PyCBF
A python binding to the CBFlib library

Jon P. Wright
Anyone who wishes to contribute, please do!

Started Dec 12, 2005, already it is August 24, 2010

Abstract

Area detectors at synchrotron facilities can result in huge amounts of data being generated very rapidly. The IUCr (International Union of Crystallography) has devised a standard file format for storing and annotating such data, in order that it might be more easily interchanged and exploited. A c library which gives access to this file format has been developed by Paul Ellis and Herbert Bernstein (Version 0.7.4, http://www.bernstein-plus-sons.com/software/CBF/). In this document a python interface is developed using the SWIG (http://www.swig.org) package in order to give the author easy access to binary cif files.

Contents

Index of file names

Index of macro names

Things to do

- Write test code to test each and every function for good and bad args etc

1 Introduction

The CBFlib library (version 0.7.4) is written in the C language, offering C (and C++) programmers a convenient interface to such files. The current author uses a different language (python) from day to day and so a python interface was desired. After a short attempt to make a quick and dirty SWIG interface it was decided that in the long run it would be better to write a proper interface for python.

All of the functions in the library return an integer reflecting error status. Usually these integers seem to be zero, and a non-zero return value appears to mean an error occurred. Actual return values are returned via pointers in argument lists. In order to simplify the authors life (as a user) all of those integers have been made to disappear if they are zero, and cause an “exception” to be generated if they are not zero. This solution might not be the best thing to do, and it can always be changed where the return value is intended to normally be used.

Actual return values which were passed back via pointer arguments are now just passed back as (perhaps multiple) return values. We must look out for INOUT arguments, none seem to have been found yet, but there might be exceptions. The author has a vague suspicion that python functions generally do not modify their arguments, but this might be wrong.

The library appears to define (at least) three objects. The one we started on was the cbf_handle_struct defined in cbf.h. Many of the functions have their first argument as a pointer to one of these structures. Therefore we make this structure an object and then everything which uses it as first argument is a member function for that object.

In order to pass image data back and forth there is a difficulty that python seems to lack a good way to represent large arrays. The standard library offers an ”array” object which claims to efficiently hold homogenous numerical data. Sadly this seems to be limited to one-dimensional arrays. The builtin string object can hold binary data and this was chosen as the way to pass the actual binary back and forth between python and CBFlib. Unfortunately this means the binary data are pretty useless when they arrive on the python side, so helper functions are provided to convert the data to a python (standard library) 1D array and also to a ”Numeric” array or a ”Numarray” array. The latter two are popular extension modules for manipulating large arrays.

2 Installation prerequisites

The document you are reading was generated from a nuweb source file. This is something very similar to latex with a few extensions for writing out source code files. As such it keeps together the whole package in a single file and makes it easier to write documentation. You will need a to obtain the preprocessing tool nuweb (perhaps from http://nuweb.sourceforge.net) in order to build from scratch with the file pycbf.w. Preproccessed output is hopefully also available to you. We do not recommend editing the SWIG generated wrappers!!

Only python version 2.4 has been targetted originally (other versions?) so that you will probably want to have that version of python installed.

We are building binary extensions, so you also need a working c compiler. The compiler used by the author was gcc (for both windows and unix) with the mingw version under windows.

Finally, you need a copy of swig (from www.swig.org) in order to (re)generate the c wrappers.

In case all that sounds scary, then fear not, it is likely that a single download for windows will just work with the right version of python. Unix systems come with many of those things available anyway.

3 Generating the c interface - the SWIG file

Essentially the swig file starts by saying what to include to build the wrappers, and then goes on to define the python interface for each function we want to call.

The library appears to define at least three "objects"; a CBF handle, a cbf_goniometer and a cbf_detector. We will attempt to map these onto python classes.

FIXME - decide whether introduce a "binary array" class with converters to more common representations?

All of the functions in the library appear to return 0 on success and a meaningful error code on failure. We try to propagate that error code across the language barrier via exceptions.

So the SWIG file will start off by including the header files needed for compilation. Note the defintion of constants to be passed as arguments in calls in the form pycbf.CONSTANTNAME

⟨ Constants used for compression 3a ⟩ ≡

```
// The actual wrappers

// Constants needed from header files

  /* Constants used for compression */

#define CBF_INTEGER     0x0010  /* Uncompressed integer               */
#define CBF_FLOAT       0x0020  /* Uncompressed IEEE floating-point   */
#define CBF_CANONICAL   0x0050  /* Canonical compression              */
#define CBF_PACKED      0x0060  /* Packed compression                 */
#define CBF_PACKED_V2   0x0090  /* CCP4 Packed (JPA) compression V2   */
#define CBF_BYTE_OFFSET 0x0070  /* Byte Offset Compression            */
#define CBF_PREDICTOR   0x0080  /* Predictor_Huffman Compression      */
#define CBF_NONE        0x0040  /* No compression flag                */
#define CBF_COMPRESSION_MASK  \
                        0x00FF  /* Mask to separate compression
                                   type from flags              */
#define CBF_FLAG_MASK   0x0F00  /* Mask to separate flags from
                                   compression type             */
#define CBF_UNCORRELATED_SECTIONS \
                        0x0100  /* Flag for uncorrelated sections     */
#define CBF_FLAT_IMAGE  0x0200  /* Flag for flat (linear) images      */
#define CBF_NO_EXPAND   0x0400  /* Flag to try not to expand          */
◇
```

Macro referenced in 9.

