string = obj.GetClassName ()` - Standard methods for a VTK class.
- `int = obj.IsA (string name)` - Standard methods for a VTK class.
- `vtkHoverWidget = obj.NewInstance ()` - Standard methods for a VTK class.

- `vtkHoverWidget = obj.SafeDownCast (vtkObject o)` - Standard methods for a VTK class.
- `obj.SetTimerDuration (int )` - Specify the hovering interval (in milliseconds). If after moving the mouse the pointer stays over a vtkProp for this duration, then a vtkTimerEvent::TimerEvent is invoked.
- `int = obj.GetTimerDurationMinValue ()` - Specify the hovering interval (in milliseconds). If after moving the mouse the pointer stays over a vtkProp for this duration, then a vtkTimerEvent::TimerEvent is invoked.
- `int = obj.GetTimerDurationMaxValue ()` - Specify the hovering interval (in milliseconds). If after moving the mouse the pointer stays over a vtkProp for this duration, then a vtkTimerEvent::TimerEvent is invoked.
- `int = obj.GetTimerDuration ()` - Specify the hovering interval (in milliseconds). If after moving the mouse the pointer stays over a vtkProp for this duration, then a vtkTimerEvent::TimerEvent is invoked.
- `obj.SetEnabled (int )` - The method for activiating and deactiviating this widget. This method must be overridden because it performs special timer-related operations.
- `obj.CreateDefaultRepresentation ()`

42.48 vtkImageActorPointPlacer

42.48.1 Usage

This PointPlacer is used to constrain the placement of points on the supplied image actor. Additionally, you may set bounds to restrict the placement of the points. The placement of points will then be constrained to lie not only on the ImageActor but also within the bounds specified. If no bounds are specified, they may lie anywhere on the supplied ImageActor.

To create an instance of class vtkImageActorPointPlacer, simply invoke its constructor as follows

```
obj = vtkImageActorPointPlacer
```

42.48.2 Methods

The class vtkImageActorPointPlacer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageActorPointPlacer class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkImageActorPointPlacer = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkImageActorPointPlacer = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double worldPos[3], double w` - Given and renderer and a display position in pixels, find a world position and orientation. In this class an internal vtkBoundedPlanePointPlacer is used to compute the world position and orientation. The internal placer is set to use the plane of the image actor and the bounds of the image actor as the constraints for placing points.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double refWorldPos[2], doubl` - This method is identical to the one above since the reference position is ignored by the bounded plane point placer.

- `int = obj.ValidateWorldPosition (double worldPos[3])` - This method validates a world position by checking to see if the world position is valid according to the constraints of the internal placer (essentially - is this world position on the image?)
- `int = obj.ValidateWorldPosition (double worldPos[3], double worldOrient[9])` - This method is identical to the one above since the bounded plane point placer ignores orientation
- `int = obj.UpdateWorldPosition (vtkRenderer ren, double worldPos[3], double worldOrient[9])` - Update the world position and orientation according the the current constraints of the placer. Will be called by the representation when it notices that this placer has been modified.
- `int = obj.UpdateInternalState ()` - A method for configuring the internal placer according to the constraints of the image actor. Called by the representation to give the placer a chance to update itself, which may cause the MTime to change, which would then cause the representation to update all of its points
- `obj.SetImageActor (vtkImageActor )` - Set / get the reference vtkImageActor used to place the points. An image actor must be set for this placer to work. An internal bounded plane point placer is created and set to match the bounds of the displayed image.
- `vtkImageActor = obj.GetImageActor ()` - Set / get the reference vtkImageActor used to place the points. An image actor must be set for this placer to work. An internal bounded plane point placer is created and set to match the bounds of the displayed image.
- `obj.SetBounds (double , double , double , double , double , double )` - Optionally, you may set bounds to restrict the placement of the points. The placement of points will then be constrained to lie not only on the ImageActor but also within the bounds specified. If no bounds are specified, they may lie anywhere on the supplied ImageActor.
- `obj.SetBounds (double a[6])` - Optionally, you may set bounds to restrict the placement of the points. The placement of points will then be constrained to lie not only on the ImageActor but also within the bounds specified. If no bounds are specified, they may lie anywhere on the supplied ImageActor.
- `double = obj. GetBounds ()` - Optionally, you may set bounds to restrict the placement of the points. The placement of points will then be constrained to lie not only on the ImageActor but also within the bounds specified. If no bounds are specified, they may lie anywhere on the supplied ImageActor.
- `obj.SetWorldTolerance (double s)` - Set the world tolerance. This propagates it to the internal BoundedPlanePointPlacer.

42.49 vtkImageOrthoPlanes

42.49.1 Usage

vtkImageOrthoPlanes is an event observer class that listens to the events from three vtkImagePlaneWidgets and keeps their orientations and scales synchronized.

To create an instance of class vtkImageOrthoPlanes, simply invoke its constructor as follows

```
obj = vtkImageOrthoPlanes
```

42.49.2 Methods

The class vtkImageOrthoPlanes has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageOrthoPlanes class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageOrthoPlanes = obj.NewInstance ()`
- `vtkImageOrthoPlanes = obj.SafeDownCast (vtkObject o)`
- `obj.SetPlane (int i, vtkImagePlaneWidget imagePlaneWidget)` - You must set three planes for the widget.
- `vtkImagePlaneWidget = obj.GetPlane (int i)` - You must set three planes for the widget.
- `obj.ResetPlanes ()` - Reset the planes to original scale, rotation, and location.
- `vtkTransform = obj.GetTransform ()` - Get the transform for the planes.
- `obj.HandlePlaneEvent (vtkImagePlaneWidget imagePlaneWidget)` - A public method to be used only by the event callback.

42.50 vtkImagePlaneWidget

42.50.1 Usage

This 3D widget defines a plane that can be interactively placed in an image volume. A nice feature of the object is that the vtkImagePlaneWidget, like any 3D widget, will work with the current interactor style. That is, if vtkImagePlaneWidget does not handle an event, then all other registered observers (including the interactor style) have an opportunity to process the event. Otherwise, the vtkImagePlaneWidget will terminate the processing of the event that it handles.

The core functionality of the widget is provided by a vtkImageReslice object which passes its output onto a texture mapping pipeline for fast slicing through volumetric data. See the key methods: GenerateTexturePlane() and UpdatePlane() for implementation details.

To use this object, just invoke SetInteractor() with the argument of the method a vtkRenderWindowInteractor. You may also wish to invoke "PlaceWidget()" to initially position the widget. If the "i" key (for "interactor") is pressed, the vtkImagePlaneWidget will appear. (See superclass documentation for information about changing this behavior.)

Selecting the widget with the middle mouse button with and without holding the shift or control keys enables complex reslicing capablilites. To facilitate use, a set of 'margins' (left, right, top, bottom) are shown as a set of plane-axes aligned lines, the properties of which can be changed as a group. Without keyboard modifiers: selecting in the middle of the margins enables translation of the plane along its normal. Selecting one of the corners within the margins enables spinning around the plane's normal at its center. Selecting within a margin allows rotating about the center of the plane around an axis aligned with the margin (i.e., selecting left margin enables rotating around the plane's local y-prime axis). With control key modifier: margin selection enables edge translation (i.e., a constrained form of scaling). Selecting within the margins enables translation of the entire plane. With shift key modifier: uniform plane scaling is enabled. Moving the mouse up enlarges the plane while downward movement shrinks it.

Window-level is achieved by using the right mouse button. Window-level values can be reset by shift + 'r' or control + 'r' while regular reset camera is maintained with 'r' or 'R'. The left mouse button can be used to query the underlying image data with a snap-to cross-hair cursor. Currently, the nearest point in the input image data to the mouse cursor generates the cross-hairs. With oblique slicing, this behaviour may appear unsatisfactory. Text display of window-level and image coordinates/data values are provided by a text actor/mapper pair.

Events that occur outside of the widget (i.e., no part of the widget is picked) are propagated to any other registered obsevers (such as the interaction style). Turn off the widget by pressing the "i" key again (or invoke the Off() method). To support interactive manipulation of objects, this class invokes the events StartInteractionEvent, InteractionEvent, and EndInteractionEvent as well as StartWindowLevelEvent, WindowLevelEvent, EndWindowLevelEvent and ResetWindowLevelEvent.

The vtkImagePlaneWidget has several methods that can be used in conjunction with other VTK objects. The GetPolyData() method can be used to get the polygonal representation of the plane and can be used as input for other VTK objects. Typical usage of the widget is to make use of the StartInteractionEvent, InteractionEvent, and EndInteractionEvent events. The InteractionEvent is called on mouse motion; the other two events are called on button down and button up (either left or right button).

Some additional features of this class include the ability to control the properties of the widget. You can set the properties of: the selected and unselected representations of the plane's outline; the text actor via its vtkTextProperty; the cross-hair cursor. In addition there are methods to constrain the plane so that it is aligned along the x-y-z axes. Finally, one can specify the degree of interpolation (vtkImageReslice): nearest neighbour, linear, and cubic.

To create an instance of class vtkImagePlaneWidget, simply invoke its constructor as follows

```
obj = vtkImagePlaneWidget
```

42.50.2 Methods

The class vtkImagePlaneWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImagePlaneWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImagePlaneWidget = obj.NewInstance ()`
- `vtkImagePlaneWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double bounds[6])` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget ()` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Set the vtkImageData* input for the vtkImageReslice.
- `obj.SetInput (vtkDataSet input)` - Set the vtkImageData* input for the vtkImageReslice.
- `obj.SetOrigin (double x, double y, double z)` - Set/Get the origin of the plane.
- `obj.SetOrigin (double xyz[3])` - Set/Get the origin of the plane.
- `double = obj.GetOrigin ()` - Set/Get the origin of the plane.
- `obj.GetOrigin (double xyz[3])` - Set/Get the origin of the plane.
- `obj.SetPoint1 (double x, double y, double z)` - Set/Get the position of the point defining the first axis of the plane.
- `obj.SetPoint1 (double xyz[3])` - Set/Get the position of the point defining the first axis of the plane.
- `double = obj.GetPoint1 ()` - Set/Get the position of the point defining the first axis of the plane.
- `obj.GetPoint1 (double xyz[3])` - Set/Get the position of the point defining the first axis of the plane.
- `obj.SetPoint2 (double x, double y, double z)` - Set/Get the position of the point defining the second axis of the plane.

- `obj.SetPoint2 (double xyz[3])` - Set/Get the position of the point defining the second axis of the plane.
- `double = obj.GetPoint2 ()` - Set/Get the position of the point defining the second axis of the plane.
- `obj.GetPoint2 (double xyz[3])` - Set/Get the position of the point defining the second axis of the plane.
- `double = obj.GetCenter ()` - Get the center of the plane.
- `obj.GetCenter (double xyz[3])` - Get the center of the plane.
- `double = obj.GetNormal ()` - Get the normal to the plane.
- `obj.GetNormal (double xyz[3])` - Get the normal to the plane.
- `obj.GetVector1 (double v1[3])` - Get the vector from the plane origin to point1.
- `obj.GetVector2 (double v2[3])` - Get the vector from the plane origin to point2.
- `int = obj.GetSliceIndex ()` - Get the slice position in terms of the data extent.
- `obj.SetSliceIndex (int index)` - Set the slice position in terms of the data extent.
- `double = obj.GetSlicePosition ()` - Get the position of the slice along its normal.
- `obj.SetSlicePosition (double position)` - Set the position of the slice along its normal.
- `obj.SetResliceInterpolate (int )` - Set the interpolation to use when texturing the plane.
- `int = obj.GetResliceInterpolate ()` - Set the interpolation to use when texturing the plane.
- `obj.SetResliceInterpolateToNearestNeighbour ()` - Set the interpolation to use when texturing the plane.
- `obj.SetResliceInterpolateToLinear ()` - Set the interpolation to use when texturing the plane.
- `obj.SetResliceInterpolateToCubic ()` - Convenience method to get the vtkImageReslice output.
- `vtkImageData = obj.GetResliceOutput ()` - Convenience method to get the vtkImageReslice output.
- `obj.SetRestrictPlaneToVolume (int )` - Make sure that the plane remains within the volume. Default is On.
- `int = obj.GetRestrictPlaneToVolume ()` - Make sure that the plane remains within the volume. Default is On.
- `obj.RestrictPlaneToVolumeOn ()` - Make sure that the plane remains within the volume. Default is On.
- `obj.RestrictPlaneToVolumeOff ()` - Make sure that the plane remains within the volume. Default is On.
- `obj.SetUserControlledLookupTable (int )` - Let the user control the lookup table. NOTE: apply this method BEFORE applying the SetLookupTable method. Default is Off.
- `int = obj.GetUserControlledLookupTable ()` - Let the user control the lookup table. NOTE: apply this method BEFORE applying the SetLookupTable method. Default is Off.
- `obj.UserControlledLookupTableOn ()` - Let the user control the lookup table. NOTE: apply this method BEFORE applying the SetLookupTable method. Default is Off.

- `obj.UserControlledLookupTableOff ()` - Let the user control the lookup table. NOTE: apply this method BEFORE applying the SetLookupTable method. Default is Off.

- `obj.SetTextureInterpolate (int )` - Specify whether to interpolate the texture or not. When off, the reslice interpolation is nearest neighbour regardless of how the interpolation is set through the API. Set before setting the vtkImageData input. Default is On.

- `int = obj.GetTextureInterpolate ()` - Specify whether to interpolate the texture or not. When off, the reslice interpolation is nearest neighbour regardless of how the interpolation is set through the API. Set before setting the vtkImageData input. Default is On.

- `obj.TextureInterpolateOn ()` - Specify whether to interpolate the texture or not. When off, the reslice interpolation is nearest neighbour regardless of how the interpolation is set through the API. Set before setting the vtkImageData input. Default is On.

- `obj.TextureInterpolateOff ()` - Specify whether to interpolate the texture or not. When off, the reslice interpolation is nearest neighbour regardless of how the interpolation is set through the API. Set before setting the vtkImageData input. Default is On.

- `obj.SetTextureVisibility (int )` - Control the visibility of the actual texture mapped reformatted plane. in some cases you may only want the plane outline for example.

- `int = obj.GetTextureVisibility ()` - Control the visibility of the actual texture mapped reformatted plane. in some cases you may only want the plane outline for example.

- `obj.TextureVisibilityOn ()` - Control the visibility of the actual texture mapped reformatted plane. in some cases you may only want the plane outline for example.

- `obj.TextureVisibilityOff ()` - Control the visibility of the actual texture mapped reformatted plane. in some cases you may only want the plane outline for example.

- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that defines the plane. The polydata consists of (res+1)*(res+1) points, and res*res quadrilateral polygons, where res is the resolution of the plane. These point values are guaranteed to be up-to-date when either the InteractionEvent or EndInteraction events are invoked. The user provides the vtkPolyData and the points and polygons are added to it.

- `vtkPolyDataAlgorithm = obj.GetPolyDataAlgorithm ()` - Satisfies superclass API. This returns a pointer to the underlying vtkPolyData. Make changes to this before calling the initial PlaceWidget() to have the initial placement follow suit. Or, make changes after the widget has been initialised and call UpdatePlacement() to realise.

- `obj.UpdatePlacement (void )` - Satisfies superclass API. This will change the state of the widget to match changes that have been made to the underlying vtkPolyDataSource

- `vtkTexture = obj.GetTexture ()` - Convenience method to get the texture used by this widget. This can be used in external slice viewers.

- `vtkImageMapToColors = obj.GetColorMap ()` - Convenience method to get the vtkImageMapToColors filter used by this widget. The user can properly render other transparent actors in a scene by calling the filter's SetOutputFormatToRGB and PassAlphaToOutputOff.

- `obj.SetColorMap (vtkImageMapToColors )` - Convenience method to get the vtkImageMapToColors filter used by this widget. The user can properly render other transparent actors in a scene by calling the filter's SetOutputFormatToRGB and PassAlphaToOutputOff.

- `obj.SetPlaneProperty (vtkProperty )` - Set/Get the plane's outline properties. The properties of the plane's outline when selected and unselected can be manipulated.

- `vtkProperty = obj.GetPlaneProperty ()` - Set/Get the plane's outline properties. The properties of the plane's outline when selected and unselected can be manipulated.
- `obj.SetSelectedPlaneProperty (vtkProperty )` - Set/Get the plane's outline properties. The properties of the plane's outline when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedPlaneProperty ()` - Set/Get the plane's outline properties. The properties of the plane's outline when selected and unselected can be manipulated.
- `obj.SetPlaneOrientation (int )` - Convenience method sets the plane orientation normal to the x, y, or z axes. Default is XAxes (0).
- `int = obj.GetPlaneOrientation ()` - Convenience method sets the plane orientation normal to the x, y, or z axes. Default is XAxes (0).
- `obj.SetPlaneOrientationToXAxes ()` - Convenience method sets the plane orientation normal to the x, y, or z axes. Default is XAxes (0).
- `obj.SetPlaneOrientationToYAxes ()` - Convenience method sets the plane orientation normal to the x, y, or z axes. Default is XAxes (0).
- `obj.SetPlaneOrientationToZAxes ()` - Set the internal picker to one defined by the user. In this way, a set of three orthogonal planes can share the same picker so that picking is performed correctly. The default internal picker can be re-set/allocated by setting to 0 (NULL).
- `obj.SetPicker (vtkAbstractPropPicker )` - Set the internal picker to one defined by the user. In this way, a set of three orthogonal planes can share the same picker so that picking is performed correctly. The default internal picker can be re-set/allocated by setting to 0 (NULL).
- `obj.SetLookupTable (vtkLookupTable )` - Set/Get the internal lookuptable (lut) to one defined by the user, or, alternatively, to the lut of another vtkImgePlaneWidget. In this way, a set of three orthogonal planes can share the same lut so that window-levelling is performed uniformly among planes. The default internal lut can be re- set/allocated by setting to 0 (NULL).
- `vtkLookupTable = obj.GetLookupTable ()` - Set/Get the internal lookuptable (lut) to one defined by the user, or, alternatively, to the lut of another vtkImgePlaneWidget. In this way, a set of three orthogonal planes can share the same lut so that window-levelling is performed uniformly among planes. The default internal lut can be re- set/allocated by setting to 0 (NULL).
- `obj.SetDisplayText (int )` - Enable/disable text display of window-level, image coordinates and scalar values in a render window.
- `int = obj.GetDisplayText ()` - Enable/disable text display of window-level, image coordinates and scalar values in a render window.
- `obj.DisplayTextOn ()` - Enable/disable text display of window-level, image coordinates and scalar values in a render window.
- `obj.DisplayTextOff ()` - Enable/disable text display of window-level, image coordinates and scalar values in a render window.
- `obj.SetCursorProperty (vtkProperty )` - Set the properties of the cross-hair cursor.
- `vtkProperty = obj.GetCursorProperty ()` - Set the properties of the cross-hair cursor.
- `obj.SetMarginProperty (vtkProperty )` - Set the properties of the margins.
- `vtkProperty = obj.GetMarginProperty ()` - Set the properties of the margins.
- `obj.SetMarginSizeX (double )` - Set the size of the margins based on a percentage of the plane's width and height, limited between 0 and 50

- `double = obj.GetMarginSizeXMinValue ()` - Set the size of the margins based on a percentage of the plane's width and height, limited between 0 and 50
- `double = obj.GetMarginSizeXMaxValue ()` - Set the size of the margins based on a percentage of the plane's width and height, limited between 0 and 50
- `double = obj.GetMarginSizeX ()` - Set the size of the margins based on a percentage of the plane's width and height, limited between 0 and 50
- `obj.SetMarginSizeY (double )` - Set the size of the margins based on a percentage of the plane's width and height, limited between 0 and 50
- `double = obj.GetMarginSizeYMinValue ()` - Set the size of the margins based on a percentage of the plane's width and height, limited between 0 and 50
- `double = obj.GetMarginSizeYMaxValue ()` - Set the size of the margins based on a percentage of the plane's width and height, limited between 0 and 50
- `double = obj.GetMarginSizeY ()` - Set the size of the margins based on a percentage of the plane's width and height, limited between 0 and 50
- `obj.SetTextProperty (vtkTextProperty tprop)` - Set/Get the text property for the image data and window-level annotation.
- `vtkTextProperty = obj.GetTextProperty ()` - Set/Get the text property for the image data and window-level annotation.
- `obj.SetTexturePlaneProperty (vtkProperty )` - Set/Get the property for the resliced image.
- `vtkProperty = obj.GetTexturePlaneProperty ()` - Set/Get the property for the resliced image.
- `obj.SetWindowLevel (double window, double level, int copy)` - Set/Get the current window and level values. SetWindowLevel should only be called after SetInput. If a shared lookup table is being used, a callback is required to update the window level values without having to update the lookup table again.
- `obj.GetWindowLevel (double wl[2])` - Set/Get the current window and level values. SetWindowLevel should only be called after SetInput. If a shared lookup table is being used, a callback is required to update the window level values without having to update the lookup table again.
- `double = obj.GetWindow ()` - Set/Get the current window and level values. SetWindowLevel should only be called after SetInput. If a shared lookup table is being used, a callback is required to update the window level values without having to update the lookup table again.
- `double = obj.GetLevel ()` - Get the image coordinate position and voxel value. Currently only supports single component image data.
- `int = obj.GetCursorData (double xyzv[4])` - Get the image coordinate position and voxel value. Currently only supports single component image data.
- `int = obj.GetCursorDataStatus ()` - Get the status of the cursor data. If this returns 1 the CurrentCursorPosition and CurrentImageValue will have current data. If it returns 0, these values are invalid.
- `double = obj. GetCurrentCursorPosition ()` - Get the current cursor position. To be used in conjunction with GetCursorDataStatus.
- `double = obj.GetCurrentImageValue ()` - Get the current image value at the current cursor position. To be used in conjunction with GetCursorDataStatus. The value is VTK_DOUBLE_MAX when the data is invalid.

- `obj.SetUseContinuousCursor (int )` - Choose between voxel centered or continuous cursor probing. With voxel centered probing, the cursor snaps to the nearest voxel and the reported cursor coordinates are extent based. With continuous probing, voxel data is interpolated using vtkDataSetAttributes' InterpolatePoint method and the reported coordinates are 3D spatial continuous.

- `int = obj.GetUseContinuousCursor ()` - Choose between voxel centered or continuous cursor probing. With voxel centered probing, the cursor snaps to the nearest voxel and the reported cursor coordinates are extent based. With continuous probing, voxel data is interpolated using vtkDataSetAttributes' InterpolatePoint method and the reported coordinates are 3D spatial continuous.

- `obj.UseContinuousCursorOn ()` - Choose between voxel centered or continuous cursor probing. With voxel centered probing, the cursor snaps to the nearest voxel and the reported cursor coordinates are extent based. With continuous probing, voxel data is interpolated using vtkDataSetAttributes' InterpolatePoint method and the reported coordinates are 3D spatial continuous.

- `obj.UseContinuousCursorOff ()` - Choose between voxel centered or continuous cursor probing. With voxel centered probing, the cursor snaps to the nearest voxel and the reported cursor coordinates are extent based. With continuous probing, voxel data is interpolated using vtkDataSetAttributes' InterpolatePoint method and the reported coordinates are 3D spatial continuous.

- `obj.SetInteraction (int interact)` - Enable/disable mouse interaction so the widget remains on display.

- `int = obj.GetInteraction ()` - Enable/disable mouse interaction so the widget remains on display.

- `obj.InteractionOn ()` - Enable/disable mouse interaction so the widget remains on display.

- `obj.InteractionOff ()` - Enable/disable mouse interaction so the widget remains on display.

- `obj.SetLeftButtonAction (int )` - Set action associated to buttons.

- `int = obj.GetLeftButtonActionMinValue ()` - Set action associated to buttons.

- `int = obj.GetLeftButtonActionMaxValue ()` - Set action associated to buttons.

- `int = obj.GetLeftButtonAction ()` - Set action associated to buttons.

- `obj.SetMiddleButtonAction (int )` - Set action associated to buttons.

- `int = obj.GetMiddleButtonActionMinValue ()` - Set action associated to buttons.

- `int = obj.GetMiddleButtonActionMaxValue ()` - Set action associated to buttons.

- `int = obj.GetMiddleButtonAction ()` - Set action associated to buttons.

- `obj.SetRightButtonAction (int )` - Set action associated to buttons.

- `int = obj.GetRightButtonActionMinValue ()` - Set action associated to buttons.

- `int = obj.GetRightButtonActionMaxValue ()` - Set action associated to buttons.

- `int = obj.GetRightButtonAction ()` - Set action associated to buttons.

- `obj.SetLeftButtonAutoModifier (int )` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `int = obj.GetLeftButtonAutoModifierMinValue ()` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `int = obj.GetLeftButtonAutoModifierMaxValue ()` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `int = obj.GetLeftButtonAutoModifier ()` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `obj.SetMiddleButtonAutoModifier (int )` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `int = obj.GetMiddleButtonAutoModifierMinValue ()` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `int = obj.GetMiddleButtonAutoModifierMaxValue ()` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `int = obj.GetMiddleButtonAutoModifier ()` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `obj.SetRightButtonAutoModifier (int )` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `int = obj.GetRightButtonAutoModifierMinValue ()` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `int = obj.GetRightButtonAutoModifierMaxValue ()` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION

(see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

- `int = obj.GetRightButtonAutoModifier ()` - Set the auto-modifiers associated to buttons. This allows users to bind some buttons to actions that are usually triggered by a key modifier. For example, if you do not need cursoring, you can bind the left button action to VTK_SLICE_MOTION_ACTION (see above) and the left button auto modifier to VTK_CONTROL_MODIFIER: you end up with the left button controling panning without pressing a key.

42.51 vtkImageTracerWidget

42.51.1 Usage

vtkImageTracerWidget is different from other widgets in three distinct ways: 1) any sub-class of vtkProp can be input rather than just vtkProp3D, so that vtkImageActor can be set as the prop and then traced over, 2) the widget fires pick events at the input prop to decide where to move its handles, 3) the widget has 2D glyphs for handles instead of 3D spheres as is done in other sub-classes of vtk3DWidget. This widget is primarily designed for manually tracing over image data. The button actions and key modifiers are as follows for controlling the widget: 1) left button click over the image, hold and drag draws a free hand line. 2) left button click and release erases the widget line, if it exists, and repositions the first handle. 3) middle button click starts a snap drawn line. The line is terminated by clicking the middle button while depressing the ctrl key. 4) when tracing a continuous or snap drawn line, if the last cursor position is within a specified tolerance to the first handle, the widget line will form a closed loop. 5) right button clicking and holding on any handle that is part of a snap drawn line allows handle dragging: existing line segments are updated accordingly. If the path is open and AutoClose is set to On, the path can be closed by repositioning the first and last points over one another. 6) ctrl key + right button down on any handle will erase it: existing snap drawn line segments are updated accordingly. If the line was formed by continous tracing, the line is deleted leaving one handle. 7) shift key + right button down on any snap drawn line segment will insert a handle at the cursor position. The line segment is split accordingly.

To create an instance of class vtkImageTracerWidget, simply invoke its constructor as follows

```
obj = vtkImageTracerWidget
```

42.51.2 Methods

The class vtkImageTracerWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImageTracerWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImageTracerWidget = obj.NewInstance ()`
- `vtkImageTracerWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double bounds[6])` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget ()` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Set/Get the handle properties (the 2D glyphs are the handles). The properties of the handles when selected and normal can be manipulated.

- `obj.SetHandleProperty (vtkProperty )` - Set/Get the handle properties (the 2D glyphs are the handles). The properties of the handles when selected and normal can be manipulated.
- `vtkProperty = obj.GetHandleProperty ()` - Set/Get the handle properties (the 2D glyphs are the handles). The properties of the handles when selected and normal can be manipulated.
- `obj.SetSelectedHandleProperty (vtkProperty )` - Set/Get the handle properties (the 2D glyphs are the handles). The properties of the handles when selected and normal can be manipulated.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Set/Get the handle properties (the 2D glyphs are the handles). The properties of the handles when selected and normal can be manipulated.
- `obj.SetLineProperty (vtkProperty )` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `vtkProperty = obj.GetLineProperty ()` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `obj.SetSelectedLineProperty (vtkProperty )` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedLineProperty ()` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `obj.SetViewProp (vtkProp prop)` - Set the prop, usually a vtkImageActor, to trace over.
- `obj.SetProjectToPlane (int )` - Force handles to be on a specific ortho plane. Default is Off.
- `int = obj.GetProjectToPlane ()` - Force handles to be on a specific ortho plane. Default is Off.
- `obj.ProjectToPlaneOn ()` - Force handles to be on a specific ortho plane. Default is Off.
- `obj.ProjectToPlaneOff ()` - Force handles to be on a specific ortho plane. Default is Off.
- `obj.SetProjectionNormal (int )` - Set the projection normal. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively. Since the handles are 2D glyphs, it is necessary to specify a plane on which to generate them, even though ProjectToPlane may be turned off.
- `int = obj.GetProjectionNormalMinValue ()` - Set the projection normal. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively. Since the handles are 2D glyphs, it is necessary to specify a plane on which to generate them, even though ProjectToPlane may be turned off.
- `int = obj.GetProjectionNormalMaxValue ()` - Set the projection normal. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively. Since the handles are 2D glyphs, it is necessary to specify a plane on which to generate them, even though ProjectToPlane may be turned off.
- `int = obj.GetProjectionNormal ()` - Set the projection normal. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively. Since the handles are 2D glyphs, it is necessary to specify a plane on which to generate them, even though ProjectToPlane may be turned off.
- `obj.SetProjectionNormalToXAxes ()` - Set the projection normal. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively. Since the handles are 2D glyphs, it is necessary to specify a plane on which to generate them, even though ProjectToPlane may be turned off.
- `obj.SetProjectionNormalToYAxes ()` - Set the projection normal. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively. Since the handles are 2D glyphs, it is necessary to specify a plane on which to generate them, even though ProjectToPlane may be turned off.
- `obj.SetProjectionNormalToZAxes ()` - Set the position of the widgets' handles in terms of a plane's position. e.g., if ProjectionNormal is 0, all of the x-coordinate values of the handles are set to ProjectionPosition. No attempt is made to ensure that the position is within the bounds of either the underlying image data or the prop on which tracing is performed.

- `obj.SetProjectionPosition (double position)` - Set the position of the widgets' handles in terms of a plane's position. e.g., if ProjectionNormal is 0, all of the x-coordinate values of the handles are set to ProjectionPosition. No attempt is made to ensure that the position is within the bounds of either the underlying image data or the prop on which tracing is performed.
- `double = obj.GetProjectionPosition ()` - Set the position of the widgets' handles in terms of a plane's position. e.g., if ProjectionNormal is 0, all of the x-coordinate values of the handles are set to ProjectionPosition. No attempt is made to ensure that the position is within the bounds of either the underlying image data or the prop on which tracing is performed.
- `obj.SetSnapToImage (int snap)` - Force snapping to image data while tracing. Default is Off.
- `int = obj.GetSnapToImage ()` - Force snapping to image data while tracing. Default is Off.
- `obj.SnapToImageOn ()` - Force snapping to image data while tracing. Default is Off.
- `obj.SnapToImageOff ()` - Force snapping to image data while tracing. Default is Off.
- `obj.SetAutoClose (int )` - In concert with a CaptureRadius value, automatically form a closed path by connecting first to last path points. Default is Off.
- `int = obj.GetAutoClose ()` - In concert with a CaptureRadius value, automatically form a closed path by connecting first to last path points. Default is Off.
- `obj.AutoCloseOn ()` - In concert with a CaptureRadius value, automatically form a closed path by connecting first to last path points. Default is Off.
- `obj.AutoCloseOff ()` - In concert with a CaptureRadius value, automatically form a closed path by connecting first to last path points. Default is Off.
- `obj.SetCaptureRadius (double )` - Set/Get the capture radius for automatic path closing. For image data, capture radius should be half the distance between voxel/pixel centers. Default is 1.0
- `double = obj.GetCaptureRadius ()` - Set/Get the capture radius for automatic path closing. For image data, capture radius should be half the distance between voxel/pixel centers. Default is 1.0
- `obj.GetPath (vtkPolyData pd)` - Grab the points and lines that define the traced path. These point values are guaranteed to be up-to-date when either the InteractionEvent or EndInteraction events are invoked. The user provides the vtkPolyData and the points and cells representing the line are added to it.
- `vtkGlyphSource2D = obj.GetGlyphSource ()` - Set/Get the type of snapping to image data: center of a pixel/voxel or nearest point defining a pixel/voxel.
- `obj.SetImageSnapType (int )` - Set/Get the type of snapping to image data: center of a pixel/voxel or nearest point defining a pixel/voxel.
- `int = obj.GetImageSnapTypeMinValue ()` - Set/Get the type of snapping to image data: center of a pixel/voxel or nearest point defining a pixel/voxel.
- `int = obj.GetImageSnapTypeMaxValue ()` - Set/Get the type of snapping to image data: center of a pixel/voxel or nearest point defining a pixel/voxel.
- `int = obj.GetImageSnapType ()` - Set/Get the type of snapping to image data: center of a pixel/voxel or nearest point defining a pixel/voxel.
- `obj.SetHandlePosition (int handle, double xyz[3])` - Set/Get the handle position in terms of a zero-based array of handles.
- `obj.SetHandlePosition (int handle, double x, double y, double z)` - Set/Get the handle position in terms of a zero-based array of handles.

- `obj.GetHandlePosition (int handle, double xyz[3])` - Set/Get the handle position in terms of a zero-based array of handles.
- `double = obj.GetHandlePosition (int handle)` - Set/Get the handle position in terms of a zero-based array of handles.
- `int = obj.GetNumberOfHandles ()` - Get the number of handles.
- `obj.SetInteraction (int interact)` - Enable/disable mouse interaction when the widget is visible.
- `int = obj.GetInteraction ()` - Enable/disable mouse interaction when the widget is visible.
- `obj.InteractionOn ()` - Enable/disable mouse interaction when the widget is visible.
- `obj.InteractionOff ()` - Enable/disable mouse interaction when the widget is visible.
- `obj.InitializeHandles (vtkPoints )` - Initialize the widget with a set of points and generate lines between them. If AutoClose is on it will handle the case wherein the first and last points are congruent.
- `int = obj.IsClosed ()` - Is the path closed or open?
- `obj.SetProp (vtkProp prop)` - @deprecated Replaced by vtkImageTracerWidget::SetViewProp() as of VTK 5.0.

42.52 vtkImplicitPlaneRepresentation

42.52.1 Usage

This class is a concrete representation for the vtkImplicitPlaneWidget2. It represents an infinite plane defined by a normal and point in the context of a bounding box. Through interaction with the widget, the plane can be manipulated by adjusting the plane normal or moving the origin point.

To use this representation, you normally define a (plane) origin and (plane) normal. The PlaceWidget() method is also used to intially position the representation.

To create an instance of class vtkImplicitPlaneRepresentation, simply invoke its constructor as follows

```
obj = vtkImplicitPlaneRepresentation
```

42.52.2 Methods

The class vtkImplicitPlaneRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitPlaneRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkImplicitPlaneRepresentation = obj.NewInstance ()` - Standard methods for the class.
- `vtkImplicitPlaneRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.
- `obj.SetOrigin (double x, double y, double z)` - Get the origin of the plane.
- `obj.SetOrigin (double x[3])` - Get the origin of the plane.
- `double = obj.GetOrigin ()` - Get the origin of the plane.
- `obj.GetOrigin (double xyz[3])` - Get the origin of the plane.

- `obj.SetNormal (double x, double y, double z)` - Get the normal to the plane.
- `obj.SetNormal (double x[3])` - Get the normal to the plane.
- `double = obj.GetNormal ()` - Get the normal to the plane.
- `obj.GetNormal (double xyz[3])` - Get the normal to the plane.
- `obj.SetNormalToXAxis (int )` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `int = obj.GetNormalToXAxis ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToXAxisOn ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToXAxisOff ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.SetNormalToYAxis (int )` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `int = obj.GetNormalToYAxis ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToYAxisOn ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToYAxisOff ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.SetNormalToZAxis (int )` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `int = obj.GetNormalToZAxis ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToZAxisOn ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToZAxisOff ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.SetTubing (int )` - Turn on/off tubing of the wire outline of the plane. The tube thickens the line by wrapping with a vtkTubeFilter.

- `int = obj.GetTubing ()` - Turn on/off tubing of the wire outline of the plane. The tube thickens the line by wrapping with a vtkTubeFilter.
- `obj.TubingOn ()` - Turn on/off tubing of the wire outline of the plane. The tube thickens the line by wrapping with a vtkTubeFilter.
- `obj.TubingOff ()` - Turn on/off tubing of the wire outline of the plane. The tube thickens the line by wrapping with a vtkTubeFilter.
- `obj.SetDrawPlane (int plane)` - Enable/disable the drawing of the plane. In some cases the plane interferes with the object that it is operating on (i.e., the plane interferes with the cut surface it produces producing z-buffer artifacts.)
- `int = obj.GetDrawPlane ()` - Enable/disable the drawing of the plane. In some cases the plane interferes with the object that it is operating on (i.e., the plane interferes with the cut surface it produces producing z-buffer artifacts.)
- `obj.DrawPlaneOn ()` - Enable/disable the drawing of the plane. In some cases the plane interferes with the object that it is operating on (i.e., the plane interferes with the cut surface it produces producing z-buffer artifacts.)
- `obj.DrawPlaneOff ()` - Enable/disable the drawing of the plane. In some cases the plane interferes with the object that it is operating on (i.e., the plane interferes with the cut surface it produces producing z-buffer artifacts.)
- `obj.SetOutlineTranslation (int )` - Turn on/off the ability to translate the bounding box by grabbing it with the left mouse button.
- `int = obj.GetOutlineTranslation ()` - Turn on/off the ability to translate the bounding box by grabbing it with the left mouse button.
- `obj.OutlineTranslationOn ()` - Turn on/off the ability to translate the bounding box by grabbing it with the left mouse button.
- `obj.OutlineTranslationOff ()` - Turn on/off the ability to translate the bounding box by grabbing it with the left mouse button.
- `obj.SetOutsideBounds (int )` - Turn on/off the ability to move the widget outside of the bounds specified in the initial PlaceWidget() invocation.
- `int = obj.GetOutsideBounds ()` - Turn on/off the ability to move the widget outside of the bounds specified in the initial PlaceWidget() invocation.
- `obj.OutsideBoundsOn ()` - Turn on/off the ability to move the widget outside of the bounds specified in the initial PlaceWidget() invocation.
- `obj.OutsideBoundsOff ()` - Turn on/off the ability to move the widget outside of the bounds specified in the initial PlaceWidget() invocation.
- `obj.SetScaleEnabled (int )` - Turn on/off the ability to scale the widget with the mouse.
- `int = obj.GetScaleEnabled ()` - Turn on/off the ability to scale the widget with the mouse.
- `obj.ScaleEnabledOn ()` - Turn on/off the ability to scale the widget with the mouse.
- `obj.ScaleEnabledOff ()` - Turn on/off the ability to scale the widget with the mouse.
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata that defines the plane. The polydata contains a single polygon that is clipped by the bounding box.
- `vtkPolyDataAlgorithm = obj.GetPolyDataAlgorithm ()` - Satisfies superclass API. This returns a pointer to the underlying PolyData (which represents the plane).

- `obj.GetPlane (vtkPlane plane)` - Get the implicit function for the plane. The user must provide the instance of the class vtkPlane. Note that vtkPlane is a subclass of vtkImplicitFunction, meaning that it can be used by a variety of filters to perform clipping, cutting, and selection of data.
- `obj.UpdatePlacement (void )` - Satisfies the superclass API. This will change the state of the widget to match changes that have been made to the underlying PolyDataSource
- `vtkProperty = obj.GetNormalProperty ()` - Get the properties on the normal (line and cone).
- `vtkProperty = obj.GetSelectedNormalProperty ()` - Get the properties on the normal (line and cone).
- `vtkProperty = obj.GetPlaneProperty ()` - Get the plane properties. The properties of the plane when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedPlaneProperty ()` - Get the plane properties. The properties of the plane when selected and unselected can be manipulated.
- `vtkProperty = obj.GetOutlineProperty ()` - Get the property of the outline.
- `vtkProperty = obj.GetSelectedOutlineProperty ()` - Get the property of the outline.
- `vtkProperty = obj.GetEdgesProperty ()` - Get the property of the intersection edges. (This property also applies to the edges when tubed.)
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Methods to interface with the vtkSliderWidget.
- `obj.PlaceWidget (double bounds[6])` - Methods to interface with the vtkSliderWidget.
- `obj.BuildRepresentation ()` - Methods to interface with the vtkSliderWidget.
- `obj.StartWidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget.
- `obj.WidgetInteraction (double newEventPos[2])` - Methods to interface with the vtkSliderWidget.
- `obj.EndWidgetInteraction (double newEventPos[2])` - Methods to interface with the vtkSliderWidget.
- `double = obj.GetBounds ()`
- `obj.GetActors (vtkPropCollection pc)`
- `obj.ReleaseGraphicsResources (vtkWindow )`
- `int = obj.RenderOpaqueGeometry (vtkViewport )`
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )`
- `int = obj.HasTranslucentPolygonalGeometry ()`
- `obj.SetInteractionState (int )` - The interaction state may be set from a widget (e.g., vtkImplicitPlaneWidget2) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking process with the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.

- `int = obj.GetInteractionStateMinValue ()` - The interaction state may be set from a widget (e.g., vtkImplicitPlaneWidget2) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking process with the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.

- `int = obj.GetInteractionStateMaxValue ()` - The interaction state may be set from a widget (e.g., vtkImplicitPlaneWidget2) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking process with the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.

- `obj.SetRepresentationState (int )` - Sets the visual appearance of the representation based on the state it is in. This state is usually the same as InteractionState.

- `int = obj.GetRepresentationState ()` - Sets the visual appearance of the representation based on the state it is in. This state is usually the same as InteractionState.

42.53 vtkImplicitPlaneWidget

42.53.1 Usage

This 3D widget defines an infinite plane that can be interactively placed in a scene. The widget is represented by a plane with a normal vector; the plane is contained by a bounding box, and where the plane intersects the bounding box the edges are shown (possibly tubed). The normal can be selected and moved to rotate the plane; the plane itself can be selected and translated in various directions. As the plane is moved, the implicit plane function and polygon (representing the plane cut against the bounding box) is updated.

To use this object, just invoke SetInteractor() with the argument of the method a vtkRenderWindowInteractor. You may also wish to invoke "PlaceWidget()" to initially position the widget. If the "i" key (for "interactor") is pressed, the vtkImplicitPlaneWidget will appear. (See superclass documentation for information about changing this behavior.) If you select the normal vector, the plane can be arbitrarily rotated. The plane can be translated along the normal by selecting the plane and moving it. The plane (the plane origin) can also be arbitrary moved by selecting the plane with the middle mouse button. The right mouse button can be used to uniformly scale the bounding box (moving "up" the box scales larger; moving "down" the box scales smaller). Events that occur outside of the widget (i.e., no part of the widget is picked) are propagated to any other registered obsevers (such as the interaction style). Turn off the widget by pressing the "i" key again (or invoke the Off() method).

The vtkImplicitPlaneWidget has several methods that can be used in conjunction with other VTK objects. The GetPolyData() method can be used to get a polygonal representation (the single polygon clipped by the bounding box). Typical usage of the widget is to make use of the StartInteractionEvent, InteractionEvent, and EndInteractionEvent events. The InteractionEvent is called on mouse motion; the other two events are called on button down and button up (either left or right button). (Note: there is also a PlaceWidgetEvent that is invoked when the widget is placed with PlaceWidget().)

Some additional features of this class include the ability to control the properties of the widget. You do this by setting property values on the normal vector (selected and unselected properties); the plane (selected and unselected properties); the outline (selected and unselected properties); and the edges. The edges may also be tubed or not.

To create an instance of class vtkImplicitPlaneWidget, simply invoke its constructor as follows

```
obj = vtkImplicitPlaneWidget
```

42.53.2 Methods

The class vtkImplicitPlaneWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely

intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitPlaneWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkImplicitPlaneWidget = obj.NewInstance ()`
- `vtkImplicitPlaneWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double bounds[6])` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget ()` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Get the origin of the plane.
- `obj.SetOrigin (double x, double y, double z)` - Get the origin of the plane.
- `obj.SetOrigin (double x[3])` - Get the origin of the plane.
- `double = obj.GetOrigin ()` - Get the origin of the plane.
- `obj.GetOrigin (double xyz[3])` - Get the origin of the plane.
- `obj.SetNormal (double x, double y, double z)` - Get the normal to the plane.
- `obj.SetNormal (double x[3])` - Get the normal to the plane.
- `double = obj.GetNormal ()` - Get the normal to the plane.
- `obj.GetNormal (double xyz[3])` - Get the normal to the plane.
- `obj.SetNormalToXAxis (int )` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `int = obj.GetNormalToXAxis ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToXAxisOn ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToXAxisOff ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.SetNormalToYAxis (int )` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `int = obj.GetNormalToYAxis ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToYAxisOn ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.

- `obj.NormalToYAxisOff ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.SetNormalToZAxis (int )` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `int = obj.GetNormalToZAxis ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToZAxisOn ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToZAxisOff ()` - Force the plane widget to be aligned with one of the x-y-z axes. If one axis is set on, the other two will be set off. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.SetTubing (int )` - Turn on/off tubing of the wire outline of the plane. The tube thickens the line by wrapping with a vtkTubeFilter.
- `int = obj.GetTubing ()` - Turn on/off tubing of the wire outline of the plane. The tube thickens the line by wrapping with a vtkTubeFilter.
- `obj.TubingOn ()` - Turn on/off tubing of the wire outline of the plane. The tube thickens the line by wrapping with a vtkTubeFilter.
- `obj.TubingOff ()` - Turn on/off tubing of the wire outline of the plane. The tube thickens the line by wrapping with a vtkTubeFilter.
- `obj.SetDrawPlane (int plane)` - Enable/disable the drawing of the plane. In some cases the plane interferes with the object that it is operating on (i.e., the plane interferes with the cut surface it produces producing z-buffer artifacts.)
- `int = obj.GetDrawPlane ()` - Enable/disable the drawing of the plane. In some cases the plane interferes with the object that it is operating on (i.e., the plane interferes with the cut surface it produces producing z-buffer artifacts.)
- `obj.DrawPlaneOn ()` - Enable/disable the drawing of the plane. In some cases the plane interferes with the object that it is operating on (i.e., the plane interferes with the cut surface it produces producing z-buffer artifacts.)
- `obj.DrawPlaneOff ()` - Enable/disable the drawing of the plane. In some cases the plane interferes with the object that it is operating on (i.e., the plane interferes with the cut surface it produces producing z-buffer artifacts.)
- `obj.SetOutlineTranslation (int )` - Turn on/off the ability to translate the bounding box by grabbing it with the left mouse button.
- `int = obj.GetOutlineTranslation ()` - Turn on/off the ability to translate the bounding box by grabbing it with the left mouse button.
- `obj.OutlineTranslationOn ()` - Turn on/off the ability to translate the bounding box by grabbing it with the left mouse button.
- `obj.OutlineTranslationOff ()` - Turn on/off the ability to translate the bounding box by grabbing it with the left mouse button.

- `obj.SetOutsideBounds (int )` - Turn on/off the ability to move the widget outside of the input's bound
- `int = obj.GetOutsideBounds ()` - Turn on/off the ability to move the widget outside of the input's bound
- `obj.OutsideBoundsOn ()` - Turn on/off the ability to move the widget outside of the input's bound
- `obj.OutsideBoundsOff ()` - Turn on/off the ability to move the widget outside of the input's bound
- `obj.SetScaleEnabled (int )` - Turn on/off the ability to scale with the mouse
- `int = obj.GetScaleEnabled ()` - Turn on/off the ability to scale with the mouse
- `obj.ScaleEnabledOn ()` - Turn on/off the ability to scale with the mouse
- `obj.ScaleEnabledOff ()` - Turn on/off the ability to scale with the mouse
- `obj.SetOriginTranslation (int )` - Turn on/off the ability to translate the origin (sphere) with the left mouse button.
- `int = obj.GetOriginTranslation ()` - Turn on/off the ability to translate the origin (sphere) with the left mouse button.
- `obj.OriginTranslationOn ()` - Turn on/off the ability to translate the origin (sphere) with the left mouse button.
- `obj.OriginTranslationOff ()` - Turn on/off the ability to translate the origin (sphere) with the left mouse button.
- `obj.SetDiagonalRatio (double )` - By default the arrow is 30this ratio in the interval [0-2]
- `double = obj.GetDiagonalRatioMinValue ()` - By default the arrow is 30this ratio in the interval [0-2]
- `double = obj.GetDiagonalRatioMaxValue ()` - By default the arrow is 30this ratio in the interval [0-2]
- `double = obj.GetDiagonalRatio ()` - By default the arrow is 30this ratio in the interval [0-2]
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata that defines the plane. The polydata contains a single polygon that is clipped by the bounding box.
- `vtkPolyDataAlgorithm = obj.GetPolyDataAlgorithm ()` - Satisfies superclass API. This returns a pointer to the underlying PolyData (which represents the plane).
- `obj.GetPlane (vtkPlane plane)` - Get the implicit function for the plane. The user must provide the instance of the class vtkPlane. Note that vtkPlane is a subclass of vtkImplicitFunction, meaning that it can be used by a variety of filters to perform clipping, cutting, and selection of data.
- `obj.UpdatePlacement ()` - Satisfies the superclass API. This will change the state of the widget to match changes that have been made to the underlying PolyDataSource
- `obj.SizeHandles ()` - Control widget appearance
- `vtkProperty = obj.GetNormalProperty ()` - Get the properties on the normal (line and cone).
- `vtkProperty = obj.GetSelectedNormalProperty ()` - Get the properties on the normal (line and cone).
- `vtkProperty = obj.GetPlaneProperty ()` - Get the plane properties. The properties of the plane when selected and unselected can be manipulated.

- `vtkProperty = obj.GetSelectedPlaneProperty ()` - Get the plane properties. The properties of the plane when selected and unselected can be manipulated.
- `vtkProperty = obj.GetOutlineProperty ()` - Get the property of the outline.
- `vtkProperty = obj.GetSelectedOutlineProperty ()` - Get the property of the outline.
- `vtkProperty = obj.GetEdgesProperty ()` - Get the property of the intersection edges. (This property also applies to the edges when tubed.)

42.54 vtkImplicitPlaneWidget2

42.54.1 Usage

This 3D widget defines an infinite plane that can be interactively placed in a scene. The widget is assumed to consist of four parts: 1) a plane contained in a 2) bounding box, with a 3) plane normal, which is rooted at a 4) point on the plane. (The representation paired with this widget determines the actual geometry of the widget.)

To use this widget, you generally pair it with a vtkImplicitPlaneRepresentation (or a subclass). Variuos options are available for controlling how the representation appears, and how the widget functions.

To create an instance of class vtkImplicitPlaneWidget2, simply invoke its constructor as follows

```
obj = vtkImplicitPlaneWidget2
```

42.54.2 Methods

The class vtkImplicitPlaneWidget2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkImplicitPlaneWidget2 class.

- `string = obj.GetClassName ()` - Standard vtkObject methods
- `int = obj.IsA (string name)` - Standard vtkObject methods
- `vtkImplicitPlaneWidget2 = obj.NewInstance ()` - Standard vtkObject methods
- `vtkImplicitPlaneWidget2 = obj.SafeDownCast (vtkObject o)` - Standard vtkObject methods
- `obj.SetRepresentation (vtkImplicitPlaneRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.55 vtkLinearContourLineInterpolator

42.55.1 Usage

The line interpolator interpolates supplied nodes (see InterpolateLine) with line segments. The finess of the curve may be controlled using SetMaximumCurveError and SetMaximumNumberOfLineSegments.

To create an instance of class vtkLinearContourLineInterpolator, simply invoke its constructor as follows

```
obj = vtkLinearContourLineInterpolator
```

42.55.2 Methods

The class vtkLinearContourLineInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLinearContourLineInterpolator class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkLinearContourLineInterpolator = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkLinearContourLineInterpolator = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `int = obj.InterpolateLine (vtkRenderer ren, vtkContourRepresentation rep, int idx1, int idx2)`

42.56 vtkLineRepresentation

42.56.1 Usage

This class is a concrete representation for the vtkLineWidget2. It represents a straight line with three handles: one at the beginning and ending of the line, and one used to translate the line. Through interaction with the widget, the line representation can be arbitrarily placed in the 3D space.

To use this representation, you normally specify the position of the two end points (either in world or display coordinates). The PlaceWidget() method is also used to intially position the representation.

To create an instance of class vtkLineRepresentation, simply invoke its constructor as follows

```
obj = vtkLineRepresentation
```

42.56.2 Methods

The class vtkLineRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLineRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkLineRepresentation = obj.NewInstance ()` - Standard methods for the class.
- `vtkLineRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.
- `obj.GetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `double = obj.GetPoint1WorldPosition ()` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.
- `obj.GetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `double = obj.GetPoint1DisplayPosition ()` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetPoint1WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetPoint1DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.GetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `double = obj.GetPoint2DisplayPosition ()` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.GetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `double = obj.GetPoint2WorldPosition ()` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetPoint2WorldPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetPoint2DisplayPosition (double pos[3])` - Methods to Set/Get the coordinates of the two points defining this representation. Note that methods are available for both display and world coordinates.

- `obj.SetHandleRepresentation (vtkPointHandleRepresentation3D handle)` - This method is used to specify the type of handle representation to use for the three internal vtkHandleWidgets within vtkLineWidget2. To use this method, create a dummy vtkHandleWidget (or subclass), and then invoke this method with this dummy. Then the vtkLineRepresentation uses this dummy to clone three vtkHandleWidgets of the same type. Make sure you set the handle representation before the widget is enabled. (The method InstantiateHandleRepresentation() is invoked by the vtkLineWidget2.)

- `obj.InstantiateHandleRepresentation ()` - This method is used to specify the type of handle representation to use for the three internal vtkHandleWidgets within vtkLineWidget2. To use this method, create a dummy vtkHandleWidget (or subclass), and then invoke this method with this dummy. Then the vtkLineRepresentation uses this dummy to clone three vtkHandleWidgets of the same type. Make sure you set the handle representation before the widget is enabled. (The method InstantiateHandleRepresentation() is invoked by the vtkLineWidget2.)

- `vtkPointHandleRepresentation3D = obj.GetPoint1Representation ()` - Get the three handle representations used for the vtkLineWidget2.

- `vtkPointHandleRepresentation3D = obj.GetPoint2Representation ()` - Get the three handle representations used for the vtkLineWidget2.

- `vtkPointHandleRepresentation3D = obj.GetLineHandleRepresentation ()` - Get the three handle representations used for the vtkLineWidget2.

- `vtkProperty = obj.GetEndPointProperty ()` - Get the end-point (sphere) properties. The properties of the end-points when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedEndPointProperty ()` - Get the end-point (sphere) properties. The properties of the end-points when selected and unselected can be manipulated.
- `vtkProperty = obj.GetEndPoint2Property ()` - Get the end-point (sphere) properties. The properties of the end-points when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedEndPoint2Property ()` - Get the end-point (sphere) properties. The properties of the end-points when selected and unselected can be manipulated.
- `vtkProperty = obj.GetLineProperty ()` - Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedLineProperty ()` - Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `obj.SetTolerance (int )` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the line or end point to be active.
- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the line or end point to be active.
- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the line or end point to be active.
- `int = obj.GetTolerance ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the line or end point to be active.
- `obj.SetResolution (int res)` - Set/Get the resolution (number of subdivisions) of the line. A line with resolution greater than one is useful when points along the line are desired; e.g., generating a rake of streamlines.
- `int = obj.GetResolution ()` - Set/Get the resolution (number of subdivisions) of the line. A line with resolution greater than one is useful when points along the line are desired; e.g., generating a rake of streamlines.
- `obj.GetPolyData (vtkPolyData pd)` - Retrieve the polydata (including points) that defines the line. The polydata consists of n+1 points, where n is the resolution of the line. These point values are guaranteed to be up-to-date whenever any one of the three handles are moved. To use this method, the user provides the vtkPolyData as an input argument, and the points and polyline are copied into it.
- `obj.PlaceWidget (double bounds[6])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.StartWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API.
- `double = obj.GetBounds ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `obj.GetActors (vtkPropCollection pc)` - Methods supporting the rendering process.

- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods supporting the rendering process.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Methods supporting the rendering process.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Methods supporting the rendering process.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods supporting the rendering process.
- `obj.SetInteractionState (int )` - The interaction state may be set from a widget (e.g., vtkLineWidget2) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking process with the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.
- `int = obj.GetInteractionStateMinValue ()` - The interaction state may be set from a widget (e.g., vtkLineWidget2) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking process with the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.
- `int = obj.GetInteractionStateMaxValue ()` - The interaction state may be set from a widget (e.g., vtkLineWidget2) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking process with the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.
- `obj.SetRepresentationState (int )` - Sets the visual appearance of the representation based on the state it is in. This state is usually the same as InteractionState.
- `int = obj.GetRepresentationState ()` - Sets the visual appearance of the representation based on the state it is in. This state is usually the same as InteractionState.
- `long = obj.GetMTime ()` - Overload the superclasses' GetMTime() because internal classes are used to keep the state of the representation.
- `obj.SetRenderer (vtkRenderer ren)` - Overridden to set the rendererer on the internal representations.
- `obj.SetDistanceAnnotationVisibility (int )` - Show the distance between the points
- `int = obj.GetDistanceAnnotationVisibility ()` - Show the distance between the points
- `obj.DistanceAnnotationVisibilityOn ()` - Show the distance between the points
- `obj.DistanceAnnotationVisibilityOff ()` - Show the distance between the points
- `obj.SetDistanceAnnotationFormat (string )` - Specify the format to use for labelling the angle. Note that an empty string results in no label, or a format string without a "will not print the angle value.
- `string = obj.GetDistanceAnnotationFormat ()` - Specify the format to use for labelling the angle. Note that an empty string results in no label, or a format string without a "will not print the angle value.
- `obj.SetDistanceAnnotationScale (double scale[3])` - Scale text (font size along each dimension).
- `double = obj.GetDistance ()` - Get the distance between the points.
- `obj.SetLineColor (double r, double g, double b)` - Convenience method to set the line color. Ideally one should use GetLineProperty()-¿SetColor().

- `vtkProperty = obj.GetDistanceAnnotationProperty ()` - Get the distance annotation property
- `vtkFollower = obj.GetTextActor ()` - Get the text actor

42.57 vtkLineWidget

42.57.1 Usage

This 3D widget defines a line that can be interactively placed in a scene. The line has two handles (at its endpoints), plus the line can be picked to translate it in the scene. A nice feature of the object is that the vtkLineWidget, like any 3D widget, will work with the current interactor style and any other widgets present in the scene. That is, if vtkLineWidget does not handle an event, then all other registered observers (including the interactor style) have an opportunity to process the event. Otherwise, the vtkLineWidget will terminate the processing of the event that it handles.

To use this object, just invoke SetInteractor() with the argument of the method a vtkRenderWindowInteractor. You may also wish to invoke "PlaceWidget()" to initially position the widget. The interactor will act normally until the "i" key (for "interactor") is pressed, at which point the vtkLineWidget will appear. (See superclass documentation for information about changing this behavior.) By grabbing one of the two end point handles (use the left mouse button), the line can be oriented and stretched (the other end point remains fixed). By grabbing the line itself, or using the middle mouse button, the entire line can be translated. Scaling (about the center of the line) is achieved by using the right mouse button. By moving the mouse "up" the render window the line will be made bigger; by moving "down" the render window the widget will be made smaller. Turn off the widget by pressing the "i" key again (or invoke the Off() method). (Note: picking the line or either one of the two end point handles causes a vtkPointWidget to appear. This widget has the ability to constrain motion to an axis by pressing the "shift" key while moving the mouse.)

The vtkLineWidget has several methods that can be used in conjunction with other VTK objects. The Set/GetResolution() methods control the number of subdivisions of the line; the GetPolyData() method can be used to get the polygonal representation and can be used for things like seeding streamlines. Typical usage of the widget is to make use of the StartInteractionEvent, InteractionEvent, and EndInteractionEvent events. The InteractionEvent is called on mouse motion; the other two events are called on button down and button up (either left or right button).

Some additional features of this class include the ability to control the properties of the widget. You can set the properties of the selected and unselected representations of the line. For example, you can set the property for the handles and line. In addition there are methods to constrain the line so that it is aligned along the x-y-z axes.

To create an instance of class vtkLineWidget, simply invoke its constructor as follows

```
obj = vtkLineWidget
```

42.57.2 Methods

The class vtkLineWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLineWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkLineWidget = obj.NewInstance ()`
- `vtkLineWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double bounds[6])` - Methods that satisfy the superclass' API.

- `obj.PlaceWidget ()` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Set/Get the resolution (number of subdivisions) of the line.
- `obj.SetResolution (int r)` - Set/Get the resolution (number of subdivisions) of the line.
- `int = obj.GetResolution ()` - Set/Get the position of first end point.
- `obj.SetPoint1 (double x, double y, double z)` - Set/Get the position of first end point.
- `obj.SetPoint1 (double x[3])` - Set/Get the position of first end point.
- `double = obj.GetPoint1 ()` - Set/Get the position of first end point.
- `obj.GetPoint1 (double xyz[3])` - Set position of other end point.
- `obj.SetPoint2 (double x, double y, double z)` - Set position of other end point.
- `obj.SetPoint2 (double x[3])` - Set position of other end point.
- `double = obj.GetPoint2 ()` - Set position of other end point.
- `obj.GetPoint2 (double xyz[3])` - Force the line widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the line to the axes if it is orginally not aligned.
- `obj.SetAlign (int )` - Force the line widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the line to the axes if it is orginally not aligned.
- `int = obj.GetAlignMinValue ()` - Force the line widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the line to the axes if it is orginally not aligned.
- `int = obj.GetAlignMaxValue ()` - Force the line widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the line to the axes if it is orginally not aligned.
- `int = obj.GetAlign ()` - Force the line widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the line to the axes if it is orginally not aligned.
- `obj.SetAlignToXAxis ()` - Force the line widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the line to the axes if it is orginally not aligned.
- `obj.SetAlignToYAxis ()` - Force the line widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the line to the axes if it is orginally not aligned.
- `obj.SetAlignToZAxis ()` - Force the line widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the line to the axes if it is orginally not aligned.
- `obj.SetAlignToNone ()` - Enable/disable clamping of the point end points to the bounding box of the data. The bounding box is defined from the last PlaceWidget() invocation, and includes the effect of the PlaceFactor which is used to gram/shrink the bounding box.
- `obj.SetClampToBounds (int )` - Enable/disable clamping of the point end points to the bounding box of the data. The bounding box is defined from the last PlaceWidget() invocation, and includes the effect of the PlaceFactor which is used to gram/shrink the bounding box.

- `int = obj.GetClampToBounds ()` - Enable/disable clamping of the point end points to the bounding box of the data. The bounding box is defined from the last PlaceWidget() invocation, and includes the effect of the PlaceFactor which is used to gram/shrink the bounding box.
- `obj.ClampToBoundsOn ()` - Enable/disable clamping of the point end points to the bounding box of the data. The bounding box is defined from the last PlaceWidget() invocation, and includes the effect of the PlaceFactor which is used to gram/shrink the bounding box.
- `obj.ClampToBoundsOff ()` - Enable/disable clamping of the point end points to the bounding box of the data. The bounding box is defined from the last PlaceWidget() invocation, and includes the effect of the PlaceFactor which is used to gram/shrink the bounding box.
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that defines the line. The polydata consists of n+1 points, where n is the resolution of the line. These point values are guaranteed to be up-to-date when either the InteractionEvent or EndInteraction events are invoked. The user provides the vtkPolyData and the points and polyline are added to it.
- `vtkProperty = obj.GetHandleProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles when selected and normal can be manipulated.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles when selected and normal can be manipulated.
- `vtkProperty = obj.GetLineProperty ()` - Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedLineProperty ()` - Get the line properties. The properties of the line when selected and unselected can be manipulated.

42.58 vtkLineWidget2

42.58.1 Usage

This 3D widget defines a straight line that can be interactively placed in a scene. The widget is assumed to consist of two parts: 1) two end points and 2) a straight line connecting the two points. (The representation paired with this widget determines the actual geometry of the widget.) The positioning of the two end points is facilitated by using vtkHandleWidgets to position the points.

To use this widget, you generally pair it with a vtkLineRepresentation (or a subclass). Variuos options are available in the representation for controlling how the widget appears, and how the widget functions.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
 If one of the two end points are selected:
   LeftButtonPressEvent - activate the associated handle widget
   LeftButtonReleaseEvent - release the handle widget associated with the point
   MouseMoveEvent - move the point
 If the line is selected:
   LeftButtonPressEvent - activate a handle widget accociated with the line
   LeftButtonReleaseEvent - release the handle widget associated with the line
   MouseMoveEvent - translate the line
 In all the cases, independent of what is picked, the widget responds to the
 following VTK events:
   MiddleButtonPressEvent - translate the widget
   MiddleButtonReleaseEvent - release the widget
   RightButtonPressEvent - scale the widget's representation
   RightButtonReleaseEvent - stop scaling the widget
   MouseMoveEvent - scale (if right button) or move (if middle button) the widget
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkLineWidget2's widget events:

```
vtkWidgetEvent::Select -- some part of the widget has been selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Move -- a request for slider motion has been invoked
```

In turn, when these widget events are processed, the vtkLineWidget2 invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkLineWidget2, simply invoke its constructor as follows

```
obj = vtkLineWidget2
```

42.58.2 Methods

The class vtkLineWidget2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLineWidget2 class.

- `string = obj.GetClassName ()` - Standard vtkObject methods
- `int = obj.IsA (string name)` - Standard vtkObject methods
- `vtkLineWidget2 = obj.NewInstance ()` - Standard vtkObject methods
- `vtkLineWidget2 = obj.SafeDownCast (vtkObject o)` - Standard vtkObject methods
- `obj.SetEnabled (int enabling)` - Override superclasses' SetEnabled() method because the line widget must enable its internal handle widgets.
- `obj.SetRepresentation (vtkLineRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `obj.SetProcessEvents (int )` - Methods to change the whether the widget responds to interaction. Overridden to pass the state to component widgets.

42.59 vtkLogoRepresentation

42.59.1 Usage

To create an instance of class vtkLogoRepresentation, simply invoke its constructor as follows

```
obj = vtkLogoRepresentation
```

42.59.2 Methods

The class vtkLogoRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLogoRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK class methods.
- `int = obj.IsA (string name)` - Standard VTK class methods.
- `vtkLogoRepresentation = obj.NewInstance ()` - Standard VTK class methods.
- `vtkLogoRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK class methods.
- `obj.SetImage (vtkImageData img)` - Specify/retrieve the image to display in the balloon.
- `vtkImageData = obj.GetImage ()` - Specify/retrieve the image to display in the balloon.
- `obj.SetImageProperty (vtkProperty2D p)` - Set/get the image property (relevant only if an image is shown).
- `vtkProperty2D = obj.GetImageProperty ()` - Set/get the image property (relevant only if an image is shown).
- `obj.BuildRepresentation ()` - Satisfy the superclasses' API.
- `obj.GetActors2D (vtkPropCollection pc)` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.

42.60 vtkLogoWidget

42.60.1 Usage

This class provides support for interactively displaying and manipulating a logo. Logos are defined by an image; this widget simply allows you to interactively place and resize the image logo. To use this widget, simply create a vtkLogoRepresentation (or subclass) and associate it with the vtkLogoWidget.

To create an instance of class vtkLogoWidget, simply invoke its constructor as follows

```
obj = vtkLogoWidget
```

42.60.2 Methods

The class vtkLogoWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkLogoWidget class.

- `string = obj.GetClassName ()` - Standar VTK class methods.
- `int = obj.IsA (string name)` - Standar VTK class methods.
- `vtkLogoWidget = obj.NewInstance ()` - Standar VTK class methods.

- `vtkLogoWidget = obj.SafeDownCast (vtkObject o)` - Standar VTK class methods.
- `obj.SetRepresentation (vtkLogoRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.61 vtkOrientationMarkerWidget

42.61.1 Usage

This class provides support for interactively manipulating the position, size, and apparent orientation of a prop that represents an orientation marker. This class works by adding its internal renderer to an external "parent" renderer on a different layer. The input orientation marker is rendered as an overlay on the parent renderer and, thus, appears superposed over all props in the parent's scene. The camera view of the orientation the marker is made to match that of the parent's by means of an observer mechanism, giving the illusion that the orientation of the marker reflects that of the prop(s) in the parent's scene.

The widget listens to left mouse button and mouse movement events. It will change the cursor shape based on its location. If the cursor is over the overlay renderer, it will change the cursor shape to a SIZEALL shape or to a resize corner shape (e.g., SIZENW) if the cursor is near a corner. If the left mouse button is pressed and held down while moving, the overlay renderer, and hence, the orientation marker, is resized or moved. I the case of a resize operation, releasing the left mouse button causes the widget to enforce its renderer to be square. The diagonally opposite corner to the one moved is repositioned such that all edges of the renderer have the same length: the minimum.

To use this object, there are two key steps: 1) invoke SetInteractor() with the argument of the method a vtkRenderWindowInteractor, and 2) invoke SetOrientationMarker with an instance of vtkProp (see caveats below). Specifically, vtkAxesActor and vtkAnnotatedCubeActor are two classes designed to work with this class. A composite orientation marker can be generated by adding instances of vtkAxesActor and vtkAnnotatedCubeActor to a vtkPropAssembly, which can then be set as the input orientation marker. The widget can be also be set up in a non-interactive fashion by setting Ineractive to Off and sizing/placing the overlay renderer in its parent renderer by calling the widget's SetViewport method.

To create an instance of class vtkOrientationMarkerWidget, simply invoke its constructor as follows

```
obj = vtkOrientationMarkerWidget
```

42.61.2 Methods

The class vtkOrientationMarkerWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOrientationMarkerWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkOrientationMarkerWidget = obj.NewInstance ()`
- `vtkOrientationMarkerWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetOrientationMarker (vtkProp prop)` - Set/get the orientation marker to be displayed in this widget.
- `vtkProp = obj.GetOrientationMarker ()` - Set/get the orientation marker to be displayed in this widget.
- `obj.SetEnabled (int )` - Enable/disable the widget. Default is 0 (disabled).

- `obj.SetInteractive (int state)` - Set/get whether to allow this widget to be interactively moved/scaled. Default is On.
- `int = obj.GetInteractive ()` - Set/get whether to allow this widget to be interactively moved/scaled. Default is On.
- `obj.InteractiveOn ()` - Set/get whether to allow this widget to be interactively moved/scaled. Default is On.
- `obj.InteractiveOff ()` - Set/get whether to allow this widget to be interactively moved/scaled. Default is On.
- `obj.SetOutlineColor (double r, double g, double b)` - Set/get the color of the outline of this widget. The outline is visible when (in interactive mode) the cursor is over this widget. Default is white (1,1,1).
- `obj.SetViewport (double minX, double minY, double maxX, double maxY)` - Set/get the viewport to position/size this widget. Default is bottom left corner (0,0,0.2,0.2).
- `obj.SetTolerance (int )` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).
- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).
- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).
- `int = obj.GetTolerance ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).

42.62 vtkOrientedGlyphContourRepresentation

42.62.1 Usage

This class provides the default concrete representation for the vtkContourWidget. It works in conjunction with the vtkContourLineInterpolator and vtkPointPlacer. See vtkContourWidget for details.

To create an instance of class vtkOrientedGlyphContourRepresentation, simply invoke its constructor as follows

```
obj = vtkOrientedGlyphContourRepresentation
```

42.62.2 Methods

The class vtkOrientedGlyphContourRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOrientedGlyphContourRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkOrientedGlyphContourRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.

- `vtkOrientedGlyphContourRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetCursorShape (vtkPolyData cursorShape)` - Specify the cursor shape. Keep in mind that the shape will be aligned with the constraining plane by orienting it such that the x axis of the geometry lies along the normal of the plane.
- `vtkPolyData = obj.GetCursorShape ()` - Specify the cursor shape. Keep in mind that the shape will be aligned with the constraining plane by orienting it such that the x axis of the geometry lies along the normal of the plane.
- `obj.SetActiveCursorShape (vtkPolyData activeShape)` - Specify the shape of the cursor (handle) when it is active. This is the geometry that will be used when the mouse is close to the handle or if the user is manipulating the handle.
- `vtkPolyData = obj.GetActiveCursorShape ()` - Specify the shape of the cursor (handle) when it is active. This is the geometry that will be used when the mouse is close to the handle or if the user is manipulating the handle.
- `vtkProperty = obj.GetProperty ()` - This is the property used when the handle is not active (the mouse is not near the handle)
- `vtkProperty = obj.GetActiveProperty ()` - This is the property used when the user is interacting with the handle.
- `vtkProperty = obj.GetLinesProperty ()` - This is the property used by the lines.
- `obj.SetRenderer (vtkRenderer ren)` - Subclasses of vtkOrientedGlyphContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.BuildRepresentation ()` - Subclasses of vtkOrientedGlyphContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.StartWidgetInteraction (double eventPos[2])` - Subclasses of vtkOrientedGlyphContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.WidgetInteraction (double eventPos[2])` - Subclasses of vtkOrientedGlyphContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `int = obj.ComputeInteractionState (int X, int Y, int modified)` - Subclasses of vtkOrientedGlyphContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.GetActors (vtkPropCollection )` - Methods to make this class behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods to make this class behave as a vtkProp.

- `vtkPolyData = obj.GetContourRepresentationAsPolyData ()` - Get the points in this contour as a vtkPolyData.
- `obj.SetAlwaysOnTop (int )` - Controls whether the contour widget should always appear on top of other actors in the scene. (In effect, this will disable OpenGL Depth buffer tests while rendering the contour). Default is to set it to false.
- `int = obj.GetAlwaysOnTop ()` - Controls whether the contour widget should always appear on top of other actors in the scene. (In effect, this will disable OpenGL Depth buffer tests while rendering the contour). Default is to set it to false.
- `obj.AlwaysOnTopOn ()` - Controls whether the contour widget should always appear on top of other actors in the scene. (In effect, this will disable OpenGL Depth buffer tests while rendering the contour). Default is to set it to false.
- `obj.AlwaysOnTopOff ()` - Controls whether the contour widget should always appear on top of other actors in the scene. (In effect, this will disable OpenGL Depth buffer tests while rendering the contour). Default is to set it to false.
- `obj.SetLineColor (double r, double g, double b)` - Convenience method to set the line color. Ideally one should use GetLinesProperty()-¿SetColor().
- `obj.SetShowSelectedNodes (int )` - A flag to indicate whether to show the Selected nodes Default is to set it to false.

42.63 vtkOrientedGlyphFocalPlaneContourRepresentation

42.63.1 Usage

This class is used to represent a contour drawn on the focal plane (usually overlayed on top of an image or volume widget). The class was written in order to be able to draw contours on a volume widget and have the contours overlayed on the focal plane in order to do contour segmentation.

To create an instance of class vtkOrientedGlyphFocalPlaneContourRepresentation, simply invoke its constructor as follows

```
obj = vtkOrientedGlyphFocalPlaneContourRepresentation
```

42.63.2 Methods

The class vtkOrientedGlyphFocalPlaneContourRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOrientedGlyphFocalPlaneContourRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkOrientedGlyphFocalPlaneContourRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkOrientedGlyphFocalPlaneContourRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetCursorShape (vtkPolyData cursorShape)` - Specify the cursor shape. Keep in mind that the shape will be aligned with the constraining plane by orienting it such that the x axis of the geometry lies along the normal of the plane.

- `vtkPolyData = obj.GetCursorShape ()` - Specify the cursor shape. Keep in mind that the shape will be aligned with the constraining plane by orienting it such that the x axis of the geometry lies along the normal of the plane.
- `obj.SetActiveCursorShape (vtkPolyData activeShape)` - Specify the shape of the cursor (handle) when it is active. This is the geometry that will be used when the mouse is close to the handle or if the user is manipulating the handle.
- `vtkPolyData = obj.GetActiveCursorShape ()` - Specify the shape of the cursor (handle) when it is active. This is the geometry that will be used when the mouse is close to the handle or if the user is manipulating the handle.
- `vtkProperty2D = obj.GetProperty ()` - This is the property used when the handle is not active (the mouse is not near the handle)
- `vtkProperty2D = obj.GetActiveProperty ()` - This is the property used when the user is interacting with the handle.
- `vtkProperty2D = obj.GetLinesProperty ()` - This is the property used by the lines.
- `obj.SetRenderer (vtkRenderer ren)` - Subclasses of vtkOrientedGlyphFocalPlaneContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.BuildRepresentation ()` - Subclasses of vtkOrientedGlyphFocalPlaneContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.StartWidgetInteraction (double eventPos[2])` - Subclasses of vtkOrientedGlyphFocalPlaneContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.WidgetInteraction (double eventPos[2])` - Subclasses of vtkOrientedGlyphFocalPlaneContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `int = obj.ComputeInteractionState (int X, int Y, int modified)` - Subclasses of vtkOrientedGlyphFocalPlaneContourRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.GetActors2D (vtkPropCollection )` - Methods to make this class behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods to make this class behave as a vtkProp.
- `vtkPolyData = obj.GetContourRepresentationAsPolyData ()` - Get the points in this contour as a vtkPolyData.
- `vtkMatrix4x4 = obj.GetContourPlaneDirectionCosines (double origin[3])` - Direction cosines of the plane on which the contour lies on in world co-ordinates. This would be the same matrix that would be set in vtkImageReslice or vtkImagePlaneWidget if there were a plane passing through the contour points. The origin must be the origin of the data under the contour.

42.64 vtkOrientedPolygonalHandleRepresentation3D

42.64.1 Usage

This class serves as the geometrical representation of a vtkHandleWidget. The handle can be represented by an arbitrary polygonal data (vtkPolyData), set via SetHandle(vtkPolyData *). The actual position of the handle will be initially assumed to be (0,0,0). You can specify an offset from this position if desired. This class differs from vtkPolygonalHandleRepresentation3D in that the handle will always remain front facing, ie it maintains a fixed orientation with respect to the camera. This is done by using vtkFollowers internally to render the actors.

To create an instance of class vtkOrientedPolygonalHandleRepresentation3D, simply invoke its constructor as follows

```
obj = vtkOrientedPolygonalHandleRepresentation3D
```

42.64.2 Methods

The class vtkOrientedPolygonalHandleRepresentation3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkOrientedPolygonalHandleRepresentation3D class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkOrientedPolygonalHandleRepresentation3D = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkOrientedPolygonalHandleRepresentation3D = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.

42.65 vtkParallelopipedRepresentation

42.65.1 Usage

This class provides the default geometrical representation for vtkParallelopipedWidget. As a result of interactions of the widget, this representation can take on of the following shapes: ¡p¿1) A parallelopiped. (8 handles, 6 faces) ¡p¿2) Paralleopiped with a chair depression on any one handle. (A chair is a depression on one of the handles that carves inwards so as to allow the user to visualize cuts in the volume). (14 handles, 9 faces).

To create an instance of class vtkParallelopipedRepresentation, simply invoke its constructor as follows

```
obj = vtkParallelopipedRepresentation
```

42.65.2 Methods

The class vtkParallelopipedRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParallelopipedRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkParallelopipedRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.

- `vtkParallelopipedRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.GetActors (vtkPropCollection pc)` - Methods to satisfy the superclass.
- `obj.PlaceWidget (double bounds[6])` - Place the widget in the scene. You can use either of the two APIs : 1) PlaceWidget(double bounds[6]) Creates a cuboid conforming to the said bounds. 2) PlaceWidget(double corners[8][3]) Creates a parallelopiped with corners specified. The order in which corners are specified must obey the following rule: Corner 0 - 1 - 2 - 3 - 0 forms a face Corner 4 - 5 - 6 - 7 - 4 forms a face Corner 0 - 4 - 5 - 1 - 0 forms a face Corner 1 - 5 - 6 - 2 - 1 forms a face Corner 2 - 6 - 7 - 3 - 2 forms a face Corner 3 - 7 - 4 - 0 - 3 forms a face
- `obj.SetInteractionState (int )` - The interaction state may be set from a widget (e.g., PointWidget) or other object. This controls how the interaction with the widget proceeds.
- `obj.GetBoundingPlanes (vtkPlaneCollection pc)` - Get the bounding planes of the object. The first 6 planes will be bounding planes of the parallelopiped. If in chair mode, three additional planes will be present. The last three planes will be those of the chair. The normals of all the planes will point into the object.
- `obj.GetPolyData (vtkPolyData pd)` - The parallelopiped polydata.
- `double = obj.GetBounds ()` - The parallelopiped polydata.
- `obj.SetHandleProperty (vtkProperty )` - Set/Get the handle properties.
- `obj.SetHoveredHandleProperty (vtkProperty )` - Set/Get the handle properties.
- `obj.SetSelectedHandleProperty (vtkProperty )` - Set/Get the handle properties.
- `vtkProperty = obj.GetHandleProperty ()` - Set/Get the handle properties.
- `vtkProperty = obj.GetHoveredHandleProperty ()` - Set/Get the handle properties.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Set/Get the handle properties.
- `obj.SetHandleRepresentation (vtkHandleRepresentation handle)`
- `vtkHandleRepresentation = obj.GetHandleRepresentation (int index)`
- `obj.HandlesOn ()` - Turns the visibility of the handles on/off. Sometimes they may get in the way of visualization.
- `obj.HandlesOff ()` - Turns the visibility of the handles on/off. Sometimes they may get in the way of visualization.
- `vtkProperty = obj.GetFaceProperty ()` - Get the face properties. When a face is being translated, the face gets highlighted with the SelectedFaceProperty.
- `vtkProperty = obj.GetSelectedFaceProperty ()` - Get the face properties. When a face is being translated, the face gets highlighted with the SelectedFaceProperty.
- `vtkProperty = obj.GetOutlineProperty ()` - Get the outline properties. These are the properties with which the parallelopiped wireframe is rendered.
- `vtkProperty = obj.GetSelectedOutlineProperty ()` - Get the outline properties. These are the properties with which the parallelopiped wireframe is rendered.
- `obj.BuildRepresentation ()` - This actually constructs the geometry of the widget from the various data parameters.
- `obj.ReleaseGraphicsResources (vtkWindow w)` - Methods required by vtkProp superclass.

- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods required by vtkProp superclass.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods required by vtkProp superclass.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Given and x-y display coordinate, compute the interaction state of the widget.
- `obj.Translate (double translation[3])`
- `obj.Translate (int X, int Y)`
- `obj.Scale (int X, int Y)`
- `obj.PositionHandles ()` - Synchronize the parallelopiped handle positions with the Polygonal datastructure.
- `obj.SetMinimumThickness (double )` - Minimum thickness for the parallelopiped. User interactions cannot make any individual axis of the parallopiped thinner than this value. Default is 0.05 expressed as a fraction of the diagonal of the bounding box used in the PlaceWidget() invocation.
- `double = obj.GetMinimumThickness ()` - Minimum thickness for the parallelopiped. User interactions cannot make any individual axis of the parallopiped thinner than this value. Default is 0.05 expressed as a fraction of the diagonal of the bounding box used in the PlaceWidget() invocation.

42.66 vtkParallelopipedWidget

42.66.1 Usage

This widget was designed with the aim of visualizing / probing cuts on a skewed image data / structured grid.

.SECTION Interaction The widget allows you to create a parallelopiped (defined by 8 handles). The widget is initially placed by using the "PlaceWidget" method in the representation class. After the widget has been created, the following interactions may be used to manipulate it : 1) Click on a handle and drag it around moves the handle in space, while keeping the same axis alignment of the parallelopiped 2) Dragging a handle with the shift button pressed resizes the piped along an axis. 3) Control-click on a handle creates a chair at that position. (A chair is a depression in the piped that allows you to visualize cuts in the volume). 4) Clicking on a chair and dragging it around moves the chair within the piped. 5) Shift-click on the piped enables you to translate it.

To create an instance of class vtkParallelopipedWidget, simply invoke its constructor as follows

```
obj = vtkParallelopipedWidget
```

42.66.2 Methods

The class vtkParallelopipedWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkParallelopipedWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkParallelopipedWidget = obj.NewInstance ()`
- `vtkParallelopipedWidget = obj.SafeDownCast (vtkObject o)`

- `obj.SetEnabled (int )` - Override the superclass method. This is a composite widget, (it internally consists of handle widgets). We will override the superclass method, so that we can pass the enabled state to the internal widgets as well.
- `obj.SetRepresentation (vtkParallelopipedRepresentation r)` - Enable/disable the creation of a chair on this widget. If off, chairs cannot be created.
- `obj.SetEnableChairCreation (int )` - Enable/disable the creation of a chair on this widget. If off, chairs cannot be created.
- `int = obj.GetEnableChairCreation ()` - Enable/disable the creation of a chair on this widget. If off, chairs cannot be created.
- `obj.EnableChairCreationOn ()` - Enable/disable the creation of a chair on this widget. If off, chairs cannot be created.
- `obj.EnableChairCreationOff ()` - Enable/disable the creation of a chair on this widget. If off, chairs cannot be created.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `obj.SetProcessEvents (int )` - Methods to change the whether the widget responds to interaction. Overridden to pass the state to component widgets.

42.67 vtkPlaneWidget

42.67.1 Usage

This 3D widget defines a finite (bounded) plane that can be interactively placed in a scene. The plane has four handles (at its corner vertices), a normal vector, and the plane itself. The handles are used to resize the plane; the normal vector to rotate it, and the plane can be picked and translated. Selecting the plane while pressing CTRL makes it spin around the normal. A nice feature of the object is that the vtkPlaneWidget, like any 3D widget, will work with the current interactor style. That is, if vtkPlaneWidget does not handle an event, then all other registered observers (including the interactor style) have an opportunity to process the event. Otherwise, the vtkPlaneWidget will terminate the processing of the event that it handles.

To use this object, just invoke SetInteractor() with the argument of the method a vtkRenderWindowInteractor. You may also wish to invoke "PlaceWidget()" to initially position the widget. If the "i" key (for "interactor") is pressed, the vtkPlaneWidget will appear. (See superclass documentation for information about changing this behavior.) By grabbing the one of the four handles (use the left mouse button), the plane can be resized. By grabbing the plane itself, the entire plane can be arbitrarily translated. Pressing CTRL while grabbing the plane will spin the plane around the normal. If you select the normal vector, the plane can be arbitrarily rotated. Selecting any part of the widget with the middle mouse button enables translation of the plane along its normal. (Once selected using middle mouse, moving the mouse in the direction of the normal translates the plane in the direction of the normal; moving in the direction opposite the normal translates the plane in the direction opposite the normal.) Scaling (about the center of the plane) is achieved by using the right mouse button. By moving the mouse "up" the render window the plane will be made bigger; by moving "down" the render window the widget will be made smaller. Events that occur outside of the widget (i.e., no part of the widget is picked) are propagated to any other registered obsevers (such as the interaction style). Turn off the widget by pressing the "i" key again (or invoke the Off() method).

The vtkPlaneWidget has several methods that can be used in conjunction with other VTK objects. The Set/GetResolution() methods control the number of subdivisions of the plane; the GetPolyData() method can be used to get the polygonal representation and can be used for things like seeding stream lines. GetPlane() can be used to update a vtkPlane implicit function. Typical usage of the widget is to make use of the StartInteractionEvent, InteractionEvent, and EndInteractionEvent events. The InteractionEvent is called on mouse motion; the other two events are called on button down and button up (either left or right button).

Some additional features of this class include the ability to control the properties of the widget. You can set the properties of the selected and unselected representations of the plane. For example, you can set the property for the handles and plane. In addition there are methods to constrain the plane so that it is perpendicular to the x-y-z axes.

To create an instance of class vtkPlaneWidget, simply invoke its constructor as follows

```
obj = vtkPlaneWidget
```

42.67.2 Methods

The class vtkPlaneWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPlaneWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPlaneWidget = obj.NewInstance ()`
- `vtkPlaneWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double bounds[6])` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget ()` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Set/Get the resolution (number of subdivisions) of the plane.
- `obj.SetResolution (int r)` - Set/Get the resolution (number of subdivisions) of the plane.
- `int = obj.GetResolution ()` - Set/Get the resolution (number of subdivisions) of the plane.
- `obj.SetOrigin (double x, double y, double z)` - Set/Get the origin of the plane.
- `obj.SetOrigin (double x[3])` - Set/Get the origin of the plane.
- `double = obj.GetOrigin ()` - Set/Get the origin of the plane.
- `obj.GetOrigin (double xyz[3])` - Set/Get the origin of the plane.
- `obj.SetPoint1 (double x, double y, double z)` - Set/Get the position of the point defining the first axis of the plane.
- `obj.SetPoint1 (double x[3])` - Set/Get the position of the point defining the first axis of the plane.
- `double = obj.GetPoint1 ()` - Set/Get the position of the point defining the first axis of the plane.
- `obj.GetPoint1 (double xyz[3])` - Set/Get the position of the point defining the first axis of the plane.
- `obj.SetPoint2 (double x, double y, double z)` - Set/Get the position of the point defining the second axis of the plane.
- `obj.SetPoint2 (double x[3])` - Set/Get the position of the point defining the second axis of the plane.
- `double = obj.GetPoint2 ()` - Set/Get the position of the point defining the second axis of the plane.

- `obj.GetPoint2 (double xyz[3])` - Set/Get the position of the point defining the second axis of the plane.
- `obj.SetCenter (double x, double y, double z)` - Get the center of the plane.
- `obj.SetCenter (double x[3])` - Get the center of the plane.
- `double = obj.GetCenter ()` - Get the center of the plane.
- `obj.GetCenter (double xyz[3])` - Get the center of the plane.
- `obj.SetNormal (double x, double y, double z)` - Get the normal to the plane.
- `obj.SetNormal (double x[3])` - Get the normal to the plane.
- `double = obj.GetNormal ()` - Get the normal to the plane.
- `obj.GetNormal (double xyz[3])` - Get the normal to the plane.
- `obj.SetRepresentation (int )` - Control how the plane appears when GetPolyData() is invoked. If the mode is "outline", then just the outline of the plane is shown. If the mode is "wireframe" then the plane is drawn with the outline plus the interior mesh (corresponding to the resolution specified). If the mode is "surface" then the plane is drawn as a surface.
- `int = obj.GetRepresentationMinValue ()` - Control how the plane appears when GetPolyData() is invoked. If the mode is "outline", then just the outline of the plane is shown. If the mode is "wireframe" then the plane is drawn with the outline plus the interior mesh (corresponding to the resolution specified). If the mode is "surface" then the plane is drawn as a surface.
- `int = obj.GetRepresentationMaxValue ()` - Control how the plane appears when GetPolyData() is invoked. If the mode is "outline", then just the outline of the plane is shown. If the mode is "wireframe" then the plane is drawn with the outline plus the interior mesh (corresponding to the resolution specified). If the mode is "surface" then the plane is drawn as a surface.
- `int = obj.GetRepresentation ()` - Control how the plane appears when GetPolyData() is invoked. If the mode is "outline", then just the outline of the plane is shown. If the mode is "wireframe" then the plane is drawn with the outline plus the interior mesh (corresponding to the resolution specified). If the mode is "surface" then the plane is drawn as a surface.
- `obj.SetRepresentationToOff ()` - Control how the plane appears when GetPolyData() is invoked. If the mode is "outline", then just the outline of the plane is shown. If the mode is "wireframe" then the plane is drawn with the outline plus the interior mesh (corresponding to the resolution specified). If the mode is "surface" then the plane is drawn as a surface.
- `obj.SetRepresentationToOutline ()` - Control how the plane appears when GetPolyData() is invoked. If the mode is "outline", then just the outline of the plane is shown. If the mode is "wireframe" then the plane is drawn with the outline plus the interior mesh (corresponding to the resolution specified). If the mode is "surface" then the plane is drawn as a surface.
- `obj.SetRepresentationToWireframe ()` - Control how the plane appears when GetPolyData() is invoked. If the mode is "outline", then just the outline of the plane is shown. If the mode is "wireframe" then the plane is drawn with the outline plus the interior mesh (corresponding to the resolution specified). If the mode is "surface" then the plane is drawn as a surface.
- `obj.SetRepresentationToSurface ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.SetNormalToXAxis (int )` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.

- `int = obj.GetNormalToXAxis ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToXAxisOn ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToXAxisOff ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.SetNormalToYAxis (int )` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `int = obj.GetNormalToYAxis ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToYAxisOn ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToYAxisOff ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.SetNormalToZAxis (int )` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `int = obj.GetNormalToZAxis ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToZAxisOn ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.NormalToZAxisOff ()` - Force the plane widget to be aligned with one of the x-y-z axes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the plane to the axes if it is orginally not aligned.
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that defines the plane. The polydata consists of (res+1)*(res+1) points, and res*res quadrilateral polygons, where res is the resolution of the plane. These point values are guaranteed to be up-to-date when either the InteractionEvent or EndInteraction events are invoked. The user provides the vtkPolyData and the points and polyplane are added to it.
- `obj.GetPlane (vtkPlane plane)` - Get the planes describing the implicit function defined by the plane widget. The user must provide the instance of the class vtkPlane. Note that vtkPlane is a subclass of vtkImplicitFunction, meaning that it can be used by a variety of filters to perform clipping, cutting, and selection of data.
- `vtkPolyDataAlgorithm = obj.GetPolyDataAlgorithm ()` - Satisfies superclass API. This returns a pointer to the underlying PolyData. Make changes to this before calling the initial PlaceWidget() to have the initial placement follow suit. Or, make changes after the widget has been initialised and call UpdatePlacement() to realise.

- `obj.UpdatePlacement (void )` - Satisfies superclass API. This will change the state of the widget to match changes that have been made to the underlying PolyDataSource
- `vtkProperty = obj.GetHandleProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles when selected and normal can be manipulated.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles when selected and normal can be manipulated.
- `obj.SetPlaneProperty (vtkProperty )` - Get the plane properties. The properties of the plane when selected and unselected can be manipulated.
- `vtkProperty = obj.GetPlaneProperty ()` - Get the plane properties. The properties of the plane when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedPlaneProperty ()` - Get the plane properties. The properties of the plane when selected and unselected can be manipulated.

42.68 vtkPlaybackRepresentation

42.68.1 Usage

This class is used to represent the vtkPlaybackWidget. Besides defining geometry, this class defines a series of virtual method stubs that are meant to be subclassed by applications for controlling playback.

To create an instance of class vtkPlaybackRepresentation, simply invoke its constructor as follows

```
obj = vtkPlaybackRepresentation
```

42.68.2 Methods

The class vtkPlaybackRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPlaybackRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK class methods.
- `int = obj.IsA (string name)` - Standard VTK class methods.
- `vtkPlaybackRepresentation = obj.NewInstance ()` - Standard VTK class methods.
- `vtkPlaybackRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK class methods.
- `vtkProperty2D = obj.GetProperty ()` - By obtaining this property you can specify the properties of the representation.
- `obj.Play ()` - Virtual callbacks that subclasses should implement.
- `obj.Stop ()` - Virtual callbacks that subclasses should implement.
- `obj.ForwardOneFrame ()` - Virtual callbacks that subclasses should implement.
- `obj.BackwardOneFrame ()` - Virtual callbacks that subclasses should implement.
- `obj.JumpToBeginning ()` - Virtual callbacks that subclasses should implement.
- `obj.JumpToEnd ()` - Satisfy the superclasses' API.
- `obj.BuildRepresentation ()` - Satisfy the superclasses' API.

- `obj.GetSize (double size[2])` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.GetActors2D (vtkPropCollection )` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - These methods are necessary to make this representation behave as a vtkProp.

42.69 vtkPlaybackWidget

42.69.1 Usage

This class provides support for interactively controlling the playback of a serial stream of information (e.g., animation sequence, video, etc.). Controls for play, stop, advance one step forward, advance one step backward, jump to beginning, and jump to end are available.

To create an instance of class vtkPlaybackWidget, simply invoke its constructor as follows

```
obj = vtkPlaybackWidget
```

42.69.2 Methods

The class vtkPlaybackWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPlaybackWidget class.

- `string = obj.GetClassName ()` - Standar VTK class methods.
- `int = obj.IsA (string name)` - Standar VTK class methods.
- `vtkPlaybackWidget = obj.NewInstance ()` - Standar VTK class methods.
- `vtkPlaybackWidget = obj.SafeDownCast (vtkObject o)` - Standar VTK class methods.
- `obj.SetRepresentation (vtkPlaybackRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.70 vtkPointHandleRepresentation2D

42.70.1 Usage

This class is used to represent a vtkHandleWidget. It represents a position in 2D world coordinates using a x-y cursor (the cursor defined by an instance of vtkPolyData and generated by a vtkPolyDataAlgorithm).

To create an instance of class vtkPointHandleRepresentation2D, simply invoke its constructor as follows

```
obj = vtkPointHandleRepresentation2D
```

42.70.2 Methods

The class vtkPointHandleRepresentation2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointHandleRepresentation2D class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkPointHandleRepresentation2D = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkPointHandleRepresentation2D = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetCursorShape (vtkPolyData cursorShape)` - Specify the cursor shape with an instance of vtkPolyData. Note that shape is assumed to be defined in the display coordinate system. By default a vtkCursor2D shape is used.
- `vtkPolyData = obj.GetCursorShape ()` - Specify the cursor shape with an instance of vtkPolyData. Note that shape is assumed to be defined in the display coordinate system. By default a vtkCursor2D shape is used.
- `obj.SetDisplayPosition (double xyz[3])` - Set/Get the position of the point in display coordinates. This overloads the superclasses SetDisplayPosition in order to set the focal point of the cursor.
- `obj.SetProperty (vtkProperty2D )` - Set/Get the handle properties when unselected and selected.
- `obj.SetSelectedProperty (vtkProperty2D )` - Set/Get the handle properties when unselected and selected.
- `vtkProperty2D = obj.GetProperty ()` - Set/Get the handle properties when unselected and selected.
- `vtkProperty2D = obj.GetSelectedProperty ()` - Set/Get the handle properties when unselected and selected.
- `double = obj.GetBounds ()` - Subclasses of vtkPointHandleRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.BuildRepresentation ()` - Subclasses of vtkPointHandleRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.StartWidgetInteraction (double eventPos[2])` - Subclasses of vtkPointHandleRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.WidgetInteraction (double eventPos[2])` - Subclasses of vtkPointHandleRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Subclasses of vtkPointHandleRepresentation2D must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.ShallowCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp.
- `obj.DeepCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp.

- `obj.GetActors2D (vtkPropCollection )` - Methods to make this class behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.

42.71 vtkPointHandleRepresentation3D

42.71.1 Usage

This class is used to represent a vtkHandleWidget. It represents a position in 3D world coordinates using a x-y-z cursor. The cursor can be configured to show a bounding box and/or shadows.

To create an instance of class vtkPointHandleRepresentation3D, simply invoke its constructor as follows

```
obj = vtkPointHandleRepresentation3D
```

42.71.2 Methods

The class vtkPointHandleRepresentation3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointHandleRepresentation3D class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkPointHandleRepresentation3D = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkPointHandleRepresentation3D = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetWorldPosition (double p[3])` - Set the position of the point in world and display coordinates. Note that if the position is set outside of the bounding box, it will be clamped to the boundary of the bounding box. This method overloads the superclasses' SetWorldPosition() and SetDisplayPosition() in order to set the focal point of the cursor properly.
- `obj.SetDisplayPosition (double p[3])` - Set the position of the point in world and display coordinates. Note that if the position is set outside of the bounding box, it will be clamped to the boundary of the bounding box. This method overloads the superclasses' SetWorldPosition() and SetDisplayPosition() in order to set the focal point of the cursor properly.
- `obj.SetOutline (int o)` - Turn on/off the wireframe bounding box.
- `int = obj.GetOutline ()` - Turn on/off the wireframe bounding box.
- `obj.OutlineOn ()` - Turn on/off the wireframe bounding box.
- `obj.OutlineOff ()` - Turn on/off the wireframe x-shadows.
- `obj.SetXShadows (int o)` - Turn on/off the wireframe x-shadows.
- `int = obj.GetXShadows ()` - Turn on/off the wireframe x-shadows.
- `obj.XShadowsOn ()` - Turn on/off the wireframe x-shadows.
- `obj.XShadowsOff ()` - Turn on/off the wireframe y-shadows.
- `obj.SetYShadows (int o)` - Turn on/off the wireframe y-shadows.

- `int = obj.GetYShadows ()` - Turn on/off the wireframe y-shadows.
- `obj.YShadowsOn ()` - Turn on/off the wireframe y-shadows.
- `obj.YShadowsOff ()` - Turn on/off the wireframe z-shadows.
- `obj.SetZShadows (int o)` - Turn on/off the wireframe z-shadows.
- `int = obj.GetZShadows ()` - Turn on/off the wireframe z-shadows.
- `obj.ZShadowsOn ()` - Turn on/off the wireframe z-shadows.
- `obj.ZShadowsOff ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated and sized simultaneously as the point moves (i.e., the left and middle mouse buttons act the same). If translation mode is off, the cursor does not scale itself (based on the specified handle size), and the bounding box and shadows do not move or size themselves as the cursor focal point moves, which is constrained by the bounds of the point representation. (Note that the bounds can be scaled up using the right mouse button, and the bounds can be manually set with the SetBounds() method.)
- `obj.SetTranslationMode (int )` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated and sized simultaneously as the point moves (i.e., the left and middle mouse buttons act the same). If translation mode is off, the cursor does not scale itself (based on the specified handle size), and the bounding box and shadows do not move or size themselves as the cursor focal point moves, which is constrained by the bounds of the point representation. (Note that the bounds can be scaled up using the right mouse button, and the bounds can be manually set with the SetBounds() method.)
- `int = obj.GetTranslationMode ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated and sized simultaneously as the point moves (i.e., the left and middle mouse buttons act the same). If translation mode is off, the cursor does not scale itself (based on the specified handle size), and the bounding box and shadows do not move or size themselves as the cursor focal point moves, which is constrained by the bounds of the point representation. (Note that the bounds can be scaled up using the right mouse button, and the bounds can be manually set with the SetBounds() method.)
- `obj.TranslationModeOn ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated and sized simultaneously as the point moves (i.e., the left and middle mouse buttons act the same). If translation mode is off, the cursor does not scale itself (based on the specified handle size), and the bounding box and shadows do not move or size themselves as the cursor focal point moves, which is constrained by the bounds of the point representation. (Note that the bounds can be scaled up using the right mouse button, and the bounds can be manually set with the SetBounds() method.)
- `obj.TranslationModeOff ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated and sized simultaneously as the point moves (i.e., the left and middle mouse buttons act the same). If translation mode is off, the cursor does not scale itself (based on the specified handle size), and the bounding box and shadows do not move or size themselves as the cursor focal point moves, which is constrained by the bounds of the point representation. (Note that the bounds can be scaled up using the right mouse button, and the bounds can be manually set with the SetBounds() method.)
- `obj.AllOn ()` - Convenience methods to turn outline and shadows on and off.
- `obj.AllOff ()` - Set/Get the handle properties when unselected and selected.
- `obj.SetProperty (vtkProperty )` - Set/Get the handle properties when unselected and selected.

- `obj.SetSelectedProperty (vtkProperty )` - Set/Get the handle properties when unselected and selected.
- `vtkProperty = obj.GetProperty ()` - Set/Get the handle properties when unselected and selected.
- `vtkProperty = obj.GetSelectedProperty ()` - Set/Get the handle properties when unselected and selected.
- `obj.SetHotSpotSize (double )` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `double = obj.GetHotSpotSizeMinValue ()` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `double = obj.GetHotSpotSizeMaxValue ()` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `double = obj.GetHotSpotSize ()` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `obj.SetHandleSize (double size)` - Overload the superclasses SetHandleSize() method to update internal variables.
- `double = obj.GetBounds ()` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.BuildRepresentation ()` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.StartWidgetInteraction (double eventPos[2])` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.WidgetInteraction (double eventPos[2])` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.PlaceWidget (double bounds[6])` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.ShallowCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp.
- `obj.GetActors (vtkPropCollection )` - Methods to make this class behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods to make this class behave as a vtkProp.

42.72 vtkPointPlacer

42.72.1 Usage

Most widgets in VTK have a need to translate of 2D display coordinates (as reported by the RenderWindowInteractor) to 3D world coordinates. This class is an abstraction of this functionality. A few subclasses are listed below: ¡p¿1) vtkFocalPlanePointPlacer: This class converts 2D display positions to world positions such that they lie on the focal plane. ¡p¿2) vtkPolygonalSurfacePointPlacer: Converts 2D display positions to world positions such that they lie on the surface of one or more specified polydatas. ¡p¿3) vtkImageActorPointPlacer: Converts 2D display positions to world positions such that they lie on an ImageActor ¡p¿4) vtkBoundedPlanePointPlacer: Converts 2D display positions to world positions such that they lie within a set of specified bounding planes. ¡p¿5) vtkTerrainDataPointPlacer: Converts 2D display positions to world positions such that they lie on a height field. ¡p¿ Point placers provide an extensible framework to specify constraints on points. The methods ComputeWorldPosition, ValidateDisplayPosition and ValidateWorldPosition may be overridden to dictate whether a world or display position is allowed. These classes are currently used by the HandleWidget and the ContourWidget to allow various constraints to be enforced on the placement of their handles.

To create an instance of class vtkPointPlacer, simply invoke its constructor as follows

```
obj = vtkPointPlacer
```

42.72.2 Methods

The class vtkPointPlacer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointPlacer class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkPointPlacer = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkPointPlacer = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double worldPos[3], double w` - Given a renderer and a display position in pixel coordinates, compute the world position and orientation where this point will be placed. This method is typically used by the representation to place the point initially. A return value of 1 indicates that constraints of the placer are met.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double refWorldPos[3], doubl` - Given a renderer, a display position, and a reference world position, compute the new world position and orientation of this point. This method is typically used by the representation to move the point. A return value of 1 indicates that constraints of the placer are met.
- `int = obj.ValidateWorldPosition (double worldPos[3])` - Given a world position check the validity of this position according to the constraints of the placer.
- `int = obj.ValidateDisplayPosition (vtkRenderer , double displayPos[2])` - Given a display position, check the validity of this position.
- `int = obj.ValidateWorldPosition (double worldPos[3], double worldOrient[9])` - Given a world position and a world orientation, validate it according to the constraints of the placer.
- `int = obj.UpdateWorldPosition (vtkRenderer ren, double worldPos[3], double worldOrient[9])` - Given a current renderer, world position and orientation, update them according to the constraints of the placer. This method is typically used when UpdateContour is called on the representation, which must be called after changes are made to the constraints in the placer. A return value of 1 indicates

that the point has been updated. A return value of 0 indicates that the point could not be updated and was left alone. By default this is a no-op - leaving the point as is.

- `int = obj.UpdateInternalState ()` - Set/get the tolerance used when performing computations in display coordinates.
- `obj.SetPixelTolerance (int )` - Set/get the tolerance used when performing computations in display coordinates.
- `int = obj.GetPixelToleranceMinValue ()` - Set/get the tolerance used when performing computations in display coordinates.
- `int = obj.GetPixelToleranceMaxValue ()` - Set/get the tolerance used when performing computations in display coordinates.
- `int = obj.GetPixelTolerance ()` - Set/get the tolerance used when performing computations in display coordinates.
- `obj.SetWorldTolerance (double )` - Set/get the tolerance used when performing computations in world coordinates.
- `double = obj.GetWorldToleranceMinValue ()` - Set/get the tolerance used when performing computations in world coordinates.
- `double = obj.GetWorldToleranceMaxValue ()` - Set/get the tolerance used when performing computations in world coordinates.
- `double = obj.GetWorldTolerance ()` - Set/get the tolerance used when performing computations in world coordinates.

42.73 vtkPointWidget

42.73.1 Usage

This 3D widget allows the user to position a point in 3D space using a 3D cursor. The cursor has an outline bounding box, axes-aligned cross-hairs, and axes shadows. (The outline and shadows can be turned off.) Any of these can be turned off. A nice feature of the object is that the vtkPointWidget, like any 3D widget, will work with the current interactor style. That is, if vtkPointWidget does not handle an event, then all other registered observers (including the interactor style) have an opportunity to process the event. Otherwise, the vtkPointWidget will terminate the processing of the event that it handles.

To use this object, just invoke SetInteractor() with the argument of the method a vtkRenderWindowInteractor. You may also wish to invoke "PlaceWidget()" to initially position the widget. The interactor will act normally until the "i" key (for "interactor") is pressed, at which point the vtkPointWidget will appear. (See superclass documentation for information about changing this behavior.) To move the point, the user can grab (left mouse) on any widget line and "slide" the point into position. Scaling is achieved by using the right mouse button "up" the render window (makes the widget bigger) or "down" the render window (makes the widget smaller). To translate the widget use the middle mouse button. (Note: all of the translation interactions can be constrained to one of the x-y-z axes by using the "shift" key.) The vtkPointWidget produces as output a polydata with a single point and a vertex cell.

Some additional features of this class include the ability to control the rendered properties of the widget. You can set the properties of the selected and unselected representations of the parts of the widget. For example, you can set the property of the 3D cursor in its normal and selected states.

To create an instance of class vtkPointWidget, simply invoke its constructor as follows

```
obj = vtkPointWidget
```

42.73.2 Methods

The class vtkPointWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPointWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkPointWidget = obj.NewInstance ()`
- `vtkPointWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double bounds[6])` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget ()` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Grab the polydata (including points) that defines the point. A single point and a vertex compose the vtkPolyData.
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that defines the point. A single point and a vertex compose the vtkPolyData.
- `obj.SetPosition (double x, double y, double z)` - Set/Get the position of the point. Note that if the position is set outside of the bounding box, it will be clamped to the boundary of the bounding box.
- `obj.SetPosition (double x[3])` - Set/Get the position of the point. Note that if the position is set outside of the bounding box, it will be clamped to the boundary of the bounding box.
- `double = obj.GetPosition ()` - Set/Get the position of the point. Note that if the position is set outside of the bounding box, it will be clamped to the boundary of the bounding box.
- `obj.GetPosition (double xyz[3])` - Turn on/off the wireframe bounding box.
- `obj.SetOutline (int o)` - Turn on/off the wireframe bounding box.
- `int = obj.GetOutline ()` - Turn on/off the wireframe bounding box.
- `obj.OutlineOn ()` - Turn on/off the wireframe bounding box.
- `obj.OutlineOff ()` - Turn on/off the wireframe x-shadows.
- `obj.SetXShadows (int o)` - Turn on/off the wireframe x-shadows.
- `int = obj.GetXShadows ()` - Turn on/off the wireframe x-shadows.
- `obj.XShadowsOn ()` - Turn on/off the wireframe x-shadows.
- `obj.XShadowsOff ()` - Turn on/off the wireframe y-shadows.
- `obj.SetYShadows (int o)` - Turn on/off the wireframe y-shadows.
- `int = obj.GetYShadows ()` - Turn on/off the wireframe y-shadows.
- `obj.YShadowsOn ()` - Turn on/off the wireframe y-shadows.
- `obj.YShadowsOff ()` - Turn on/off the wireframe z-shadows.

- `obj.SetZShadows (int o)` - Turn on/off the wireframe z-shadows.
- `int = obj.GetZShadows ()` - Turn on/off the wireframe z-shadows.
- `obj.ZShadowsOn ()` - Turn on/off the wireframe z-shadows.
- `obj.ZShadowsOff ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated simultaneously as the point moves.
- `obj.SetTranslationMode (int mode)` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated simultaneously as the point moves.
- `int = obj.GetTranslationMode ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated simultaneously as the point moves.
- `obj.TranslationModeOn ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated simultaneously as the point moves.
- `obj.TranslationModeOff ()` - Convenience methods to turn outline and shadows on and off.
- `obj.AllOn ()` - Convenience methods to turn outline and shadows on and off.
- `obj.AllOff ()` - Get the handle properties (the little balls are the handles). The properties of the handles when selected and normal can be set.
- `vtkProperty = obj.GetProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles when selected and normal can be set.
- `vtkProperty = obj.GetSelectedProperty ()` - Get the handle properties (the little balls are the handles). The properties of the handles when selected and normal can be set.
- `obj.SetHotSpotSize (double )` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `double = obj.GetHotSpotSizeMinValue ()` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `double = obj.GetHotSpotSizeMaxValue ()` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `double = obj.GetHotSpotSize ()` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.

42.74 vtkPolyDataContourLineInterpolator

42.74.1 Usage

vtkPolyDataContourLineInterpolator is an abstract base class for contour line interpolators that interpolate on polygonal data.

To create an instance of class vtkPolyDataContourLineInterpolator, simply invoke its constructor as follows

```
obj = vtkPolyDataContourLineInterpolator
```

42.74.2 Methods

The class vtkPolyDataContourLineInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataContourLineInterpolator class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkPolyDataContourLineInterpolator = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkPolyDataContourLineInterpolator = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `int = obj.InterpolateLine (vtkRenderer ren, vtkContourRepresentation rep, int idx1, int idx2)` - Subclasses that wish to interpolate a line segment must implement this. For instance vtkBezierContourLineInterpolator adds nodes between idx1 and idx2, that allow the contour to adhere to a bezier curve.
- `int = obj.UpdateNode (vtkRenderer , vtkContourRepresentation , double , int )` - The interpolator is given a chance to update the node. vtkImageContourLineInterpolator updates the idx'th node in the contour, so it automatically sticks to edges in the vicinity as the user constructs the contour. Returns 0 if the node (world position) is unchanged.
- `vtkPolyDataCollection = obj.GetPolys ()` - Be sure to add polydata on which you wish to place points to this list or they will not be considered for placement.

42.75 vtkPolyDataPointPlacer

42.75.1 Usage

vtkPolyDataPointPlacer is a base class to place points on the surface of polygonal data.

.SECTION Usage The actors that render polygonal data and wish to be considered for placement by this placer are added to the list as

```
placer->AddProp( polyDataActor );
```

To create an instance of class vtkPolyDataPointPlacer, simply invoke its constructor as follows

```
obj = vtkPolyDataPointPlacer
```

42.75.2 Methods

The class vtkPolyDataPointPlacer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataPointPlacer class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkPolyDataPointPlacer = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkPolyDataPointPlacer = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.

- `obj.AddProp (vtkProp )`
- `obj.RemoveViewProp (vtkProp prop)`
- `obj.RemoveAllProps ()`
- `int = obj.HasProp (vtkProp )`
- `int = obj.GetNumberOfProps ()`
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double worldPos[3], double w` - Given a renderer and a display position in pixel coordinates, compute the world position and orientation where this point will be placed. This method is typically used by the representation to place the point initially. For the Terrain point placer this computes world points that lie at the specified height above the terrain.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double refWorldPos[3], doubl` - Given a renderer, a display position, and a reference world position, compute the new world position and orientation of this point. This method is typically used by the representation to move the point.
- `int = obj.ValidateWorldPosition (double worldPos[3])` - Given a world position check the validity of this position according to the constraints of the placer
- `int = obj.ValidateDisplayPosition (vtkRenderer , double displayPos[2])` - Given a display position, check the validity of this position.
- `int = obj.ValidateWorldPosition (double worldPos[3], double worldOrient[9])` - Given a world position and a world orientation, validate it according to the constraints of the placer.
- `vtkPropPicker = obj.GetPropPicker ()` - Get the Prop picker.

42.76 vtkPolyDataSourceWidget

42.76.1 Usage

This abstract class serves as parent to 3D widgets that have simple vtkPolyDataSource instances defining their geometry.

In addition to what is offered by the vtk3DWidget parent, this class makes it possible to manipulate the underlying polydatasource and to PlaceWidget() according to that, instead of having to make use of SetInput() or SetProp3D().

Implementors of child classes HAVE to implement their PlaceWidget(bounds) to check for the existence of Input and Prop3D FIRST. If these don't exist, place according to the underlying PolyDataSource. Child classes also have to imprement UpdatePlacement(), which updates the widget according to the geometry of the underlying PolyDataSource.

To create an instance of class vtkPolyDataSourceWidget, simply invoke its constructor as follows

```
obj = vtkPolyDataSourceWidget
```

42.76.2 Methods

The class vtkPolyDataSourceWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolyDataSourceWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`

- `vtkPolyDataSourceWidget = obj.NewInstance ()`
- `vtkPolyDataSourceWidget = obj.SafeDownCast (vtkObject o)`
- `obj.PlaceWidget ()` - Overrides vtk3DWidget PlaceWidget() so that it doesn't complain if there's no Input and no Prop3D.
- `obj.PlaceWidget (double bounds[6])` - We have to redeclare this abstract, PlaceWidget() requires it. You HAVE to override this in your concrete child classes. If there's no Prop3D and no Input, your PlaceWidget must make use of the underlying PolyDataSource to do its work.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Returns underlying vtkPolyDataSource that determines geometry. This can be modified after which PlaceWidget() or UpdatePlacement() can be called. UpdatePlacement() will always update the planewidget according to the geometry of the underlying PolyDataSource. PlaceWidget() will only make use of this geometry if there is no Input and no Prop3D set.
- `vtkPolyDataSource = obj.GetPolyDataSource ()` - Returns underlying vtkPolyDataSource that determines geometry. This can be modified after which PlaceWidget() or UpdatePlacement() can be called. UpdatePlacement() will always update the planewidget according to the geometry of the underlying PolyDataSource. PlaceWidget() will only make use of this geometry if there is no Input and no Prop3D set.
- `vtkPolyDataAlgorithm = obj.GetPolyDataAlgorithm ()` - Returns underlying vtkPolyDataSource that determines geometry. This can be modified after which PlaceWidget() or UpdatePlacement() can be called. UpdatePlacement() will always update the planewidget according to the geometry of the underlying PolyDataSource. PlaceWidget() will only make use of this geometry if there is no Input and no Prop3D set.
- `obj.UpdatePlacement ()` - If you've made changes to the underlying vtkPolyDataSource AFTER your initial call to PlaceWidget(), use this method to realise the changes in the widget.

42.77 vtkPolygonalHandleRepresentation3D

42.77.1 Usage

This class serves as the geometrical representation of a vtkHandleWidget. The handle can be represented by an arbitrary polygonal data (vtkPolyData), set via SetHandle(vtkPolyData *). The actual position of the handle will be initially assumed to be (0,0,0). You can specify an offset from this position if desired.

To create an instance of class vtkPolygonalHandleRepresentation3D, simply invoke its constructor as follows

```
obj = vtkPolygonalHandleRepresentation3D
```

42.77.2 Methods

The class vtkPolygonalHandleRepresentation3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolygonalHandleRepresentation3D class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkPolygonalHandleRepresentation3D = obj.NewInstance ()` - Standard methods for instances of this class.

- `vtkPolygonalHandleRepresentation3D = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetWorldPosition (double p[3])` - Set the position of the point in world and display coordinates.
- `obj.SetOffset (double , double , double )` - Set/get the offset of the handle position with respect to the handle center, assumed to be the origin.
- `obj.SetOffset (double a[3])` - Set/get the offset of the handle position with respect to the handle center, assumed to be the origin.
- `double = obj. GetOffset ()` - Set/get the offset of the handle position with respect to the handle center, assumed to be the origin.

42.78 vtkPolygonalSurfaceContourLineInterpolator

42.78.1 Usage

vtkPolygonalSurfaceContourLineInterpolator interpolates and places contour points on polygonal surfaces. The class interpolates nodes by computing a *graph* geodesic lying on the polygonal data. By *graph* Geodesic, we mean that the line interpolating the two end points traverses along on the mesh edges so as to form the shortest path. A Dijkstra algorithm is used to compute the path. See vtkDijkstraGraphGeodesicPath.

The class is mean to be used in conjunction with vtkPolygonalSurfacePointPlacer. The reason for this weak coupling is a performance issue, both classes need to perform a cell pick, and coupling avoids multiple cell picks (cell picks are slow).

To create an instance of class vtkPolygonalSurfaceContourLineInterpolator, simply invoke its constructor as follows

```
obj = vtkPolygonalSurfaceContourLineInterpolator
```

42.78.2 Methods

The class vtkPolygonalSurfaceContourLineInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolygonalSurfaceContourLineInterpolator class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkPolygonalSurfaceContourLineInterpolator = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkPolygonalSurfaceContourLineInterpolator = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `int = obj.InterpolateLine (vtkRenderer ren, vtkContourRepresentation rep, int idx1, int idx2)` - Subclasses that wish to interpolate a line segment must implement this. For instance vtkBezierContourLineInterpolator adds nodes between idx1 and idx2, that allow the contour to adhere to a bezier curve.
- `int = obj.UpdateNode (vtkRenderer , vtkContourRepresentation , double , int )` - The interpolator is given a chance to update the node. vtkImageContourLineInterpolator updates the idx'th node in the contour, so it automatically sticks to edges in the vicinity as the user constructs the contour. Returns 0 if the node (world position) is unchanged.

- `obj.SetDistanceOffset (double )` - Height offset at which points may be placed on the polygonal surface. If you specify a non-zero value here, be sure to have computed vertex normals on your input polygonal data. (easily done with vtkPolyDataNormals).
- `double = obj.GetDistanceOffset ()` - Height offset at which points may be placed on the polygonal surface. If you specify a non-zero value here, be sure to have computed vertex normals on your input polygonal data. (easily done with vtkPolyDataNormals).

42.79 vtkPolygonalSurfacePointPlacer

42.79.1 Usage

vtkPolygonalSurfacePointPlacer places points on polygonal data and is meant to be used in conjunction with vtkPolygonalSurfaceContourLineInterpolator.

.SECTION Usage

To create an instance of class vtkPolygonalSurfacePointPlacer, simply invoke its constructor as follows

```
obj = vtkPolygonalSurfacePointPlacer
```

42.79.2 Methods

The class vtkPolygonalSurfacePointPlacer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkPolygonalSurfacePointPlacer class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkPolygonalSurfacePointPlacer = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkPolygonalSurfacePointPlacer = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.AddProp (vtkProp )`
- `obj.RemoveViewProp (vtkProp prop)`
- `obj.RemoveAllProps ()`
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double worldPos[3], double w` - Given a renderer and a display position in pixel coordinates, compute the world position and orientation where this point will be placed. This method is typically used by the representation to place the point initially. For the Terrain point placer this computes world points that lie at the specified height above the terrain.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double refWorldPos[3], doubl` - Given a renderer, a display position, and a reference world position, compute the new world position and orientation of this point. This method is typically used by the representation to move the point.
- `int = obj.ValidateWorldPosition (double worldPos[3])` - Given a world position check the validity of this position according to the constraints of the placer
- `int = obj.ValidateDisplayPosition (vtkRenderer , double displayPos[2])` - Given a display position, check the validity of this position.

- `int = obj.ValidateWorldPosition (double worldPos[3], double worldOrient[9])` - Given a world position and a world orientation, validate it according to the constraints of the placer.
- `vtkCellPicker = obj.GetCellPicker ()` - Get the Prop picker.
- `vtkPolyDataCollection = obj.GetPolys ()` - Be sure to add polydata on which you wish to place points to this list or they will not be considered for placement.
- `obj.SetDistanceOffset (double )` - Height offset at which points may be placed on the polygonal surface. If you specify a non-zero value here, be sure to compute cell normals on your input polygonal data (easily done with vtkPolyDataNormals).
- `double = obj.GetDistanceOffset ()` - Height offset at which points may be placed on the polygonal surface. If you specify a non-zero value here, be sure to compute cell normals on your input polygonal data (easily done with vtkPolyDataNormals).

42.80 vtkRectilinearWipeRepresentation

42.80.1 Usage

This class is used to represent and render a vtkRectilinearWipeWidget. To use this class, you need to specify an instance of a vtkImageRectilinearWipe and vtkImageActor. This provides the information for this representation to construct and place itself.

The class may be subclassed so that alternative representations can be created. The class defines an API and a default implementation that the vtkRectilinearWipeWidget interacts with to render itself in the scene.

To create an instance of class vtkRectilinearWipeRepresentation, simply invoke its constructor as follows

```
obj = vtkRectilinearWipeRepresentation
```

42.80.2 Methods

The class vtkRectilinearWipeRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearWipeRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkRectilinearWipeRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkRectilinearWipeRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetRectilinearWipe (vtkImageRectilinearWipe wipe)` - Specify an instance of vtkImageRectilinearWipe to manipulate.
- `vtkImageRectilinearWipe = obj.GetRectilinearWipe ()` - Specify an instance of vtkImageRectilinearWipe to manipulate.
- `obj.SetImageActor (vtkImageActor imageActor)` - Specify an instance of vtkImageActor to decorate.
- `vtkImageActor = obj.GetImageActor ()` - Specify an instance of vtkImageActor to decorate.
- `obj.SetTolerance (int )` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).

- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).
- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).
- `int = obj.GetTolerance ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered to be on the widget, or on a widget feature (e.g., a corner point or edge).
- `vtkProperty2D = obj.GetProperty ()` - Get the properties for the widget. This can be manipulated to set different colors, line widths, etc.
- `obj.BuildRepresentation ()` - Subclasses of vtkRectilinearWipeRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.StartWidgetInteraction (double eventPos[2])` - Subclasses of vtkRectilinearWipeRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.WidgetInteraction (double eventPos[2])` - Subclasses of vtkRectilinearWipeRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Subclasses of vtkRectilinearWipeRepresentation must implement these methods. These are the methods that the widget and its representation use to communicate with each other.
- `obj.GetActors2D (vtkPropCollection )` - Methods to make this class behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods to make this class behave as a vtkProp.

42.81 vtkRectilinearWipeWidget

42.81.1 Usage

The vtkRectilinearWipeWidget is used to interactively control an instance of vtkImageRectilinearWipe (and an associated vtkImageActor used to display the rectilinear wipe). A rectilinear wipe is a 2x2 checkerboard pattern created by combining two separate images, where various combinations of the checker squares are possible. Using this widget, the user can adjust the layout of the checker pattern, such as moving the center point, moving the horizontal separator, or moving the vertical separator. These capabilities are particularly useful for comparing two images.

To use this widget, specify its representation (by default the representation is an instance of vtkRectilinearWipeProp). The representation generally requires that you specify an instance of vtkImageRectilinearWipe and an instance of vtkImageActor. Other instance variables may also be required to be set – see the documentation for vtkRectilinearWipeProp (or appropriate subclass).

By default, the widget responds to the following events:

```
Selecting the center point, horizontal separator, and verticel separator:
  LeftButtonPressEvent - move the separators
  LeftButtonReleaseEvent - release the separators
  MouseMoveEvent - move the separators
```

Selecting the center point allows you to move the horizontal and vertical separators simultaneously. Otherwise only horizontal or vertical motion is possible/

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkRectilinearWipeWidget's widget events:

```
vtkWidgetEvent::Select -- some part of the widget has been selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Move -- a request for motion has been invoked
```

In turn, when these widget events are processed, the vtkRectilinearWipeWidget invokes the following VTK events (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkRectilinearWipeWidget, simply invoke its constructor as follows

```
obj = vtkRectilinearWipeWidget
```

42.81.2 Methods

The class vtkRectilinearWipeWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkRectilinearWipeWidget class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkRectilinearWipeWidget = obj.NewInstance ()` - Standard macros.
- `vtkRectilinearWipeWidget = obj.SafeDownCast (vtkObject o)` - Standard macros.
- `obj.SetRepresentation (vtkRectilinearWipeRepresentation r)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.82 vtkScalarBarRepresentation

42.82.1 Usage

This class represents a scalar bar for a vtkScalarBarWidget. This class provides support for interactively placing a scalar bar on the 2D overlay plane. The scalar bar is defined by an instance of vtkScalarBarActor.

One specialty of this class is that if the scalar bar is moved near enough to an edge, it's orientation is flipped to match that edge.

To create an instance of class vtkScalarBarRepresentation, simply invoke its constructor as follows

```
obj = vtkScalarBarRepresentation
```

42.82.2 Methods

The class vtkScalarBarRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkScalarBarRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkScalarBarRepresentation = obj.NewInstance ()`
- `vtkScalarBarRepresentation = obj.SafeDownCast (vtkObject o)`
- `vtkScalarBarActor = obj.GetScalarBarActor ()` - The prop that is placed in the renderer.
- `obj.SetScalarBarActor (vtkScalarBarActor )` - The prop that is placed in the renderer.
- `obj.BuildRepresentation ()` - Satisfy the superclass' API.
- `obj.WidgetInteraction (double eventPos[2])` - Satisfy the superclass' API.
- `obj.GetSize (double size[2])` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.GetActors2D (vtkPropCollection collection)` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow window)` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.SetOrientation (int orient)` - Get/Set the orientation.
- `int = obj.GetOrientation ()` - Get/Set the orientation.

42.83 vtkScalarBarWidget

42.83.1 Usage

This class provides support for interactively manipulating the position, size, and orientation of a scalar bar. It listens to Left mouse events and mouse movement. It also listens to Right mouse events and notifies any observers of Right mouse events on this object when they occur. It will change the cursor shape based on its location. If the cursor is over an edge of the scalar bar it will change the cursor shape to a resize edge shape. If the position of a scalar bar is moved to be close to the center of one of the four edges of the viewport, then the scalar bar will change its orientation to align with that edge. This orientation is sticky in that it will stay that orientation until the position is moved close to another edge.

To create an instance of class vtkScalarBarWidget, simply invoke its constructor as follows

```
obj = vtkScalarBarWidget
```

42.83.2 Methods

The class vtkScalarBarWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkScalarBarWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkScalarBarWidget = obj.NewInstance ()`
- `vtkScalarBarWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetRepresentation (vtkScalarBarRepresentation rep)` - Specify an instance of vtkWidgetRepresentation used to represent this widget in the scene. Note that the representation is a subclass of vtkProp so it can be added to the renderer independent of the widget.
- `vtkScalarBarRepresentation = obj.GetScalarBarRepresentation ()` - Get the ScalarBar used by this Widget. One is created automatically.
- `obj.SetScalarBarActor (vtkScalarBarActor actor)` - Get the ScalarBar used by this Widget. One is created automatically.
- `vtkScalarBarActor = obj.GetScalarBarActor ()` - Get the ScalarBar used by this Widget. One is created automatically.
- `obj.SetRepositionable (int )` - Can the widget be moved. On by default. If off, the widget cannot be moved around.

  TODO: This functionality should probably be moved to the superclass.
- `int = obj.GetRepositionable ()` - Can the widget be moved. On by default. If off, the widget cannot be moved around.

  TODO: This functionality should probably be moved to the superclass.
- `obj.RepositionableOn ()` - Can the widget be moved. On by default. If off, the widget cannot be moved around.

  TODO: This functionality should probably be moved to the superclass.
- `obj.RepositionableOff ()` - Can the widget be moved. On by default. If off, the widget cannot be moved around.

  TODO: This functionality should probably be moved to the superclass.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.84 vtkSeedRepresentation

42.84.1 Usage

The vtkSeedRepresentation is a superclass for classes representing the vtkSeedWidget. This representation consists of one or more handles (vtkHandleRepresentation) which are used to place and manipulate the points defining the collection of seeds.

To create an instance of class vtkSeedRepresentation, simply invoke its constructor as follows

```
obj = vtkSeedRepresentation
```

42.84.2 Methods

The class vtkSeedRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSeedRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkSeedRepresentation = obj.NewInstance ()` - Standard VTK methods.
- `vtkSeedRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `obj.GetSeedWorldPosition (int seedNum, double pos[3])` - Methods to Set/Get the coordinates of seed points defining this representation. Note that methods are available for both display and world coordinates. The seeds are accessed by a seed number.
- `obj.SetSeedDisplayPosition (int seedNum, double pos[3])` - Methods to Set/Get the coordinates of seed points defining this representation. Note that methods are available for both display and world coordinates. The seeds are accessed by a seed number.
- `obj.GetSeedDisplayPosition (int seedNum, double pos[3])` - Methods to Set/Get the coordinates of seed points defining this representation. Note that methods are available for both display and world coordinates. The seeds are accessed by a seed number.
- `int = obj.GetNumberOfSeeds ()` - Return the number of seeds (or handles) that have been created.
- `obj.SetHandleRepresentation (vtkHandleRepresentation handle)` - This method is used to specify the type of handle representation to use for the internal vtkHandleWidgets within vtkSeedWidget. To use this method, create a dummy vtkHandleWidget (or subclass), and then invoke this method with this dummy. Then the vtkSeedRepresentation uses this dummy to clone vtkHandleWidgets of the same type. Make sure you set the handle representation before the widget is enabled.
- `vtkHandleRepresentation = obj.GetHandleRepresentation (int num)` - Get the handle representations used for a particular seed. A side effect of this method is that it will create a handle representation in the list of representations if one has not yet been created.
- `vtkHandleRepresentation = obj.GetHandleRepresentation ()` - Returns the model HandleRepresentation.
- `obj.SetTolerance (int )` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the end points of the widget to be active.
- `int = obj.GetToleranceMinValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the end points of the widget to be active.
- `int = obj.GetToleranceMaxValue ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the end points of the widget to be active.
- `int = obj.GetTolerance ()` - The tolerance representing the distance to the widget (in pixels) in which the cursor is considered near enough to the end points of the widget to be active.
- `int = obj.GetActiveHandle ()` - These are methods specific to vtkSeedRepresentation and which are invoked from vtkSeedWidget.
- `int = obj.CreateHandle (double e[2])` - These are methods specific to vtkSeedRepresentation and which are invoked from vtkSeedWidget.

- `obj.RemoveLastHandle ()` - These are methods specific to vtkSeedRepresentation and which are invoked from vtkSeedWidget.
- `obj.RemoveActiveHandle ()` - These are methods specific to vtkSeedRepresentation and which are invoked from vtkSeedWidget.
- `obj.RemoveHandle (int n)` - Remove the nth handle.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - These are methods that satisfy vtkWidgetRepresentation's API.

42.85 vtkSeedWidget

42.85.1 Usage

The vtkSeedWidget is used to placed multiple seed points in the scene. The seed points can be used for operations like connectivity, segmentation, and region growing.

To use this widget, specify an instance of vtkSeedWidget and a representation (a subclass of vtkSeedRepresentation). The widget is implemented using multiple instances of vtkHandleWidget which can be used to position the seed points (after they are initially placed). The representations for these handle widgets are provided by the vtkSeedRepresentation.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
LeftButtonPressEvent - add a point or select a handle (i.e., seed)
RightButtonPressEvent - finish adding the seeds
MouseMoveEvent - move a handle (i.e., seed)
LeftButtonReleaseEvent - release the selected handle (seed)
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkSeedWidget's widget events:

```
vtkWidgetEvent::AddPoint -- add one point; depending on the state
                            it may the first or second point added. Or,
                            if near handle, select handle.
vtkWidgetEvent::Completed -- finished adding seeds.
vtkWidgetEvent::Move -- move the second point or handle depending on the state.
vtkWidgetEvent::EndSelect -- the handle manipulation process has completed.
```

This widget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (beginning to interact)
vtkCommand::EndInteractionEvent (completing interaction)
vtkCommand::InteractionEvent (moving after selecting something)
vtkCommand::PlacePointEvent (after point is positioned;
                             call data includes handle id (0,1))
```

To create an instance of class vtkSeedWidget, simply invoke its constructor as follows

```
obj = vtkSeedWidget
```

42.85.2 Methods

The class vtkSeedWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSeedWidget class.

- `string = obj.GetClassName ()` - Standard methods for a VTK class.
- `int = obj.IsA (string name)` - Standard methods for a VTK class.
- `vtkSeedWidget = obj.NewInstance ()` - Standard methods for a VTK class.
- `vtkSeedWidget = obj.SafeDownCast (vtkObject o)` - Standard methods for a VTK class.
- `obj.SetEnabled (int )` - The method for activiating and deactiviating this widget. This method must be overridden because it is a composite widget and does more than its superclasses' vtkAbstractWidget::SetEnabled() method.
- `obj.SetCurrentRenderer (vtkRenderer )` - Set the current renderer. This method also propagates to all the child handle widgets, if any exist
- `obj.SetInteractor (vtkRenderWindowInteractor )` - Set the interactor. This method also propagates to all the child handle widgets, if any exist
- `obj.SetRepresentation (vtkSeedRepresentation rep)` - Create the default widget representation if one is not set.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.
- `obj.SetProcessEvents (int )` - Methods to change the whether the widget responds to interaction. Overridden to pass the state to component widgets.
- `obj.CompleteInteraction ()` - Method to be called when the seed widget should stop responding to the place point interaction. The seed widget, when defined allows you place seeds by clicking on the render window. Use this method to indicate that you would like to stop placing seeds interactively. If you'd like the widget to stop responding to *any* user interaction simply disable event processing by the widget by calling widget-¿ProcessEventsOff()
- `obj.RestartInteraction ()` - Method to be called when the seed widget should start responding to the interaction.
- `vtkHandleWidget = obj.CreateNewHandle ()` - Use this method to programmatically create a new handle. In interactive mode, (when the widget is in the PlacingSeeds state) this method is automatically invoked. The method returns the handle created. A valid seed representation must exist for the widget to create a new handle.
- `obj.DeleteSeed (int n)` - Delete the nth seed.
- `vtkHandleWidget = obj.GetSeed (int n)` - Get the nth seed

42.86 vtkSliderRepresentation

42.86.1 Usage

This abstract class is used to specify how the vtkSliderWidget should interact with representations of the vtkSliderWidget. This class may be subclassed so that alternative representations can be created. The class defines an API, and a default implementation, that the vtkSliderWidget interacts with to render itself in the scene.

To create an instance of class vtkSliderRepresentation, simply invoke its constructor as follows

```
obj = vtkSliderRepresentation
```

42.86.2 Methods

The class vtkSliderRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSliderRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkSliderRepresentation = obj.NewInstance ()` - Standard methods for the class.
- `vtkSliderRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.
- `obj.SetValue (double value)` - Specify the current value for the widget. The value should lie between the minimum and maximum values.
- `double = obj.GetValue ()` - Specify the current value for the widget. The value should lie between the minimum and maximum values.
- `obj.SetMinimumValue (double value)` - Set the current minimum value that the slider can take. Setting the minimum value greater than the maximum value will cause the maximum value to grow to (minimum value + 1).
- `double = obj.GetMinimumValue ()` - Set the current minimum value that the slider can take. Setting the minimum value greater than the maximum value will cause the maximum value to grow to (minimum value + 1).
- `obj.SetMaximumValue (double value)` - Set the current maximum value that the slider can take. Setting the maximum value less than the minimum value will cause the minimum value to change to (maximum value - 1).
- `double = obj.GetMaximumValue ()` - Set the current maximum value that the slider can take. Setting the maximum value less than the minimum value will cause the minimum value to change to (maximum value - 1).
- `obj.SetSliderLength (double )` - Specify the length of the slider shape (in normalized display coordinates [0.01,0.5]). The slider length by default is 0.05.
- `double = obj.GetSliderLengthMinValue ()` - Specify the length of the slider shape (in normalized display coordinates [0.01,0.5]). The slider length by default is 0.05.
- `double = obj.GetSliderLengthMaxValue ()` - Specify the length of the slider shape (in normalized display coordinates [0.01,0.5]). The slider length by default is 0.05.
- `double = obj.GetSliderLength ()` - Specify the length of the slider shape (in normalized display coordinates [0.01,0.5]). The slider length by default is 0.05.
- `obj.SetSliderWidth (double )` - Set the width of the slider in the directions orthogonal to the slider axis. Using this it is possible to create ellipsoidal and hockey puck sliders (in some subclasses). By default the width is 0.05.
- `double = obj.GetSliderWidthMinValue ()` - Set the width of the slider in the directions orthogonal to the slider axis. Using this it is possible to create ellipsoidal and hockey puck sliders (in some subclasses). By default the width is 0.05.
- `double = obj.GetSliderWidthMaxValue ()` - Set the width of the slider in the directions orthogonal to the slider axis. Using this it is possible to create ellipsoidal and hockey puck sliders (in some subclasses). By default the width is 0.05.

- `double = obj.GetSliderWidth ()` - Set the width of the slider in the directions orthogonal to the slider axis. Using this it is possible to create ellipsoidal and hockey puck sliders (in some subclasses). By default the width is 0.05.
- `obj.SetTubeWidth (double )` - Set the width of the tube (in normalized display coordinates) on which the slider moves. By default the width is 0.05.
- `double = obj.GetTubeWidthMinValue ()` - Set the width of the tube (in normalized display coordinates) on which the slider moves. By default the width is 0.05.
- `double = obj.GetTubeWidthMaxValue ()` - Set the width of the tube (in normalized display coordinates) on which the slider moves. By default the width is 0.05.
- `double = obj.GetTubeWidth ()` - Set the width of the tube (in normalized display coordinates) on which the slider moves. By default the width is 0.05.
- `obj.SetEndCapLength (double )` - Specify the length of each end cap (in normalized coordinates [0.0,0.25]). By default the length is 0.025. If the end cap length is set to 0.0, then the end cap will not display at all.
- `double = obj.GetEndCapLengthMinValue ()` - Specify the length of each end cap (in normalized coordinates [0.0,0.25]). By default the length is 0.025. If the end cap length is set to 0.0, then the end cap will not display at all.
- `double = obj.GetEndCapLengthMaxValue ()` - Specify the length of each end cap (in normalized coordinates [0.0,0.25]). By default the length is 0.025. If the end cap length is set to 0.0, then the end cap will not display at all.
- `double = obj.GetEndCapLength ()` - Specify the length of each end cap (in normalized coordinates [0.0,0.25]). By default the length is 0.025. If the end cap length is set to 0.0, then the end cap will not display at all.
- `obj.SetEndCapWidth (double )` - Specify the width of each end cap (in normalized coordinates [0.0,0.25]). By default the width is twice the tube width.
- `double = obj.GetEndCapWidthMinValue ()` - Specify the width of each end cap (in normalized coordinates [0.0,0.25]). By default the width is twice the tube width.
- `double = obj.GetEndCapWidthMaxValue ()` - Specify the width of each end cap (in normalized coordinates [0.0,0.25]). By default the width is twice the tube width.
- `double = obj.GetEndCapWidth ()` - Specify the width of each end cap (in normalized coordinates [0.0,0.25]). By default the width is twice the tube width.
- `obj.SetTitleText (string )` - Specify the label text for this widget. If the value is not set, or set to the empty string "", then the label text is not displayed.
- `string = obj.GetTitleText ()` - Set/Get the format with which to print the slider value.
- `obj.SetLabelFormat (string )` - Set/Get the format with which to print the slider value.
- `string = obj.GetLabelFormat ()` - Set/Get the format with which to print the slider value.
- `obj.SetLabelHeight (double )` - Specify the relative height of the label as compared to the length of the slider.
- `double = obj.GetLabelHeightMinValue ()` - Specify the relative height of the label as compared to the length of the slider.
- `double = obj.GetLabelHeightMaxValue ()` - Specify the relative height of the label as compared to the length of the slider.

- `double = obj.GetLabelHeight ()` - Specify the relative height of the label as compared to the length of the slider.
- `obj.SetTitleHeight (double )` - Specify the relative height of the title as compared to the length of the slider.
- `double = obj.GetTitleHeightMinValue ()` - Specify the relative height of the title as compared to the length of the slider.
- `double = obj.GetTitleHeightMaxValue ()` - Specify the relative height of the title as compared to the length of the slider.
- `double = obj.GetTitleHeight ()` - Specify the relative height of the title as compared to the length of the slider.
- `obj.SetShowSliderLabel (int )` - Indicate whether the slider text label should be displayed. This is a number corresponding to the current Value of this widget.
- `int = obj.GetShowSliderLabel ()` - Indicate whether the slider text label should be displayed. This is a number corresponding to the current Value of this widget.
- `obj.ShowSliderLabelOn ()` - Indicate whether the slider text label should be displayed. This is a number corresponding to the current Value of this widget.
- `obj.ShowSliderLabelOff ()` - Indicate whether the slider text label should be displayed. This is a number corresponding to the current Value of this widget.
- `double = obj.GetCurrentT ()` - Methods to interface with the vtkSliderWidget. Subclasses of this class actually do something.
- `double = obj.GetPickedT ()`

42.87 vtkSliderRepresentation2D

42.87.1 Usage

This class is used to represent and render a vtkSliderWidget. To use this class, you must at a minimum specify the end points of the slider. Optional instance variable can be used to modify the appearance of the widget.

To create an instance of class vtkSliderRepresentation2D, simply invoke its constructor as follows

```
obj = vtkSliderRepresentation2D
```

42.87.2 Methods

The class vtkSliderRepresentation2D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSliderRepresentation2D class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkSliderRepresentation2D = obj.NewInstance ()` - Standard methods for the class.
- `vtkSliderRepresentation2D = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.

- `vtkCoordinate = obj.GetPoint1Coordinate ()` - Position the first end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point1Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.

- `vtkCoordinate = obj.GetPoint2Coordinate ()` - Position the second end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point2Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.

- `obj.SetTitleText (string )` - Specify the label text for this widget. If the value is not set, or set to the empty string "", then the label text is not displayed.

- `string = obj.GetTitleText ()` - Specify the label text for this widget. If the value is not set, or set to the empty string "", then the label text is not displayed.

- `vtkProperty2D = obj.GetSliderProperty ()` - Get the slider properties. The properties of the slider when selected and unselected can be manipulated.

- `vtkProperty2D = obj.GetTubeProperty ()` - Get the properties for the tube and end caps.

- `vtkProperty2D = obj.GetCapProperty ()` - Get the properties for the tube and end caps.

- `vtkProperty2D = obj.GetSelectedProperty ()` - Get the selection property. This property is used to modify the appearance of selected objects (e.g., the slider).

- `vtkTextProperty = obj.GetLabelProperty ()` - Set/Get the properties for the label and title text.

- `vtkTextProperty = obj.GetTitleProperty ()` - Set/Get the properties for the label and title text.

- `obj.PlaceWidget (double bounds[6])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.

- `obj.BuildRepresentation ()` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.

- `obj.StartWidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.

- `obj.WidgetInteraction (double newEventPos[2])` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.

- `obj.Highlight (int )` - Methods to interface with the vtkSliderWidget. The PlaceWidget() method assumes that the parameter bounds[6] specifies the location in display space where the widget should be placed.

- `obj.GetActors2D (vtkPropCollection )`

- `obj.ReleaseGraphicsResources (vtkWindow )`

- `int = obj.RenderOverlay (vtkViewport )`

- `int = obj.RenderOpaqueGeometry (vtkViewport )`

42.88 vtkSliderRepresentation3D

42.88.1 Usage

This class is used to represent and render a vtkSliderWidget. To use this class, you must at a minimum specify the end points of the slider. Optional instance variable can be used to modify the appearance of the widget.

To create an instance of class vtkSliderRepresentation3D, simply invoke its constructor as follows

```
obj = vtkSliderRepresentation3D
```

42.88.2 Methods

The class vtkSliderRepresentation3D has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSliderRepresentation3D class.

- `string = obj.GetClassName ()` - Standard methods for the class.
- `int = obj.IsA (string name)` - Standard methods for the class.
- `vtkSliderRepresentation3D = obj.NewInstance ()` - Standard methods for the class.
- `vtkSliderRepresentation3D = obj.SafeDownCast (vtkObject o)` - Standard methods for the class.
- `vtkCoordinate = obj.GetPoint1Coordinate ()` - Position the first end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point1Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.
- `obj.SetPoint1InWorldCoordinates (double x, double y, double z)` - Position the first end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point1Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.
- `vtkCoordinate = obj.GetPoint2Coordinate ()` - Position the second end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point2Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.
- `obj.SetPoint2InWorldCoordinates (double x, double y, double z)` - Position the second end point of the slider. Note that this point is an instance of vtkCoordinate, meaning that Point 1 can be specified in a variety of coordinate systems, and can even be relative to another point. To set the point, you'll want to get the Point2Coordinate and then invoke the necessary methods to put it into the correct coordinate system and set the correct initial value.
- `obj.SetTitleText (string )` - Specify the title text for this widget. If the value is not set, or set to the empty string "", then the title text is not displayed.
- `string = obj.GetTitleText ()` - Specify the title text for this widget. If the value is not set, or set to the empty string "", then the title text is not displayed.
- `obj.SetSliderShape (int )` - Specify whether to use a sphere or cylinder slider shape. By default, a sphere shape is used.

- `int = obj.GetSliderShapeMinValue ()` - Specify whether to use a sphere or cylinder slider shape. By default, a sphere shape is used.
- `int = obj.GetSliderShapeMaxValue ()` - Specify whether to use a sphere or cylinder slider shape. By default, a sphere shape is used.
- `int = obj.GetSliderShape ()` - Specify whether to use a sphere or cylinder slider shape. By default, a sphere shape is used.
- `obj.SetSliderShapeToSphere ()` - Specify whether to use a sphere or cylinder slider shape. By default, a sphere shape is used.
- `obj.SetSliderShapeToCylinder ()` - Set the rotation of the slider widget around the axis of the widget. This is used to control which way the widget is initially oriented. (This is especially important for the label and title.)
- `obj.SetRotation (double )` - Set the rotation of the slider widget around the axis of the widget. This is used to control which way the widget is initially oriented. (This is especially important for the label and title.)
- `double = obj.GetRotation ()` - Set the rotation of the slider widget around the axis of the widget. This is used to control which way the widget is initially oriented. (This is especially important for the label and title.)
- `vtkProperty = obj.GetSliderProperty ()` - Get the slider properties. The properties of the slider when selected and unselected can be manipulated.
- `vtkProperty = obj.GetTubeProperty ()` - Get the properties for the tube and end caps.
- `vtkProperty = obj.GetCapProperty ()` - Get the properties for the tube and end caps.
- `vtkProperty = obj.GetSelectedProperty ()` - Get the selection property. This property is used to modify the appearance of selected objects (e.g., the slider).
- `obj.PlaceWidget (double bounds[6])` - Methods to interface with the vtkSliderWidget.
- `obj.BuildRepresentation ()` - Methods to interface with the vtkSliderWidget.
- `obj.StartWidgetInteraction (double eventPos[2])` - Methods to interface with the vtkSliderWidget.
- `obj.WidgetInteraction (double newEventPos[2])` - Methods to interface with the vtkSliderWidget.
- `obj.Highlight (int )` - Methods to interface with the vtkSliderWidget.
- `double = obj.GetBounds ()`
- `obj.GetActors (vtkPropCollection )`
- `obj.ReleaseGraphicsResources (vtkWindow )`
- `int = obj.RenderOpaqueGeometry (vtkViewport )`
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )`
- `int = obj.HasTranslucentPolygonalGeometry ()`
- `long = obj.GetMTime ()` - Override GetMTime to include point coordinates

42.89 vtkSliderWidget

42.89.1 Usage

The vtkSliderWidget is used to set a scalar value in an application. This class assumes that a slider is moved along a 1D parameter space (e.g., a spherical bead that can be moved along a tube). Moving the slider modifies the value of the widget, which can be used to set parameters on other objects. Note that the actual appearance of the widget depends on the specific representation for the widget.

To use this widget, set the widget representation. The representation is assumed to consist of a tube, two end caps, and a slider (the details may vary depending on the particulars of the representation). Then in the representation you will typically set minimum and maximum value, as well as the current value. The position of the slider must also be set, as well as various properties.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
If the slider bead is selected:
  LeftButtonPressEvent - select slider (if on slider)
  LeftButtonReleaseEvent - release slider (if selected)
  MouseMoveEvent - move slider
If the end caps or slider tube are selected:
  LeftButtonPressEvent - move (or animate) to cap or point on tube;
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkSliderWidget's widget events:

```
vtkWidgetEvent::Select -- some part of the widget has been selected
vtkWidgetEvent::EndSelect -- the selection process has completed
vtkWidgetEvent::Move -- a request for slider motion has been invoked
```

In turn, when these widget events are processed, the vtkSliderWidget invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkSliderWidget, simply invoke its constructor as follows

```
obj = vtkSliderWidget
```

42.89.2 Methods

The class vtkSliderWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSliderWidget class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkSliderWidget = obj.NewInstance ()` - Standard macros.
- `vtkSliderWidget = obj.SafeDownCast (vtkObject o)` - Standard macros.

- `obj.SetRepresentation (vtkSliderRepresentation r)` - Control the behavior of the slider when selecting the tube or caps. If Jump, then selecting the tube, left cap, or right cap causes the slider to jump to the selection point. If the mode is Animate, the slider moves towards the selection point in NumberOfAnimationSteps number of steps. If Off, then the slider does not move.
- `obj.SetAnimationMode (int )` - Control the behavior of the slider when selecting the tube or caps. If Jump, then selecting the tube, left cap, or right cap causes the slider to jump to the selection point. If the mode is Animate, the slider moves towards the selection point in NumberOfAnimationSteps number of steps. If Off, then the slider does not move.
- `int = obj.GetAnimationModeMinValue ()` - Control the behavior of the slider when selecting the tube or caps. If Jump, then selecting the tube, left cap, or right cap causes the slider to jump to the selection point. If the mode is Animate, the slider moves towards the selection point in NumberOfAnimationSteps number of steps. If Off, then the slider does not move.
- `int = obj.GetAnimationModeMaxValue ()` - Control the behavior of the slider when selecting the tube or caps. If Jump, then selecting the tube, left cap, or right cap causes the slider to jump to the selection point. If the mode is Animate, the slider moves towards the selection point in NumberOfAnimationSteps number of steps. If Off, then the slider does not move.
- `int = obj.GetAnimationMode ()` - Control the behavior of the slider when selecting the tube or caps. If Jump, then selecting the tube, left cap, or right cap causes the slider to jump to the selection point. If the mode is Animate, the slider moves towards the selection point in NumberOfAnimationSteps number of steps. If Off, then the slider does not move.
- `obj.SetAnimationModeToOff ()` - Control the behavior of the slider when selecting the tube or caps. If Jump, then selecting the tube, left cap, or right cap causes the slider to jump to the selection point. If the mode is Animate, the slider moves towards the selection point in NumberOfAnimationSteps number of steps. If Off, then the slider does not move.
- `obj.SetAnimationModeToJump ()` - Control the behavior of the slider when selecting the tube or caps. If Jump, then selecting the tube, left cap, or right cap causes the slider to jump to the selection point. If the mode is Animate, the slider moves towards the selection point in NumberOfAnimationSteps number of steps. If Off, then the slider does not move.
- `obj.SetAnimationModeToAnimate ()` - Specify the number of animation steps to take if the animation mode is set to animate.
- `obj.SetNumberOfAnimationSteps (int )` - Specify the number of animation steps to take if the animation mode is set to animate.
- `int = obj.GetNumberOfAnimationStepsMinValue ()` - Specify the number of animation steps to take if the animation mode is set to animate.
- `int = obj.GetNumberOfAnimationStepsMaxValue ()` - Specify the number of animation steps to take if the animation mode is set to animate.
- `int = obj.GetNumberOfAnimationSteps ()` - Specify the number of animation steps to take if the animation mode is set to animate.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.90 vtkSphereHandleRepresentation

42.90.1 Usage

This class is a concrete implementation of vtkHandleRepresentation. It renders handles as spherical blobs in 3D space.

To create an instance of class vtkSphereHandleRepresentation, simply invoke its constructor as follows

```
obj = vtkSphereHandleRepresentation
```

42.90.2 Methods

The class vtkSphereHandleRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSphereHandleRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkSphereHandleRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkSphereHandleRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetWorldPosition (double p[3])` - Set the position of the point in world and display coordinates. Note that if the position is set outside of the bounding box, it will be clamped to the boundary of the bounding box. This method overloads the superclasses' SetWorldPosition() and SetDisplayPosition() in order to set the focal point of the cursor properly.
- `obj.SetDisplayPosition (double p[3])` - Set the position of the point in world and display coordinates. Note that if the position is set outside of the bounding box, it will be clamped to the boundary of the bounding box. This method overloads the superclasses' SetWorldPosition() and SetDisplayPosition() in order to set the focal point of the cursor properly.
- `obj.SetTranslationMode (int )` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated simultaneously as the point moves (i.e., the left and middle mouse buttons act the same). Otherwise, only the cursor focal point moves, which is constrained by the bounds of the point representation. (Note that the bounds can be scaled up using the right mouse button.)
- `int = obj.GetTranslationMode ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated simultaneously as the point moves (i.e., the left and middle mouse buttons act the same). Otherwise, only the cursor focal point moves, which is constrained by the bounds of the point representation. (Note that the bounds can be scaled up using the right mouse button.)
- `obj.TranslationModeOn ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated simultaneously as the point moves (i.e., the left and middle mouse buttons act the same). Otherwise, only the cursor focal point moves, which is constrained by the bounds of the point representation. (Note that the bounds can be scaled up using the right mouse button.)
- `obj.TranslationModeOff ()` - If translation mode is on, as the widget is moved the bounding box, shadows, and cursor are all translated simultaneously as the point moves (i.e., the left and middle mouse buttons act the same). Otherwise, only the cursor focal point moves, which is constrained by the bounds of the point representation. (Note that the bounds can be scaled up using the right mouse button.)
- `obj.SetSphereRadius (double )`
- `double = obj.GetSphereRadius ()`
- `obj.SetProperty (vtkProperty )` - Set/Get the handle properties when unselected and selected.

- `obj.SetSelectedProperty (vtkProperty )` - Set/Get the handle properties when unselected and selected.
- `vtkProperty = obj.GetProperty ()` - Set/Get the handle properties when unselected and selected.
- `vtkProperty = obj.GetSelectedProperty ()` - Set/Get the handle properties when unselected and selected.
- `obj.SetHotSpotSize (double )` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `double = obj.GetHotSpotSizeMinValue ()` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `double = obj.GetHotSpotSizeMaxValue ()` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `double = obj.GetHotSpotSize ()` - Set the "hot spot" size; i.e., the region around the focus, in which the motion vector is used to control the constrained sliding action. Note the size is specified as a fraction of the length of the diagonal of the point widget's bounding box.
- `obj.SetHandleSize (double size)` - Overload the superclasses SetHandleSize() method to update internal variables.
- `double = obj.GetBounds ()` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.BuildRepresentation ()` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.StartWidgetInteraction (double eventPos[2])` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.WidgetInteraction (double eventPos[2])` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.PlaceWidget (double bounds[6])` - Methods to make this class properly act like a vtkWidgetRepresentation.
- `obj.ShallowCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp.
- `obj.DeepCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp.
- `obj.GetActors (vtkPropCollection )` - Methods to make this class behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport viewport)` - Methods to make this class behave as a vtkProp.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods to make this class behave as a vtkProp.

42.91 vtkSphereRepresentation

42.91.1 Usage

This class is a concrete representation for the vtkSphereWidget2. It represents a sphere with an optional handle. Through interaction with the widget, the sphere can be arbitrarily positioned and scaled in 3D space; and the handle can be moved on the surface of the sphere. Typically the vtkSphereWidget2/vtkSphereRepresentation are used to position a sphere for the purpose of extracting, cutting or clipping data; or the handle is moved on the sphere to position a light or camera.

To use this representation, you normally use the PlaceWidget() method to position the widget at a specified region in space. It is also possible to set the center of the sphere, a radius, and/or a handle position.

To create an instance of class vtkSphereRepresentation, simply invoke its constructor as follows

```
obj = vtkSphereRepresentation
```

42.91.2 Methods

The class vtkSphereRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSphereRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for type information and to print out the contents of the class.
- `int = obj.IsA (string name)` - Standard methods for type information and to print out the contents of the class.
- `vtkSphereRepresentation = obj.NewInstance ()` - Standard methods for type information and to print out the contents of the class.
- `vtkSphereRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for type information and to print out the contents of the class.
- `obj.SetRepresentation (int )` - Set the representation (i.e., appearance) of the sphere. Different representations are useful depending on the application.
- `int = obj.GetRepresentationMinValue ()` - Set the representation (i.e., appearance) of the sphere. Different representations are useful depending on the application.
- `int = obj.GetRepresentationMaxValue ()` - Set the representation (i.e., appearance) of the sphere. Different representations are useful depending on the application.
- `int = obj.GetRepresentation ()` - Set the representation (i.e., appearance) of the sphere. Different representations are useful depending on the application.
- `obj.SetRepresentationToOff ()` - Set the representation (i.e., appearance) of the sphere. Different representations are useful depending on the application.
- `obj.SetRepresentationToWireframe ()` - Set the representation (i.e., appearance) of the sphere. Different representations are useful depending on the application.
- `obj.SetRepresentationToSurface ()` - Set/Get the resolution of the sphere in the theta direction.
- `obj.SetThetaResolution (int r)` - Set/Get the resolution of the sphere in the theta direction.
- `int = obj.GetThetaResolution ()` - Set/Get the resolution of the sphere in the phi direction.
- `obj.SetPhiResolution (int r)` - Set/Get the resolution of the sphere in the phi direction.

- `int = obj.GetPhiResolution ()` - Set/Get the center position of the sphere. Note that this may adjust the direction from the handle to the center, as well as the radius of the sphere.
- `obj.SetCenter (double c[3])` - Set/Get the center position of the sphere. Note that this may adjust the direction from the handle to the center, as well as the radius of the sphere.
- `obj.SetCenter (double x, double y, double z)` - Set/Get the center position of the sphere. Note that this may adjust the direction from the handle to the center, as well as the radius of the sphere.
- `double = obj.GetCenter ()` - Set/Get the center position of the sphere. Note that this may adjust the direction from the handle to the center, as well as the radius of the sphere.
- `obj.GetCenter (double xyz[3])` - Set/Get the radius of sphere. Default is 0.5. Note that this may modify the position of the handle based on the handle direction.
- `obj.SetRadius (double r)` - Set/Get the radius of sphere. Default is 0.5. Note that this may modify the position of the handle based on the handle direction.
- `double = obj.GetRadius ()` - The handle sits on the surface of the sphere and may be moved around the surface by picking (left mouse) and then moving. The position of the handle can be retrieved, this is useful for positioning cameras and lights. By default, the handle is turned off.
- `obj.SetHandleVisibility (int )` - The handle sits on the surface of the sphere and may be moved around the surface by picking (left mouse) and then moving. The position of the handle can be retrieved, this is useful for positioning cameras and lights. By default, the handle is turned off.
- `int = obj.GetHandleVisibility ()` - The handle sits on the surface of the sphere and may be moved around the surface by picking (left mouse) and then moving. The position of the handle can be retrieved, this is useful for positioning cameras and lights. By default, the handle is turned off.
- `obj.HandleVisibilityOn ()` - The handle sits on the surface of the sphere and may be moved around the surface by picking (left mouse) and then moving. The position of the handle can be retrieved, this is useful for positioning cameras and lights. By default, the handle is turned off.
- `obj.HandleVisibilityOff ()` - The handle sits on the surface of the sphere and may be moved around the surface by picking (left mouse) and then moving. The position of the handle can be retrieved, this is useful for positioning cameras and lights. By default, the handle is turned off.
- `obj.SetHandlePosition (double handle[3])` - Set/Get the position of the handle. Note that this may adjust the radius of the sphere and the handle direction.
- `obj.SetHandlePosition (double x, double y, double z)` - Set/Get the position of the handle. Note that this may adjust the radius of the sphere and the handle direction.
- `double = obj. GetHandlePosition ()` - Set/Get the position of the handle. Note that this may adjust the radius of the sphere and the handle direction.
- `obj.SetHandleDirection (double dir[3])` - Set/Get the direction vector of the handle relative to the center of the sphere. This may affect the position of the handle and the radius of the sphere.
- `obj.SetHandleDirection (double dx, double dy, double dz)` - Set/Get the direction vector of the handle relative to the center of the sphere. This may affect the position of the handle and the radius of the sphere.
- `double = obj. GetHandleDirection ()` - Set/Get the direction vector of the handle relative to the center of the sphere. This may affect the position of the handle and the radius of the sphere.
- `obj.SetHandleText (int )` - Enable/disable a label that displays the location of the handle in spherical coordinates (radius,theta,phi). The two angles, theta and phi, are displayed in degrees. Note that phi is measured from the north pole down towards the equator; and theta is the angle around the north/south axis.

- `int = obj.GetHandleText ()` - Enable/disable a label that displays the location of the handle in spherical coordinates (radius,theta,phi). The two angles, theta and phi, are displayed in degrees. Note that phi is measured from the north pole down towards the equator; and theta is the angle around the north/south axis.
- `obj.HandleTextOn ()` - Enable/disable a label that displays the location of the handle in spherical coordinates (radius,theta,phi). The two angles, theta and phi, are displayed in degrees. Note that phi is measured from the north pole down towards the equator; and theta is the angle around the north/south axis.
- `obj.HandleTextOff ()` - Enable/disable a label that displays the location of the handle in spherical coordinates (radius,theta,phi). The two angles, theta and phi, are displayed in degrees. Note that phi is measured from the north pole down towards the equator; and theta is the angle around the north/south axis.
- `obj.SetRadialLine (int )` - Enable/disable a radial line segment that joins the center of the outer sphere and the handle.
- `int = obj.GetRadialLine ()` - Enable/disable a radial line segment that joins the center of the outer sphere and the handle.
- `obj.RadialLineOn ()` - Enable/disable a radial line segment that joins the center of the outer sphere and the handle.
- `obj.RadialLineOff ()` - Enable/disable a radial line segment that joins the center of the outer sphere and the handle.
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that defines the sphere. The polydata consists of n+1 points, where n is the resolution of the sphere. These point values are guaranteed to be up-to-date when either the InteractionEvent or EndInteraction events are invoked. The user provides the vtkPolyData and the points and polysphere are added to it.
- `obj.GetSphere (vtkSphere sphere)` - Get the spherical implicit function defined by this widget. Note that vtkSphere is a subclass of vtkImplicitFunction, meaning that it can be used by a variety of filters to perform clipping, cutting, and selection of data.
- `vtkProperty = obj.GetSphereProperty ()` - Get the sphere properties. The properties of the sphere when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedSphereProperty ()` - Get the sphere properties. The properties of the sphere when selected and unselected can be manipulated.
- `vtkProperty = obj.GetHandleProperty ()` - Get the handle properties (the little ball on the sphere is the handle). The properties of the handle when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Get the handle properties (the little ball on the sphere is the handle). The properties of the handle when selected and unselected can be manipulated.
- `vtkTextProperty = obj.GetHandleTextProperty ()` - Get the handle text property. This can be used to control the appearance of the handle text.
- `vtkProperty = obj.GetRadialLineProperty ()` - Get the property of the radial line. This can be used to control the appearance of the optional line connecting the center to the handle.
- `obj.SetInteractionState (int state)` - The interaction state may be set from a widget (e.g., vtkSphereWidget2) or other object. This controls how the interaction with the widget proceeds. Normally this method is used as part of a handshaking process with the widget: First ComputeInteractionState() is invoked that returns a state based on geometric considerations (i.e., cursor near a widget feature), then based on events, the widget may modify this further.

- `obj.PlaceWidget (double bounds[6])` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `obj.PlaceWidget (double center[3], double handlePosition[3])` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `obj.StartWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `obj.WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods supporting, and required by, the rendering process.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Methods supporting, and required by, the rendering process.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Methods supporting, and required by, the rendering process.
- `int = obj.RenderOverlay (vtkViewport )` - Methods supporting, and required by, the rendering process.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods supporting, and required by, the rendering process.

42.92 vtkSphereWidget

42.92.1 Usage

This 3D widget defines a sphere that can be interactively placed in a scene.

To use this object, just invoke SetInteractor() with the argument of the method a vtkRenderWindowInteractor. You may also wish to invoke "PlaceWidget()" to initially position the widget. The interactor will act normally until the "i" key (for "interactor") is pressed, at which point the vtkSphereWidget will appear. (See superclass documentation for information about changing this behavior.) Events that occur outside of the widget (i.e., no part of the widget is picked) are propagated to any other registered obsevers (such as the interaction style). Turn off the widget by pressing the "i" key again (or invoke the Off() method).

The vtkSphereWidget has several methods that can be used in conjunction with other VTK objects. The Set/GetThetaResolution() and Set/GetPhiResolution() methods control the number of subdivisions of the sphere in the theta and phi directions; the GetPolyData() method can be used to get the polygonal representation and can be used for things like seeding streamlines. The GetSphere() method returns a sphere implicit function that can be used for cutting and clipping. Typical usage of the widget is to make use of the StartInteractionEvent, InteractionEvent, and EndInteractionEvent events. The InteractionEvent is called on mouse motion; the other two events are called on button down and button up (any mouse button).

Some additional features of this class include the ability to control the properties of the widget. You can set the properties of the selected and unselected representations of the sphere.

To create an instance of class vtkSphereWidget, simply invoke its constructor as follows

```
obj = vtkSphereWidget
```

42.92.2 Methods

The class vtkSphereWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSphereWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSphereWidget = obj.NewInstance ()`
- `vtkSphereWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double bounds[6])` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget ()` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Set the representation of the sphere. Different representations are useful depending on the application. The default is VTK_SPHERE_WIREFRAME.
- `obj.SetRepresentation (int )` - Set the representation of the sphere. Different representations are useful depending on the application. The default is VTK_SPHERE_WIREFRAME.
- `int = obj.GetRepresentationMinValue ()` - Set the representation of the sphere. Different representations are useful depending on the application. The default is VTK_SPHERE_WIREFRAME.
- `int = obj.GetRepresentationMaxValue ()` - Set the representation of the sphere. Different representations are useful depending on the application. The default is VTK_SPHERE_WIREFRAME.
- `int = obj.GetRepresentation ()` - Set the representation of the sphere. Different representations are useful depending on the application. The default is VTK_SPHERE_WIREFRAME.
- `obj.SetRepresentationToOff ()` - Set the representation of the sphere. Different representations are useful depending on the application. The default is VTK_SPHERE_WIREFRAME.
- `obj.SetRepresentationToWireframe ()` - Set the representation of the sphere. Different representations are useful depending on the application. The default is VTK_SPHERE_WIREFRAME.
- `obj.SetRepresentationToSurface ()` - Set/Get the resolution of the sphere in the Theta direction. The default is 16.
- `obj.SetThetaResolution (int r)` - Set/Get the resolution of the sphere in the Theta direction. The default is 16.
- `int = obj.GetThetaResolution ()` - Set/Get the resolution of the sphere in the Phi direction. The default is 8.
- `obj.SetPhiResolution (int r)` - Set/Get the resolution of the sphere in the Phi direction. The default is 8.
- `int = obj.GetPhiResolution ()` - Set/Get the radius of sphere. Default is .5.
- `obj.SetRadius (double r)` - Set/Get the radius of sphere. Default is .5.
- `double = obj.GetRadius ()` - Set/Get the center of the sphere.
- `obj.SetCenter (double x, double y, double z)` - Set/Get the center of the sphere.

- `obj.SetCenter (double x[3])` - Set/Get the center of the sphere.
- `double = obj.GetCenter ()` - Set/Get the center of the sphere.
- `obj.GetCenter (double xyz[3])` - Enable translation and scaling of the widget. By default, the widget can be translated and rotated.
- `obj.SetTranslation (int )` - Enable translation and scaling of the widget. By default, the widget can be translated and rotated.
- `int = obj.GetTranslation ()` - Enable translation and scaling of the widget. By default, the widget can be translated and rotated.
- `obj.TranslationOn ()` - Enable translation and scaling of the widget. By default, the widget can be translated and rotated.
- `obj.TranslationOff ()` - Enable translation and scaling of the widget. By default, the widget can be translated and rotated.
- `obj.SetScale (int )` - Enable translation and scaling of the widget. By default, the widget can be translated and rotated.
- `int = obj.GetScale ()` - Enable translation and scaling of the widget. By default, the widget can be translated and rotated.
- `obj.ScaleOn ()` - Enable translation and scaling of the widget. By default, the widget can be translated and rotated.
- `obj.ScaleOff ()` - Enable translation and scaling of the widget. By default, the widget can be translated and rotated.
- `obj.SetHandleVisibility (int )` - The handle sits on the surface of the sphere and may be moved around the surface by picking (left mouse) and then moving. The position of the handle can be retrieved, this is useful for positioning cameras and lights. By default, the handle is turned off.
- `int = obj.GetHandleVisibility ()` - The handle sits on the surface of the sphere and may be moved around the surface by picking (left mouse) and then moving. The position of the handle can be retrieved, this is useful for positioning cameras and lights. By default, the handle is turned off.
- `obj.HandleVisibilityOn ()` - The handle sits on the surface of the sphere and may be moved around the surface by picking (left mouse) and then moving. The position of the handle can be retrieved, this is useful for positioning cameras and lights. By default, the handle is turned off.
- `obj.HandleVisibilityOff ()` - The handle sits on the surface of the sphere and may be moved around the surface by picking (left mouse) and then moving. The position of the handle can be retrieved, this is useful for positioning cameras and lights. By default, the handle is turned off.
- `obj.SetHandleDirection (double , double , double )` - Set/Get the direction vector of the handle relative to the center of the sphere. The direction of the handle is from the sphere center to the handle position.
- `obj.SetHandleDirection (double  a[3])` - Set/Get the direction vector of the handle relative to the center of the sphere. The direction of the handle is from the sphere center to the handle position.
- `double = obj. GetHandleDirection ()` - Set/Get the direction vector of the handle relative to the center of the sphere. The direction of the handle is from the sphere center to the handle position.
- `double = obj. GetHandlePosition ()` - Get the position of the handle.

- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that defines the sphere. The polydata consists of n+1 points, where n is the resolution of the sphere. These point values are guaranteed to be up-to-date when either the InteractionEvent or EndInteraction events are invoked. The user provides the vtkPolyData and the points and polysphere are added to it.
- `obj.GetSphere (vtkSphere sphere)` - Get the spherical implicit function defined by this widget. Note that vtkSphere is a subclass of vtkImplicitFunction, meaning that it can be used by a variety of filters to perform clipping, cutting, and selection of data.
- `vtkProperty = obj.GetSphereProperty ()` - Get the sphere properties. The properties of the sphere when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedSphereProperty ()` - Get the sphere properties. The properties of the sphere when selected and unselected can be manipulated.
- `vtkProperty = obj.GetHandleProperty ()` - Get the handle properties (the little ball on the sphere is the handle). The properties of the handle when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Get the handle properties (the little ball on the sphere is the handle). The properties of the handle when selected and unselected can be manipulated.

42.93 vtkSphereWidget2

42.93.1 Usage

This 3D widget interacts with a vtkSphereRepresentation class (i.e., it handles the events that drive its corresponding representation). It can be used to position a point on a sphere (for example, to place a light or camera), or to position a sphere in a scene, including translating and scaling the sphere.

A nice feature of vtkSphereWidget2, like any 3D widget, is that it will work in combination with the current interactor style (or any other interactor observer). That is, if vtkSphereWidget2 does not handle an event, then all other registered observers (including the interactor style) have an opportunity to process the event. Otherwise, the vtkSphereWidget2 will terminate the processing of the event that it handles.

To use this widget, you generally pair it with a vtkSphereRepresentation (or a subclass). Variuos options are available in the representation for controlling how the widget appears, and how the widget functions.

.SECTION Event Bindings By default, the widget responds to the following VTK events (i.e., it watches the vtkRenderWindowInteractor for these events):

```
If the handle or sphere are selected:
  LeftButtonPressEvent - select the handle or sphere
  LeftButtonReleaseEvent - release the handle ot sphere
  MouseMoveEvent - move the handle or translate the sphere
In all the cases, independent of what is picked, the widget responds to the
following VTK events:
  MiddleButtonPressEvent - translate the representation
  MiddleButtonReleaseEvent - stop translating the representation
  RightButtonPressEvent - scale the widget's representation
  RightButtonReleaseEvent - stop scaling the representation
  MouseMoveEvent - scale (if right button) or move (if middle button) the widget
```

Note that the event bindings described above can be changed using this class's vtkWidgetEventTranslator. This class translates VTK events into the vtkSphereWidget2's widget events:

```
  vtkWidgetEvent::Select -- some part of the widget has been selected
  vtkWidgetEvent::EndSelect -- the selection process has completed
  vtkWidgetEvent::Scale -- some part of the widget has been selected
```

```
vtkWidgetEvent::EndScale -- the selection process has completed
vtkWidgetEvent::Translate -- some part of the widget has been selected
vtkWidgetEvent::EndTranslate -- the selection process has completed
vtkWidgetEvent::Move -- a request for motion has been invoked
```

In turn, when these widget events are processed, the vtkSphereWidget2 invokes the following VTK events on itself (which observers can listen for):

```
vtkCommand::StartInteractionEvent (on vtkWidgetEvent::Select)
vtkCommand::EndInteractionEvent (on vtkWidgetEvent::EndSelect)
vtkCommand::InteractionEvent (on vtkWidgetEvent::Move)
```

To create an instance of class vtkSphereWidget2, simply invoke its constructor as follows

```
obj = vtkSphereWidget2
```

42.93.2 Methods

The class vtkSphereWidget2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSphereWidget2 class.

- `string = obj.GetClassName ()` - Standard class methods for type information and printing.
- `int = obj.IsA (string name)` - Standard class methods for type information and printing.
- `vtkSphereWidget2 = obj.NewInstance ()` - Standard class methods for type information and printing.
- `vtkSphereWidget2 = obj.SafeDownCast (vtkObject o)` - Standard class methods for type information and printing.
- `obj.SetRepresentation (vtkSphereRepresentation r)` - Control the behavior of the widget (i.e., how it processes events). Translation, and scaling can all be enabled and disabled.
- `obj.SetTranslationEnabled (int )` - Control the behavior of the widget (i.e., how it processes events). Translation, and scaling can all be enabled and disabled.
- `int = obj.GetTranslationEnabled ()` - Control the behavior of the widget (i.e., how it processes events). Translation, and scaling can all be enabled and disabled.
- `obj.TranslationEnabledOn ()` - Control the behavior of the widget (i.e., how it processes events). Translation, and scaling can all be enabled and disabled.
- `obj.TranslationEnabledOff ()` - Control the behavior of the widget (i.e., how it processes events). Translation, and scaling can all be enabled and disabled.
- `obj.SetScalingEnabled (int )` - Control the behavior of the widget (i.e., how it processes events). Translation, and scaling can all be enabled and disabled.
- `int = obj.GetScalingEnabled ()` - Control the behavior of the widget (i.e., how it processes events). Translation, and scaling can all be enabled and disabled.
- `obj.ScalingEnabledOn ()` - Control the behavior of the widget (i.e., how it processes events). Translation, and scaling can all be enabled and disabled.
- `obj.ScalingEnabledOff ()` - Control the behavior of the widget (i.e., how it processes events). Translation, and scaling can all be enabled and disabled.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set. By default, this is an instance of the vtkSphereRepresentation class.

42.94 vtkSplineRepresentation

42.94.1 Usage

vtkSplineRepresentation is a vtkWidgetRepresentation for a spline. This 3D widget defines a spline that can be interactively placed in a scene. The spline has handles, the number of which can be changed, plus it can be picked on the spline itself to translate or rotate it in the scene. This is based on vtkSplineWidget.

To create an instance of class vtkSplineRepresentation, simply invoke its constructor as follows

```
obj = vtkSplineRepresentation
```

42.94.2 Methods

The class vtkSplineRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSplineRepresentation class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSplineRepresentation = obj.NewInstance ()`
- `vtkSplineRepresentation = obj.SafeDownCast (vtkObject o)`
- `obj.SetInteractionState (int )`
- `obj.SetProjectToPlane (int )` - Force the spline widget to be projected onto one of the orthogonal planes. Remember that when the InteractionState changes, a ModifiedEvent is invoked. This can be used to snap the spline to the plane if it is orginally not aligned. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively and 3 for arbitrary oblique planes when the widget is tied to a vtkPlaneSource.
- `int = obj.GetProjectToPlane ()` - Force the spline widget to be projected onto one of the orthogonal planes. Remember that when the InteractionState changes, a ModifiedEvent is invoked. This can be used to snap the spline to the plane if it is orginally not aligned. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively and 3 for arbitrary oblique planes when the widget is tied to a vtkPlaneSource.
- `obj.ProjectToPlaneOn ()` - Force the spline widget to be projected onto one of the orthogonal planes. Remember that when the InteractionState changes, a ModifiedEvent is invoked. This can be used to snap the spline to the plane if it is orginally not aligned. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively and 3 for arbitrary oblique planes when the widget is tied to a vtkPlaneSource.
- `obj.ProjectToPlaneOff ()` - Force the spline widget to be projected onto one of the orthogonal planes. Remember that when the InteractionState changes, a ModifiedEvent is invoked. This can be used to snap the spline to the plane if it is orginally not aligned. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively and 3 for arbitrary oblique planes when the widget is tied to a vtkPlaneSource.
- `obj.SetPlaneSource (vtkPlaneSource plane)` - Set up a reference to a vtkPlaneSource that could be from another widget object, e.g. a vtkPolyDataSourceWidget.
- `obj.SetProjectionNormal (int )`
- `int = obj.GetProjectionNormalMinValue ()`
- `int = obj.GetProjectionNormalMaxValue ()`

- `int = obj.GetProjectionNormal ()`
- `obj.SetProjectionNormalToXAxes ()`
- `obj.SetProjectionNormalToYAxes ()`
- `obj.SetProjectionNormalToZAxes ()`
- `obj.SetProjectionNormalToOblique ()` - Set the position of spline handles and points in terms of a plane's position. i.e., if ProjectionNormal is 0, all of the x-coordinate values of the points are set to position. Any value can be passed (and is ignored) to update the spline points when Projection normal is set to 3 for arbritrary plane orientations.
- `obj.SetProjectionPosition (double position)` - Set the position of spline handles and points in terms of a plane's position. i.e., if ProjectionNormal is 0, all of the x-coordinate values of the points are set to position. Any value can be passed (and is ignored) to update the spline points when Projection normal is set to 3 for arbritrary plane orientations.
- `double = obj.GetProjectionPosition ()` - Set the position of spline handles and points in terms of a plane's position. i.e., if ProjectionNormal is 0, all of the x-coordinate values of the points are set to position. Any value can be passed (and is ignored) to update the spline points when Projection normal is set to 3 for arbritrary plane orientations.
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that defines the spline. The polydata consists of points and line segments numbering Resolution + 1 and Resoltuion, respectively. Points are guaranteed to be up-to-date when either the InteractionEvent or EndInteraction events are invoked. The user provides the vtkPolyData and the points and polyline are added to it.
- `vtkProperty = obj.GetHandleProperty ()` - Set/Get the handle properties (the spheres are the handles). The properties of the handles when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Set/Get the handle properties (the spheres are the handles). The properties of the handles when selected and unselected can be manipulated.
- `vtkProperty = obj.GetLineProperty ()` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedLineProperty ()` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `obj.SetNumberOfHandles (int npts)` - Set/Get the number of handles for this widget.
- `int = obj.GetNumberOfHandles ()` - Set/Get the number of handles for this widget.
- `obj.SetResolution (int resolution)` - Set/Get the number of line segments representing the spline for this widget.
- `int = obj.GetResolution ()` - Set/Get the number of line segments representing the spline for this widget.
- `obj.SetParametricSpline (vtkParametricSpline )` - Set the parametric spline object. Through vtkParametricSpline's API, the user can supply and configure one of currently two types of spline: vtkCardinalSpline, vtkKochanekSpline. The widget controls the open or closed configuration of the spline. WARNING: The widget does not enforce internal consistency so that all three are of the same type.
- `vtkParametricSpline = obj.GetParametricSpline ()` - Set the parametric spline object. Through vtkParametricSpline's API, the user can supply and configure one of currently two types of spline: vtkCardinalSpline, vtkKochanekSpline. The widget controls the open or closed configuration of the spline. WARNING: The widget does not enforce internal consistency so that all three are of the same type.

- `obj.SetHandlePosition (int handle, double x, double y, double z)` - Set/Get the position of the spline handles. Call GetNumberOfHandles to determine the valid range of handle indices.
- `obj.SetHandlePosition (int handle, double xyz[3])` - Set/Get the position of the spline handles. Call GetNumberOfHandles to determine the valid range of handle indices.
- `obj.GetHandlePosition (int handle, double xyz[3])` - Set/Get the position of the spline handles. Call GetNumberOfHandles to determine the valid range of handle indices.
- `vtkDoubleArray = obj.GetHandlePositions ()` - Set/Get the position of the spline handles. Call GetNumberOfHandles to determine the valid range of handle indices.
- `obj.SetClosed (int closed)` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous. A minimum of 3 handles are required to form a closed loop. This method enforces consistency with user supplied subclasses of vtkSpline.
- `int = obj.GetClosed ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous. A minimum of 3 handles are required to form a closed loop. This method enforces consistency with user supplied subclasses of vtkSpline.
- `obj.ClosedOn ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous. A minimum of 3 handles are required to form a closed loop. This method enforces consistency with user supplied subclasses of vtkSpline.
- `obj.ClosedOff ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous. A minimum of 3 handles are required to form a closed loop. This method enforces consistency with user supplied subclasses of vtkSpline.
- `int = obj.IsClosed ()` - Convenience method to determine whether the spline is closed in a geometric sense. The widget may be set "closed" but still be geometrically open (e.g., a straight line).
- `double = obj.GetSummedLength ()` - Get the approximate vs. the true arc length of the spline. Calculated as the summed lengths of the individual straight line segments. Use SetResolution to control the accuracy.
- `obj.InitializeHandles (vtkPoints points)` - Convenience method to allocate and set the handles from a vtkPoints instance. If the first and last points are the same, the spline sets Closed to the on InteractionState and disregards the last point, otherwise Closed remains unchanged.
- `obj.BuildRepresentation ()` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `obj.StartWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `obj.WidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.
- `obj.EndWidgetInteraction (double e[2])` - These are methods that satisfy vtkWidgetRepresentation's API. Note that a version of place widget is available where the center and handle position are specified.

- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods supporting, and required by, the rendering process.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Methods supporting, and required by, the rendering process.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Methods supporting, and required by, the rendering process.
- `int = obj.RenderOverlay (vtkViewport )` - Methods supporting, and required by, the rendering process.
- `int = obj.HasTranslucentPolygonalGeometry ()` - Methods supporting, and required by, the rendering process.
- `obj.SetLineColor (double r, double g, double b)` - Convenience method to set the line color. Ideally one should use GetLineProperty()-¿SetColor().

42.95 vtkSplineWidget

42.95.1 Usage

This 3D widget defines a spline that can be interactively placed in a scene. The spline has handles, the number of which can be changed, plus it can be picked on the spline itself to translate or rotate it in the scene. A nice feature of the object is that the vtkSplineWidget, like any 3D widget, will work with the current interactor style. That is, if vtkSplineWidget does not handle an event, then all other registered observers (including the interactor style) have an opportunity to process the event. Otherwise, the vtkSplineWidget will terminate the processing of the event that it handles.

To use this object, just invoke SetInteractor() with the argument of the method a vtkRenderWindowInteractor. You may also wish to invoke "PlaceWidget()" to initially position the widget. The interactor will act normally until the "i" key (for "interactor") is pressed, at which point the vtkSplineWidget will appear. (See superclass documentation for information about changing this behavior.) Events that occur outside of the widget (i.e., no part of the widget is picked) are propagated to any other registered obsevers (such as the interaction style). Turn off the widget by pressing the "i" key again (or invoke the Off() method).

The button actions and key modifiers are as follows for controlling the widget: 1) left button down on and drag one of the spherical handles to change the shape of the spline: the handles act as "control points". 2) left button or middle button down on a line segment forming the spline allows uniform translation of the widget. 3) ctrl + middle button down on the widget enables spinning of the widget about its center. 4) right button down on the widget enables scaling of the widget. By moving the mouse "up" the render window the spline will be made bigger; by moving "down" the render window the widget will be made smaller. 5) ctrl key + right button down on any handle will erase it providing there will be two or more points remaining to form a spline. 6) shift key + right button down on any line segment will insert a handle onto the spline at the cursor position.

The vtkSplineWidget has several methods that can be used in conjunction with other VTK objects. The Set/GetResolution() methods control the number of subdivisions of the spline; the GetPolyData() method can be used to get the polygonal representation and can be used for things like seeding streamlines or probing other data sets. Typical usage of the widget is to make use of the StartInteractionEvent, InteractionEvent, and EndInteractionEvent events. The InteractionEvent is called on mouse motion; the other two events are called on button down and button up (either left or right button).

Some additional features of this class include the ability to control the properties of the widget. You can set the properties of the selected and unselected representations of the spline. For example, you can set the property for the handles and spline. In addition there are methods to constrain the spline so that it is aligned with a plane. Note that a simple ruler widget can be derived by setting the resolution to 1, the number of handles to 2, and calling the GetSummedLength method!

To create an instance of class vtkSplineWidget, simply invoke its constructor as follows

```
obj = vtkSplineWidget
```

42.95.2 Methods

The class vtkSplineWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSplineWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSplineWidget = obj.NewInstance ()`
- `vtkSplineWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetEnabled (int )` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double bounds[6])` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget ()` - Methods that satisfy the superclass' API.
- `obj.PlaceWidget (double xmin, double xmax, double ymin, double ymax, double zmin, double zmax)` - Force the spline widget to be projected onto one of the orthogonal planes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the spline to the plane if it is orginally not aligned. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively and 3 for arbitrary oblique planes when the widget is tied to a vtkPlaneSource.
- `obj.SetProjectToPlane (int )` - Force the spline widget to be projected onto one of the orthogonal planes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the spline to the plane if it is orginally not aligned. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively and 3 for arbitrary oblique planes when the widget is tied to a vtkPlaneSource.
- `int = obj.GetProjectToPlane ()` - Force the spline widget to be projected onto one of the orthogonal planes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the spline to the plane if it is orginally not aligned. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively and 3 for arbitrary oblique planes when the widget is tied to a vtkPlaneSource.
- `obj.ProjectToPlaneOn ()` - Force the spline widget to be projected onto one of the orthogonal planes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the spline to the plane if it is orginally not aligned. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively and 3 for arbitrary oblique planes when the widget is tied to a vtkPlaneSource.
- `obj.ProjectToPlaneOff ()` - Force the spline widget to be projected onto one of the orthogonal planes. Remember that when the state changes, a ModifiedEvent is invoked. This can be used to snap the spline to the plane if it is orginally not aligned. The normal in SetProjectionNormal is 0,1,2 for YZ,XZ,XY planes respectively and 3 for arbitrary oblique planes when the widget is tied to a vtkPlaneSource.
- `obj.SetPlaneSource (vtkPlaneSource plane)` - Set up a reference to a vtkPlaneSource that could be from another widget object, e.g. a vtkPolyDataSourceWidget.
- `obj.SetProjectionNormal (int )`
- `int = obj.GetProjectionNormalMinValue ()`
- `int = obj.GetProjectionNormalMaxValue ()`
- `int = obj.GetProjectionNormal ()`

- `obj.SetProjectionNormalToXAxes ()`
- `obj.SetProjectionNormalToYAxes ()`
- `obj.SetProjectionNormalToZAxes ()`
- `obj.SetProjectionNormalToOblique ()` - Set the position of spline handles and points in terms of a plane's position. i.e., if ProjectionNormal is 0, all of the x-coordinate values of the points are set to position. Any value can be passed (and is ignored) to update the spline points when Projection normal is set to 3 for arbritrary plane orientations.
- `obj.SetProjectionPosition (double position)` - Set the position of spline handles and points in terms of a plane's position. i.e., if ProjectionNormal is 0, all of the x-coordinate values of the points are set to position. Any value can be passed (and is ignored) to update the spline points when Projection normal is set to 3 for arbritrary plane orientations.
- `double = obj.GetProjectionPosition ()` - Set the position of spline handles and points in terms of a plane's position. i.e., if ProjectionNormal is 0, all of the x-coordinate values of the points are set to position. Any value can be passed (and is ignored) to update the spline points when Projection normal is set to 3 for arbritrary plane orientations.
- `obj.GetPolyData (vtkPolyData pd)` - Grab the polydata (including points) that defines the spline. The polydata consists of points and line segments numbering Resolution + 1 and Resoltuion, respectively. Points are guaranteed to be up-to-date when either the InteractionEvent or EndInteraction events are invoked. The user provides the vtkPolyData and the points and polyline are added to it.
- `obj.SetHandleProperty (vtkProperty )` - Set/Get the handle properties (the spheres are the handles). The properties of the handles when selected and unselected can be manipulated.
- `vtkProperty = obj.GetHandleProperty ()` - Set/Get the handle properties (the spheres are the handles). The properties of the handles when selected and unselected can be manipulated.
- `obj.SetSelectedHandleProperty (vtkProperty )` - Set/Get the handle properties (the spheres are the handles). The properties of the handles when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedHandleProperty ()` - Set/Get the handle properties (the spheres are the handles). The properties of the handles when selected and unselected can be manipulated.
- `obj.SetLineProperty (vtkProperty )` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `vtkProperty = obj.GetLineProperty ()` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `obj.SetSelectedLineProperty (vtkProperty )` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `vtkProperty = obj.GetSelectedLineProperty ()` - Set/Get the line properties. The properties of the line when selected and unselected can be manipulated.
- `obj.SetNumberOfHandles (int npts)` - Set/Get the number of handles for this widget.
- `int = obj.GetNumberOfHandles ()` - Set/Get the number of handles for this widget.
- `obj.SetResolution (int resolution)` - Set/Get the number of line segments representing the spline for this widget.
- `int = obj.GetResolution ()` - Set/Get the number of line segments representing the spline for this widget.

- `obj.SetParametricSpline (vtkParametricSpline )` - Set the parametric spline object. Through vtkParametricSpline's API, the user can supply and configure one of currently two types of spline: vtkCardinalSpline, vtkKochanekSpline. The widget controls the open or closed configuration of the spline. WARNING: The widget does not enforce internal consistency so that all three are of the same type.

- `vtkParametricSpline = obj.GetParametricSpline ()` - Set the parametric spline object. Through vtkParametricSpline's API, the user can supply and configure one of currently two types of spline: vtkCardinalSpline, vtkKochanekSpline. The widget controls the open or closed configuration of the spline. WARNING: The widget does not enforce internal consistency so that all three are of the same type.

- `obj.SetHandlePosition (int handle, double x, double y, double z)` - Set/Get the position of the spline handles. Call GetNumberOfHandles to determine the valid range of handle indices.

- `obj.SetHandlePosition (int handle, double xyz[3])` - Set/Get the position of the spline handles. Call GetNumberOfHandles to determine the valid range of handle indices.

- `obj.GetHandlePosition (int handle, double xyz[3])` - Set/Get the position of the spline handles. Call GetNumberOfHandles to determine the valid range of handle indices.

- `double = obj.GetHandlePosition (int handle)` - Set/Get the position of the spline handles. Call GetNumberOfHandles to determine the valid range of handle indices.

- `obj.SetClosed (int closed)` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous. A minimum of 3 handles are required to form a closed loop. This method enforces consistency with user supplied subclasses of vtkSpline.

- `int = obj.GetClosed ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous. A minimum of 3 handles are required to form a closed loop. This method enforces consistency with user supplied subclasses of vtkSpline.

- `obj.ClosedOn ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous. A minimum of 3 handles are required to form a closed loop. This method enforces consistency with user supplied subclasses of vtkSpline.

- `obj.ClosedOff ()` - Control whether the spline is open or closed. A closed spline forms a continuous loop: the first and last points are the same, and derivatives are continuous. A minimum of 3 handles are required to form a closed loop. This method enforces consistency with user supplied subclasses of vtkSpline.

- `int = obj.IsClosed ()` - Convenience method to determine whether the spline is closed in a geometric sense. The widget may be set "closed" but still be geometrically open (e.g., a straight line).

- `double = obj.GetSummedLength ()` - Get the approximate vs. the true arc length of the spline. Calculated as the summed lengths of the individual straight line segments. Use SetResolution to control the accuracy.

- `obj.InitializeHandles (vtkPoints points)` - Convenience method to allocate and set the handles from a vtkPoints instance. If the first and last points are the same, the spline sets Closed to the on state and disregards the last point, otherwise Closed remains unchanged.

42.96 vtkSplineWidget2

42.96.1 Usage

vtkSplineWidget2 is the vtkAbstractWidget subclass for vtkSplineRepresentation which manages the interactions with vtkSplineRepresentation. This is based on vtkSplineWidget.

To create an instance of class vtkSplineWidget2, simply invoke its constructor as follows

```
obj = vtkSplineWidget2
```

42.96.2 Methods

The class vtkSplineWidget2 has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkSplineWidget2 class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkSplineWidget2 = obj.NewInstance ()`
- `vtkSplineWidget2 = obj.SafeDownCast (vtkObject o)`
- `obj.SetRepresentation (vtkSplineRepresentation r)` - Create the default widget representation if one is not set. By default, this is an instance of the vtkSplineRepresentation class.
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set. By default, this is an instance of the vtkSplineRepresentation class.

42.97 vtkTensorProbeRepresentation

42.97.1 Usage

The class serves as an abstract geometrical representation for the vtkTensorProbeWidget. It is left to the concrete implementation to render the tensors as it desires. For instance, vtkEllipsoidTensorProbeRepresentation renders the tensors as ellipsoids.

To create an instance of class vtkTensorProbeRepresentation, simply invoke its constructor as follows

```
obj = vtkTensorProbeRepresentation
```

42.97.2 Methods

The class vtkTensorProbeRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTensorProbeRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkTensorProbeRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkTensorProbeRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.

- `obj.BuildRepresentation ()` - See vtkWidgetRepresentation for details.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - See vtkWidgetRepresentation for details.
- `obj.SetProbePosition (double , double , double )` - Set the position of the Tensor probe.
- `obj.SetProbePosition (double a[3])` - Set the position of the Tensor probe.
- `double = obj. GetProbePosition ()` - Set the position of the Tensor probe.
- `obj.SetProbeCellId (vtkIdType )` - Set the position of the Tensor probe.
- `vtkIdType = obj.GetProbeCellId ()` - Set the position of the Tensor probe.
- `obj.SetTrajectory (vtkPolyData )` - Set the trajectory that we are trying to probe tensors on
- `obj.Initialize ()` - Set the probe position to a reasonable location on the trajectory.
- `int = obj.SelectProbe (int pos[2])` - This method is invoked by the widget during user interaction. Can we pick the tensor glyph at the current cursor pos
- `int = obj.Move (double motionVector[2])` - INTERNAL - Do not use This method is invoked by the widget during user interaction. Move probe based on the position and the motion vector.
- `obj.GetActors (vtkPropCollection )` - See vtkProp for details.
- `obj.ReleaseGraphicsResources (vtkWindow )` - See vtkProp for details.

42.98 vtkTensorProbeWidget

42.98.1 Usage

The class is used to probe tensors on a trajectory. The representation (vtkTensorProbeRepresentation) is free to choose its own method of rendering the tensors. For instance vtkEllipsoidTensorProbeRepresentation renders the tensors as ellipsoids. The interactions of the widget are controlled by the left mouse button. A left click on the tensor selects it. It can dragged around the trajectory to probe the tensors on it.

For instance dragging the ellipsoid around with vtkEllipsoidTensorProbeRepresentation will manifest itself with the ellipsoid shape changing as needed along the trajectory.

To create an instance of class vtkTensorProbeWidget, simply invoke its constructor as follows

```
obj = vtkTensorProbeWidget
```

42.98.2 Methods

The class vtkTensorProbeWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTensorProbeWidget class.

- `string = obj.GetClassName ()` - Standard VTK class macros.
- `int = obj.IsA (string name)` - Standard VTK class macros.
- `vtkTensorProbeWidget = obj.NewInstance ()` - Standard VTK class macros.
- `vtkTensorProbeWidget = obj.SafeDownCast (vtkObject o)` - Standard VTK class macros.
- `obj.SetRepresentation (vtkTensorProbeRepresentation r)` - See vtkWidgetRepresentation for details.
- `obj.CreateDefaultRepresentation ()` - See vtkWidgetRepresentation for details.

42.99 vtkTerrainContourLineInterpolator

42.99.1 Usage

vtkTerrainContourLineInterpolator interpolates nodes on height field data. The class is meant to be used in conjunciton with a vtkContourWidget, enabling you to draw paths on terrain data. The class internally uses a vtkProjectedTerrainPath. Users can set kind of interpolation desired between two node points by setting the modes of the this filter. For instance:

```
contourRepresentation->SetLineInterpolator(interpolator);
interpolator->SetImageData( demDataFile );
interpolator->GetProjector()->SetProjectionModeToHug();
interpolator->SetHeightOffset(25.0);
```

You are required to set the ImageData to this class as the height-field image.
To create an instance of class vtkTerrainContourLineInterpolator, simply invoke its constructor as follows

```
obj = vtkTerrainContourLineInterpolator
```

42.99.2 Methods

The class vtkTerrainContourLineInterpolator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTerrainContourLineInterpolator class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkTerrainContourLineInterpolator = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkTerrainContourLineInterpolator = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `int = obj.InterpolateLine (vtkRenderer ren, vtkContourRepresentation rep, int idx1, int idx2)` - Interpolate to create lines between contour nodes idx1 and idx2. Depending on the projection mode, the interpolated line may either hug the terrain, just connect the two points with a straight line or a non-occluded interpolation. Used internally by vtkContourRepresentation.
- `int = obj.UpdateNode (vtkRenderer , vtkContourRepresentation , double , int )` - The interpolator is given a chance to update the node. Used internally by vtkContourRepresentation Returns 0 if the node (world position) is unchanged.
- `obj.SetImageData (vtkImageData )` - Set the height field data. The height field data is a 2D image. The scalars in the image represent the height field. This must be set.
- `vtkImageData = obj.GetImageData ()` - Set the height field data. The height field data is a 2D image. The scalars in the image represent the height field. This must be set.
- `vtkProjectedTerrainPath = obj.GetProjector ()` - Get the vtkProjectedTerrainPath operator used to project the terrain onto the data. This operator has several modes, See the documentation of vtkProjectedTerrainPath. The default mode is to hug the terrain data at 0 height offset.

42.100 vtkTerrainDataPointPlacer

42.100.1 Usage

vtkTerrainDataPointPlacer dictates the placement of points on height field data. The class takes as input the list of props that represent the terrain in a rendered scene. A height offset can be specified to dicatate the placement of points at a certain height above the surface.

.SECTION Usage A typical usage of this class is as follows:

```
pointPlacer->AddProp(demActor);    // the actor(s) containing the terrain.
rep->SetPointPlacer(pointPlacer);
pointPlacer->SetHeightOffset( 100 );
```

To create an instance of class vtkTerrainDataPointPlacer, simply invoke its constructor as follows

```
obj = vtkTerrainDataPointPlacer
```

42.100.2 Methods

The class vtkTerrainDataPointPlacer has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTerrainDataPointPlacer class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkTerrainDataPointPlacer = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkTerrainDataPointPlacer = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.AddProp (vtkProp )`
- `obj.RemoveAllProps ()`
- `obj.SetHeightOffset (double )` - This is the height above (or below) the terrain that the dictated point should be placed. Positive values indicate distances above the terrain; negative values indicate distances below the terrain. The default is 0.0.
- `double = obj.GetHeightOffset ()` - This is the height above (or below) the terrain that the dictated point should be placed. Positive values indicate distances above the terrain; negative values indicate distances below the terrain. The default is 0.0.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double worldPos[3], double w` - Given a renderer and a display position in pixel coordinates, compute the world position and orientation where this point will be placed. This method is typically used by the representation to place the point initially. For the Terrain point placer this computes world points that lie at the specified height above the terrain.
- `int = obj.ComputeWorldPosition (vtkRenderer ren, double displayPos[2], double refWorldPos[3], doubl` - Given a renderer, a display position, and a reference world position, compute the new world position and orientation of this point. This method is typically used by the representation to move the point.
- `int = obj.ValidateWorldPosition (double worldPos[3])` - Given a world position check the validity of this position according to the constraints of the placer
- `int = obj.ValidateDisplayPosition (vtkRenderer , double displayPos[2])` - Given a display position, check the validity of this position.

- `int = obj.ValidateWorldPosition (double worldPos[3], double worldOrient[9])` - Given a world position and a world orientation, validate it according to the constraints of the placer.
- `vtkPropPicker = obj.GetPropPicker ()` - Get the Prop picker.

42.101 vtkTextRepresentation

42.101.1 Usage

This class represents text for a vtkTextWidget. This class provides support for interactively placing text on the 2D overlay plane. The text is defined by an instance of vtkTextActor.

To create an instance of class vtkTextRepresentation, simply invoke its constructor as follows

```
obj = vtkTextRepresentation
```

42.101.2 Methods

The class vtkTextRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextRepresentation class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkTextRepresentation = obj.NewInstance ()` - Standard VTK methods.
- `vtkTextRepresentation = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `obj.SetTextActor (vtkTextActor textActor)` - Specify the vtkTextActor to manage. If not specified, then one is automatically created.
- `vtkTextActor = obj.GetTextActor ()` - Specify the vtkTextActor to manage. If not specified, then one is automatically created.
- `obj.SetText (string text)` - Get/Set the text string display by this representation.
- `string = obj.GetText ()` - Get/Set the text string display by this representation.
- `obj.BuildRepresentation ()` - Satisfy the superclasses API.
- `obj.GetSize (double size[2])` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.GetActors2D (vtkPropCollection )` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.ReleaseGraphicsResources (vtkWindow )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOverlay (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderOpaqueGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.
- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - These methods are necessary to make this representation behave as a vtkProp.

- `int = obj.HasTranslucentPolygonalGeometry ()` - These methods are necessary to make this representation behave as a vtkProp.
- `obj.SetWindowLocation (int enumLocation)` - Set the text position, by enumeration (AnyLocation = 0, LowerLeftCorner, LowerRightCorner, LowerCenter, UpperLeftCorner, UpperRightCorner, UpperCenter) related to the render window
- `int = obj.GetWindowLocation ()` - Set the text position, by enumeration (AnyLocation = 0, LowerLeftCorner, LowerRightCorner, LowerCenter, UpperLeftCorner, UpperRightCorner, UpperCenter) related to the render window
- `obj.SetPosition (double x, double y)` - Set the text position, by overiding the same function of vtkBorderRepresentation so that the Modified() will be called.
- `obj.SetPosition (double pos[2])` - Set the text position, by overiding the same function of vtkBorderRepresentation so that the Modified() will be called.

42.102 vtkTextWidget

42.102.1 Usage

This class provides support for interactively placing text on the 2D overlay plane. The text is defined by an instance of vtkTextActor. It uses the event bindings of its superclass (vtkBorderWidget). In addition, when the text is selected, the widget emits a WidgetActivateEvent that observers can watch for. This is useful for opening GUI dialogues to adjust font characteristics, etc. (Please see the superclass for a description of event bindings.)

To create an instance of class vtkTextWidget, simply invoke its constructor as follows

```
obj = vtkTextWidget
```

42.102.2 Methods

The class vtkTextWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkTextWidget class.

- `string = obj.GetClassName ()` - Standard VTK methods.
- `int = obj.IsA (string name)` - Standard VTK methods.
- `vtkTextWidget = obj.NewInstance ()` - Standard VTK methods.
- `vtkTextWidget = obj.SafeDownCast (vtkObject o)` - Standard VTK methods.
- `obj.SetRepresentation (vtkTextRepresentation r)` - Specify a vtkTextActor to manage. This is a convenient, alternative method to specify the representation for the widget (i.e., used instead of SetRepresentation()). It internally creates a vtkTextRepresentation and then invokes vtkTextRepresentation::SetTextActor().
- `obj.SetTextActor (vtkTextActor textActor)` - Specify a vtkTextActor to manage. This is a convenient, alternative method to specify the representation for the widget (i.e., used instead of SetRepresentation()). It internally creates a vtkTextRepresentation and then invokes vtkTextRepresentation::SetTextActor().
- `vtkTextActor = obj.GetTextActor ()` - Specify a vtkTextActor to manage. This is a convenient, alternative method to specify the representation for the widget (i.e., used instead of SetRepresentation()). It internally creates a vtkTextRepresentation and then invokes vtkTextRepresentation::SetTextActor().
- `obj.CreateDefaultRepresentation ()` - Create the default widget representation if one is not set.

42.103 vtkWidgetCallbackMapper

42.103.1 Usage

vtkWidgetCallbackMapper maps widget events (defined in vtkWidgetEvent.h) into static class methods, and provides facilities to invoke the methods. This class is templated and meant to be used as an internal helper class by the widget classes. The class works in combination with the class vtkWidgetEventTranslator, which translates VTK events into widget events.

To create an instance of class vtkWidgetCallbackMapper, simply invoke its constructor as follows

```
obj = vtkWidgetCallbackMapper
```

42.103.2 Methods

The class vtkWidgetCallbackMapper has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWidgetCallbackMapper class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkWidgetCallbackMapper = obj.NewInstance ()` - Standard macros.
- `vtkWidgetCallbackMapper = obj.SafeDownCast (vtkObject o)` - Standard macros.
- `obj.SetEventTranslator (vtkWidgetEventTranslator t)` - Specify the vtkWidgetEventTranslator to coordinate with.
- `vtkWidgetEventTranslator = obj.GetEventTranslator ()` - Specify the vtkWidgetEventTranslator to coordinate with.
- `obj.InvokeCallback (long widgetEvent)` - This method invokes the callback given a widget event. A non-zero value is returned if the listed event is registered.

42.104 vtkWidgetEvent

42.104.1 Usage

vtkWidgetEvent defines widget events. These events are processed by subclasses of vtkInteractorObserver.

To create an instance of class vtkWidgetEvent, simply invoke its constructor as follows

```
obj = vtkWidgetEvent
```

42.104.2 Methods

The class vtkWidgetEvent has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWidgetEvent class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkWidgetEvent = obj.NewInstance ()` - Standard macros.
- `vtkWidgetEvent = obj.SafeDownCast (vtkObject o)` - Standard macros.

42.105 vtkWidgetEventTranslator

42.105.1 Usage

vtkWidgetEventTranslator maps VTK events (defined on vtkCommand) into widget events (defined in vtkWidgetEvent.h). This class is typically used in combination with vtkWidgetCallbackMapper, which is responsible for translating widget events into method callbacks, and then invoking the callbacks.

This class can be used to define different mappings of VTK events into the widget events. Thus widgets can be reconfigured to use different event bindings.

To create an instance of class vtkWidgetEventTranslator, simply invoke its constructor as follows

```
obj = vtkWidgetEventTranslator
```

42.105.2 Methods

The class vtkWidgetEventTranslator has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWidgetEventTranslator class.

- `string = obj.GetClassName ()` - Standard macros.
- `int = obj.IsA (string name)` - Standard macros.
- `vtkWidgetEventTranslator = obj.NewInstance ()` - Standard macros.
- `vtkWidgetEventTranslator = obj.SafeDownCast (vtkObject o)` - Standard macros.
- `obj.SetTranslation (long VTKEvent, long widgetEvent)` - Use these methods to create the translation from a VTK event to a widget event. Specifying vtkWidgetEvent::NoEvent or an empty string for the (toEvent) erases the mapping for the event.
- `obj.SetTranslation (string VTKEvent, string widgetEvent)` - Use these methods to create the translation from a VTK event to a widget event. Specifying vtkWidgetEvent::NoEvent or an empty string for the (toEvent) erases the mapping for the event.
- `obj.SetTranslation (long VTKEvent, int modifier, char keyCode, int repeatCount, string keySym, long` - Use these methods to create the translation from a VTK event to a widget event. Specifying vtkWidgetEvent::NoEvent or an empty string for the (toEvent) erases the mapping for the event.
- `obj.SetTranslation (vtkEvent VTKevent, long widgetEvent)` - Use these methods to create the translation from a VTK event to a widget event. Specifying vtkWidgetEvent::NoEvent or an empty string for the (toEvent) erases the mapping for the event.
- `long = obj.GetTranslation (long VTKEvent)` - Translate a VTK event into a widget event. If no event mapping is found, then the methods return vtkWidgetEvent::NoEvent or a NULL string.
- `string = obj.GetTranslation (string VTKEvent)` - Translate a VTK event into a widget event. If no event mapping is found, then the methods return vtkWidgetEvent::NoEvent or a NULL string.
- `long = obj.GetTranslation (long VTKEvent, int modifier, char keyCode, int repeatCount, string keySy` - Translate a VTK event into a widget event. If no event mapping is found, then the methods return vtkWidgetEvent::NoEvent or a NULL string.
- `long = obj.GetTranslation (vtkEvent VTKEvent)` - Translate a VTK event into a widget event. If no event mapping is found, then the methods return vtkWidgetEvent::NoEvent or a NULL string.
- `int = obj.RemoveTranslation (long VTKEvent, int modifier, char keyCode, int repeatCount, string key` - Remove translations for a binding. Returns the number of translations removed.

- `int = obj.RemoveTranslation (vtkEvent e)` - Remove translations for a binding. Returns the number of translations removed.
- `int = obj.RemoveTranslation (long VTKEvent)` - Remove translations for a binding. Returns the number of translations removed.
- `obj.ClearEvents ()` - Clear all events from the translator (i.e., no events will be translated).

42.106 vtkWidgetRepresentation

42.106.1 Usage

This class is used to define the API for, and partially implement, a representation for different types of widgets. Note that the widget representation (i.e., subclasses of vtkWidgetRepresentation) are a type of vtkProp; meaning that they can be associated with a vtkRenderer end embedded in a scene like any other vtkActor. However, vtkWidgetRepresentation also defines an API that enables it to be paired with a subclass vtkAbstractWidget, meaning that it can be driven by a widget, serving to represent the widget as the widget responds to registered events.

The API defined here should be regarded as a guideline for implementing widgets and widget representations. Widget behavior is complex, as is the way the representation responds to the registered widget events, so the API may vary from widget to widget to reflect this complexity.

To create an instance of class vtkWidgetRepresentation, simply invoke its constructor as follows

```
obj = vtkWidgetRepresentation
```

42.106.2 Methods

The class vtkWidgetRepresentation has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWidgetRepresentation class.

- `string = obj.GetClassName ()` - Standard methods for instances of this class.
- `int = obj.IsA (string name)` - Standard methods for instances of this class.
- `vtkWidgetRepresentation = obj.NewInstance ()` - Standard methods for instances of this class.
- `vtkWidgetRepresentation = obj.SafeDownCast (vtkObject o)` - Standard methods for instances of this class.
- `obj.SetRenderer (vtkRenderer ren)` - Subclasses of vtkWidgetRepresentation must implement these methods. This is considered the minimum API for a widget representation.

  ```
  SetRenderer() - the renderer in which the widget is to appear must be set.
  BuildRepresentation() - update the geometry of the widget based on its
                          current state.
  ```

  WARNING: The renderer is NOT reference counted by the representation, in order to avoid reference loops. Be sure that the representation lifetime does not extend beyond the renderer lifetime.

- `vtkRenderer = obj.GetRenderer ()` - Subclasses of vtkWidgetRepresentation must implement these methods. This is considered the minimum API for a widget representation.

  ```
  SetRenderer() - the renderer in which the widget is to appear must be set.
  BuildRepresentation() - update the geometry of the widget based on its
                          current state.
  ```

WARNING: The renderer is NOT reference counted by the representation, in order to avoid reference loops. Be sure that the representation lifetime does not extend beyond the renderer lifetime.

- `obj.BuildRepresentation ()` - Subclasses of vtkWidgetRepresentation must implement these methods. This is considered the minimum API for a widget representation.

```
SetRenderer() - the renderer in which the widget is to appear must be set.
BuildRepresentation() - update the geometry of the widget based on its
                        current state.
```

WARNING: The renderer is NOT reference counted by the representation, in order to avoid reference loops. Be sure that the representation lifetime does not extend beyond the renderer lifetime.

- `obj.PlaceWidget (double )` - The following is a suggested API for widget representations. These methods define the communication between the widget and its representation. These methods are only suggestions because widgets take on so many different forms that a universal API is not deemed practical. However, these methods should be implemented when possible to insure that the VTK widget hierarchy remains self-consistent.

```
PlaceWidget() - given a bounding box (xmin,xmax,ymin,ymax,zmin,zmax), place
                the widget inside of it. The current orientation of the widget
                is preserved, only scaling and translation is performed.
StartWidgetInteraction() - generally corresponds to a initial event (e.g.,
                           mouse down) that starts the interaction process
                           with the widget.
WidgetInteraction() - invoked when an event causes the widget to change
                      appearance.
EndWidgetInteraction() - generally corresponds to a final event (e.g., mouse up)
                         and completes the interaction sequence.
ComputeInteractionState() - given (X,Y) display coordinates in a renderer, with a
                            possible flag that modifies the computation,
                            what is the state of the widget?
GetInteractionState() - return the current state of the widget. Note that the
                        value of ''0'' typically refers to ''outside''. The
                        interaction state is strictly a function of the
                        representation, and the widget/represent must agree
                        on what they mean.
Highlight() - turn on or off any highlights associated with the widget.
              Highlights are generally turned on when the widget is selected.
```

Note that subclasses may ignore some of these methods and implement their own depending on the specifics of the widget.

- `obj.StartWidgetInteraction (double eventPos[2])` - The following is a suggested API for widget representations. These methods define the communication between the widget and its representation. These methods are only suggestions because widgets take on so many different forms that a universal API is not deemed practical. However, these methods should be implemented when possible to insure that the VTK widget hierarchy remains self-consistent.

```
PlaceWidget() - given a bounding box (xmin,xmax,ymin,ymax,zmin,zmax), place
                the widget inside of it. The current orientation of the widget
                is preserved, only scaling and translation is performed.
StartWidgetInteraction() - generally corresponds to a initial event (e.g.,
```

```
                            mouse down) that starts the interaction process
                            with the widget.
WidgetInteraction() - invoked when an event causes the widget to change
                      appearance.
EndWidgetInteraction() - generally corresponds to a final event (e.g., mouse up)
                         and completes the interaction sequence.
ComputeInteractionState() - given (X,Y) display coordinates in a renderer, with a
                            possible flag that modifies the computation,
                            what is the state of the widget?
GetInteractionState() - return the current state of the widget. Note that the
                        value of ''0'' typically refers to ''outside''. The
                        interaction state is strictly a function of the
                        representation, and the widget/represent must agree
                        on what they mean.
Highlight() - turn on or off any highlights associated with the widget.
              Highlights are generally turned on when the widget is selected.
```

Note that subclasses may ignore some of these methods and implement their own depending on the specifics of the widget.

- `obj.WidgetInteraction (double newEventPos[2])` - The following is a suggested API for widget representations. These methods define the communication between the widget and its representation. These methods are only suggestions because widgets take on so many different forms that a universal API is not deemed practical. However, these methods should be implemented when possible to insure that the VTK widget hierarchy remains self-consistent.

```
PlaceWidget() - given a bounding box (xmin,xmax,ymin,ymax,zmin,zmax), place
                the widget inside of it. The current orientation of the widget
                is preserved, only scaling and translation is performed.
StartWidgetInteraction() - generally corresponds to a initial event (e.g.,
                           mouse down) that starts the interaction process
                           with the widget.
WidgetInteraction() - invoked when an event causes the widget to change
                      appearance.
EndWidgetInteraction() - generally corresponds to a final event (e.g., mouse up)
                         and completes the interaction sequence.
ComputeInteractionState() - given (X,Y) display coordinates in a renderer, with a
                            possible flag that modifies the computation,
                            what is the state of the widget?
GetInteractionState() - return the current state of the widget. Note that the
                        value of ''0'' typically refers to ''outside''. The
                        interaction state is strictly a function of the
                        representation, and the widget/represent must agree
                        on what they mean.
Highlight() - turn on or off any highlights associated with the widget.
              Highlights are generally turned on when the widget is selected.
```

Note that subclasses may ignore some of these methods and implement their own depending on the specifics of the widget.

- `obj.EndWidgetInteraction (double newEventPos[2])` - The following is a suggested API for widget representations. These methods define the communication between the widget and its representation. These methods are only suggestions because widgets take on so many different forms that a universal

API is not deemed practical. However, these methods should be implemented when possible to insure that the VTK widget hierarchy remains self-consistent.

```
PlaceWidget() - given a bounding box (xmin,xmax,ymin,ymax,zmin,zmax), place
                the widget inside of it. The current orientation of the widget
                is preserved, only scaling and translation is performed.
StartWidgetInteraction() - generally corresponds to a initial event (e.g.,
                           mouse down) that starts the interaction process
                           with the widget.
WidgetInteraction() - invoked when an event causes the widget to change
                      appearance.
EndWidgetInteraction() - generally corresponds to a final event (e.g., mouse up)
                         and completes the interaction sequence.
ComputeInteractionState() - given (X,Y) display coordinates in a renderer, with a
                            possible flag that modifies the computation,
                            what is the state of the widget?
GetInteractionState() - return the current state of the widget. Note that the
                        value of ''0'' typically refers to ''outside''. The
                        interaction state is strictly a function of the
                        representation, and the widget/represent must agree
                        on what they mean.
Highlight() - turn on or off any highlights associated with the widget.
              Highlights are generally turned on when the widget is selected.
```

Note that subclasses may ignore some of these methods and implement their own depending on the specifics of the widget.

- `int = obj.ComputeInteractionState (int X, int Y, int modify)` - The following is a suggested API for widget representations. These methods define the communication between the widget and its representation. These methods are only suggestions because widgets take on so many different forms that a universal API is not deemed practical. However, these methods should be implemented when possible to insure that the VTK widget hierarchy remains self-consistent.

```
PlaceWidget() - given a bounding box (xmin,xmax,ymin,ymax,zmin,zmax), place
                the widget inside of it. The current orientation of the widget
                is preserved, only scaling and translation is performed.
StartWidgetInteraction() - generally corresponds to a initial event (e.g.,
                           mouse down) that starts the interaction process
                           with the widget.
WidgetInteraction() - invoked when an event causes the widget to change
                      appearance.
EndWidgetInteraction() - generally corresponds to a final event (e.g., mouse up)
                         and completes the interaction sequence.
ComputeInteractionState() - given (X,Y) display coordinates in a renderer, with a
                            possible flag that modifies the computation,
                            what is the state of the widget?
GetInteractionState() - return the current state of the widget. Note that the
                        value of ''0'' typically refers to ''outside''. The
                        interaction state is strictly a function of the
                        representation, and the widget/represent must agree
                        on what they mean.
Highlight() - turn on or off any highlights associated with the widget.
              Highlights are generally turned on when the widget is selected.
```

Note that subclasses may ignore some of these methods and implement their own depending on the specifics of the widget.

- `int = obj.GetInteractionState ()` - The following is a suggested API for widget representations. These methods define the communication between the widget and its representation. These methods are only suggestions because widgets take on so many different forms that a universal API is not deemed practical. However, these methods should be implemented when possible to insure that the VTK widget hierarchy remains self-consistent.

```
PlaceWidget() - given a bounding box (xmin,xmax,ymin,ymax,zmin,zmax), place
                the widget inside of it. The current orientation of the widget
                is preserved, only scaling and translation is performed.
StartWidgetInteraction() - generally corresponds to a initial event (e.g.,
                           mouse down) that starts the interaction process
                           with the widget.
WidgetInteraction() - invoked when an event causes the widget to change
                      appearance.
EndWidgetInteraction() - generally corresponds to a final event (e.g., mouse up)
                         and completes the interaction sequence.
ComputeInteractionState() - given (X,Y) display coordinates in a renderer, with a
                            possible flag that modifies the computation,
                            what is the state of the widget?
GetInteractionState() - return the current state of the widget. Note that the
                        value of ''0'' typically refers to ''outside''. The
                        interaction state is strictly a function of the
                        representation, and the widget/represent must agree
                        on what they mean.
Highlight() - turn on or off any highlights associated with the widget.
              Highlights are generally turned on when the widget is selected.
```

Note that subclasses may ignore some of these methods and implement their own depending on the specifics of the widget.

- `obj.Highlight (int )` - Set/Get a factor representing the scaling of the widget upon placement (via the PlaceWidget() method). Normally the widget is placed so that it just fits within the bounding box defined in PlaceWidget(bounds). The PlaceFactor will make the widget larger (PlaceFactor ¿ 1) or smaller (PlaceFactor ¡ 1). By default, PlaceFactor is set to 0.5.

- `obj.SetPlaceFactor (double )` - Set/Get a factor representing the scaling of the widget upon placement (via the PlaceWidget() method). Normally the widget is placed so that it just fits within the bounding box defined in PlaceWidget(bounds). The PlaceFactor will make the widget larger (PlaceFactor ¿ 1) or smaller (PlaceFactor ¡ 1). By default, PlaceFactor is set to 0.5.

- `double = obj.GetPlaceFactorMinValue ()` - Set/Get a factor representing the scaling of the widget upon placement (via the PlaceWidget() method). Normally the widget is placed so that it just fits within the bounding box defined in PlaceWidget(bounds). The PlaceFactor will make the widget larger (PlaceFactor ¿ 1) or smaller (PlaceFactor ¡ 1). By default, PlaceFactor is set to 0.5.

- `double = obj.GetPlaceFactorMaxValue ()` - Set/Get a factor representing the scaling of the widget upon placement (via the PlaceWidget() method). Normally the widget is placed so that it just fits within the bounding box defined in PlaceWidget(bounds). The PlaceFactor will make the widget larger (PlaceFactor ¿ 1) or smaller (PlaceFactor ¡ 1). By default, PlaceFactor is set to 0.5.

- `double = obj.GetPlaceFactor ()` - Set/Get a factor representing the scaling of the widget upon placement (via the PlaceWidget() method). Normally the widget is placed so that it just fits within

the bounding box defined in PlaceWidget(bounds). The PlaceFactor will make the widget larger (PlaceFactor ¿ 1) or smaller (PlaceFactor ¡ 1). By default, PlaceFactor is set to 0.5.

- `obj.SetHandleSize (double )` - Set/Get the factor that controls the size of the handles that appear as part of the widget (if any). These handles (like spheres, etc.) are used to manipulate the widget. The HandleSize data member allows you to change the relative size of the handles. Note that while the handle size is typically expressed in pixels, some subclasses may use a relative size with respect to the viewport. (As a corollary, the value of this ivar is often set by subclasses of this class during instance instantiation.)

- `double = obj.GetHandleSizeMinValue ()` - Set/Get the factor that controls the size of the handles that appear as part of the widget (if any). These handles (like spheres, etc.) are used to manipulate the widget. The HandleSize data member allows you to change the relative size of the handles. Note that while the handle size is typically expressed in pixels, some subclasses may use a relative size with respect to the viewport. (As a corollary, the value of this ivar is often set by subclasses of this class during instance instantiation.)

- `double = obj.GetHandleSizeMaxValue ()` - Set/Get the factor that controls the size of the handles that appear as part of the widget (if any). These handles (like spheres, etc.) are used to manipulate the widget. The HandleSize data member allows you to change the relative size of the handles. Note that while the handle size is typically expressed in pixels, some subclasses may use a relative size with respect to the viewport. (As a corollary, the value of this ivar is often set by subclasses of this class during instance instantiation.)

- `double = obj.GetHandleSize ()` - Set/Get the factor that controls the size of the handles that appear as part of the widget (if any). These handles (like spheres, etc.) are used to manipulate the widget. The HandleSize data member allows you to change the relative size of the handles. Note that while the handle size is typically expressed in pixels, some subclasses may use a relative size with respect to the viewport. (As a corollary, the value of this ivar is often set by subclasses of this class during instance instantiation.)

- `int = obj.GetNeedToRender ()` - Some subclasses use this data member to keep track of whether to render or not (i.e., to minimize the total number of renders).

- `obj.SetNeedToRender (int )` - Some subclasses use this data member to keep track of whether to render or not (i.e., to minimize the total number of renders).

- `int = obj.GetNeedToRenderMinValue ()` - Some subclasses use this data member to keep track of whether to render or not (i.e., to minimize the total number of renders).

- `int = obj.GetNeedToRenderMaxValue ()` - Some subclasses use this data member to keep track of whether to render or not (i.e., to minimize the total number of renders).

- `obj.NeedToRenderOn ()` - Some subclasses use this data member to keep track of whether to render or not (i.e., to minimize the total number of renders).

- `obj.NeedToRenderOff ()` - Some subclasses use this data member to keep track of whether to render or not (i.e., to minimize the total number of renders).

- `double = obj.GetBounds ()` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e., not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `obj.ShallowCopy (vtkProp prop)` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e.,

not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `obj.GetActors (vtkPropCollection )` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e., not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `obj.GetActors2D (vtkPropCollection )` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e., not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `obj.GetVolumes (vtkPropCollection )` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e., not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `obj.ReleaseGraphicsResources (vtkWindow )` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e., not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `int = obj.RenderOverlay (vtkViewport )` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e., not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `int = obj.RenderOpaqueGeometry (vtkViewport )` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e., not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `int = obj.RenderTranslucentPolygonalGeometry (vtkViewport )` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e., not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `int = obj.RenderVolumetricGeometry (vtkViewport )` - Methods to make this class behave as a vtkProp. They are repeated here (from the vtkProp superclass) as a reminder to the widget implementor. Failure to implement these methods properly may result in the representation not appearing in the scene (i.e., not implementing the Render() methods properly) or leaking graphics resources (i.e., not implementing ReleaseGraphicsResources() properly).

- `int = obj.HasTranslucentPolygonalGeometry ()`

42.107 vtkWidgetSet

42.107.1 Usage

The class synchronizes a set of vtkAbstractWidget(s). Widgets typically invoke "Actions" that drive the geometry/behaviour of their representations in response to interactor events. Interactor interactions on a render window are mapped into "Callbacks" by the widget, from which "Actions" are dispatched to the entire set. This architecture allows us to tie widgets existing in different render windows together. For instance a HandleWidget might exist on the sagittal view. Moving it around should update the representations of the corresponding handle widget that lies on the axial and coronal and volume views as well.

.SECTION User API A user would use this class as follows.

```
vtkWidgetSet *set = vtkWidgetSet::New();
vtkParallelopipedWidget *w1 = vtkParallelopipedWidget::New();
set->AddWidget(w1);
w1->SetInteractor(axialRenderWindow->GetInteractor());
vtkParallelopipedWidget *w2 = vtkParallelopipedWidget::New();
set->AddWidget(w2);
w2->SetInteractor(coronalRenderWindow->GetInteractor());
vtkParallelopipedWidget *w3 = vtkParallelopipedWidget::New();
set->AddWidget(w3);
w3->SetInteractor(sagittalRenderWindow->GetInteractor());
set->SetEnabled(1);
```

.SECTION Motivation The motivation for this class is really to provide a usable API to tie together multiple widgets of the same kind. To enable this, subclasses of vtkAbstractWidget, must be written as follows: They will generally have callback methods mapped to some user interaction such as:

```
this->CallbackMapper->SetCallbackMethod(vtkCommand::LeftButtonPressEvent,
                        vtkEvent::NoModifier, 0, 0, NULL,
                        vtkPaintbrushWidget::BeginDrawStrokeEvent,
                        this, vtkPaintbrushWidget::BeginDrawCallback);
```

The callback invoked when the left button is pressed looks like:

```
void vtkPaintbrushWidget::BeginDrawCallback(vtkAbstractWidget *w)
{
  vtkPaintbrushWidget *self = vtkPaintbrushWidget::SafeDownCast(w);
  self->WidgetSet->DispatchAction(self, \&vtkPaintbrushWidget::BeginDrawAction);
}
```

The actual code for handling the drawing is written in the BeginDrawAction method.

```
void vtkPaintbrushWidget::BeginDrawAction( vtkPaintbrushWidget *dispatcher)
{
// Do stuff to draw...
// Here dispatcher is the widget that was interacted with, the one that
// dispatched an action to all the other widgets in its group. You may, if
// necessary find it helpful to get parameters from it.
//   For instance for a ResizeAction:
//     if (this != dispatcher)
//       {
//       double *newsize = dispatcher->GetRepresentation()->GetSize();
//       this->WidgetRep->SetSize(newsize);
```

```
//       }
//     else
//       {
//       this->WidgetRep->IncrementSizeByDelta();
//       }
}
```

To create an instance of class vtkWidgetSet, simply invoke its constructor as follows

```
obj = vtkWidgetSet
```

42.107.2 Methods

The class vtkWidgetSet has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkWidgetSet class.

- `string = obj.GetClassName ()` - Standard methods for a VTK class.
- `int = obj.IsA (string name)` - Standard methods for a VTK class.
- `vtkWidgetSet = obj.NewInstance ()` - Standard methods for a VTK class.
- `vtkWidgetSet = obj.SafeDownCast (vtkObject o)` - Standard methods for a VTK class.
- `obj.SetEnabled (int )` - Method for activiating and deactiviating all widgets in the group.
- `obj.EnabledOn ()` - Method for activiating and deactiviating all widgets in the group.
- `obj.EnabledOff ()` - Method for activiating and deactiviating all widgets in the group.
- `obj.AddWidget (vtkAbstractWidget )` - Add a widget to the set.
- `obj.RemoveWidget (vtkAbstractWidget )` - Remove a widget from the set
- `int = obj.GetNumberOfWidgets ()` - Get number of widgets in the set.
- `vtkAbstractWidget = obj.GetNthWidget (int )` - Get the Nth widget in the set.

42.108 vtkXYPlotWidget

42.108.1 Usage

This class provides support for interactively manipulating the position, size, and orientation of a XY Plot. It listens to Left mouse events and mouse movement. It will change the cursor shape based on its location. If the cursor is over an edge of thea XY plot it will change the cursor shape to a resize edge shape. If the position of a XY plot is moved to be close to the center of one of the four edges of the viewport, then the XY plot will change its orientation to align with that edge. This orientation is sticky in that it will stay that orientation until the position is moved close to another edge.

To create an instance of class vtkXYPlotWidget, simply invoke its constructor as follows

```
obj = vtkXYPlotWidget
```

42.108.2 Methods

The class vtkXYPlotWidget has several methods that can be used. They are listed below. Note that the documentation is translated automatically from the VTK sources, and may not be completely intelligible. When in doubt, consult the VTK website. In the methods listed below, `obj` is an instance of the vtkXYPlotWidget class.

- `string = obj.GetClassName ()`
- `int = obj.IsA (string name)`
- `vtkXYPlotWidget = obj.NewInstance ()`
- `vtkXYPlotWidget = obj.SafeDownCast (vtkObject o)`
- `obj.SetXYPlotActor (vtkXYPlotActor )` - Get the XY plot used by this Widget. One is created automatically.
- `vtkXYPlotActor = obj.GetXYPlotActor ()` - Get the XY plot used by this Widget. One is created automatically.
- `obj.SetEnabled (int )` - Methods for turning the interactor observer on and off.