⟨ Constants used for headers 3b ⟩ ≡

```
  /* Constants used for headers */

#define PLAIN_HEADERS   0x0001  /* Use plain ASCII headers            */
#define MIME_HEADERS    0x0002  /* Use MIME headers                   */
#define MSG_NODIGEST    0x0004  /* Do not check message digests       */
#define MSG_DIGEST      0x0008  /* Check message digests              */
#define MSG_DIGESTNOW   0x0010  /* Check message digests immediately  */
```

```
#define MSG_DIGESTWARN  0x0020  /* Warn on message digests immediately*/
#define PAD_1K          0x0020  /* Pad binaries with 1023 0's        */
#define PAD_2K          0x0040  /* Pad binaries with 2047 0's        */
#define PAD_4K          0x0080  /* Pad binaries with 4095 0's        */
◇
```

Macro referenced in 9.

⟨ Constants used to control CIF parsing 4a ⟩ ≡

```
  /* Constants used to control CIF parsing */

#define CBF_PARSE_BRC   0x0100  /* PARSE DDLm/CIF2 brace {,...}             */
#define CBF_PARSE_PRN   0x0200  /* PARSE DDLm parens     (,...)             */
#define CBF_PARSE_BKT   0x0400  /* PARSE DDLm brackets   [,...]             */
#define CBF_PARSE_BRACKETS \
                        0x0700  /* PARSE ALL brackets                       */
#define CBF_PARSE_TQ    0x0800  /* PARSE treble quotes """...""" and '''...'''       */
#define CBF_PARSE_CIF2_DELIMS  \
                        0x1000  /* Do not scan past an unescaped close quote
                                   do not accept {} , : " ' in non-delimited
                                   strings'{ */
#define CBF_PARSE_DDLm  0x0700  /* For DDLm parse (), [], {}                */
#define CBF_PARSE_CIF2  0x1F00  /* For CIF2 parse {}, treble quotes,
                                   stop on unescaped close quotes           */
#define CBF_PARSE_DEFINES      \
                        0x2000  /* Recognize DEFINE_name            */


#define CBF_PARSE_WIDE      0x4000  /* PARSE wide files                         */

#define CBF_PARSE_UTF8      0x10000 /* PARSE UTF-8                              */

#define HDR_DEFAULT (MIME_HEADERS | MSG_NODIGEST)

#define MIME_NOHEADERS  PLAIN_HEADERS

  /* CBF vs CIF */

#define CBF             0x0000  /* Use simple binary sections        */
#define CIF             0x0001  /* Use MIME-encoded binary sections   */
◇
```

Macro referenced in 9.

⟨ Constants used for encoding 4b ⟩ ≡

```
  /* Constants used for encoding */

#define ENC_NONE        0x0001  /* Use BINARY encoding                 */
#define ENC_BASE64      0x0002  /* Use BASE64 encoding                 */
#define ENC_BASE32K     0x0004  /* Use X-BASE32K encoding              */
#define ENC_QP          0x0008  /* Use QUOTED-PRINTABLE encoding       */
#define ENC_BASE10      0x0010  /* Use BASE10 encoding                 */
#define ENC_BASE16      0x0020  /* Use BASE16 encoding                 */
#define ENC_BASE8       0x0040  /* Use BASE8  encoding                 */
#define ENC_FORWARD     0x0080  /* Map bytes to words forward (1234)   */
#define ENC_BACKWARD    0x0100  /* Map bytes to words backward (4321)  */
#define ENC_CRTERM      0x0200  /* Terminate lines with CR             */
#define ENC_LFTERM      0x0400  /* Terminate lines with LF             */

#define ENC_DEFAULT (ENC_BASE64 | ENC_LFTERM | ENC_FORWARD)
◇
```

Macro referenced in 9.

3.1 Exceptions

We attempt to catch the errors and pass them back to python as exceptions. This could still do with a little work to propagage back the calls causing the errors.

Currently there are two global constants defined, called error_message and error_status. These are filled out when an error occurred, converting the numerical error value into something the author can read.

There is an implicit assumption that if the library is used correctly you will not normally get exceptions. This should be addressed further in areas like file opening, proper python exceptions should be returned.

See the section on exception handling in pycbf.i, above.

Currently you get a meaningful string back. Should perhaps look into defining these as python exception classes? In any case - the SWIG exception handling is defined via the following. It could have retained the old style if(status = action) but then harder to see what to return...

⟨ Exception handling 5a ⟩ ≡

```
// Exception handling

  /* Convenience definitions for functions returning error codes */
%exception {
   error_status=0;
   $action
   if (error_status){
     get_error_message();
     PyErr_SetString(PyExc_Exception,error_message);
     return NULL;
   }
}

/* Retain notation from cbf lib but pass on as python exception */

#define cbf_failnez(x) {(error_status = x);}

/* printf("Called \"x\", status %d\n",error_status);} */

#define cbf_onfailnez(x,c) {int err; err = (x); if (err) { fprintf (stderr, \
                      "\nCBFlib error %d in \"x\"\n", err); \
                         { c; } return err; }}
```
◇

Macro referenced in 9.

"pycbf.i" 5b ≡

```
/* File: pycbf.i */

// Indicate that we want to generate a module call pycbf
%module pycbf

%pythoncode %{
__author__ = "Jon Wright <wright@esrf.fr>"
__date__ = "14 Dec 2005"
__version__ = "CBFlib 0.9"
__credits__ = """Paul Ellis and Herbert Bernstein for the excellent CBFlib!"""
__doc__=""" pycbf - python bindings to the CBFlib library

 A library for reading and writing ImageCIF and CBF files
 which store area detector images for crystallography.

 This work is a derivative of the CBFlib version 0.7.7 library
 by  Paul J. Ellis of Stanford Synchrotron Radiation Laboratory
 and Herbert J. Bernstein of Bernstein + Sons
 See:
   http://www.bernstein-plus-sons.com/software/CBF/
 Licensing is GPL based, see:
   http://www.bernstein-plus-sons.com/software/CBF/doc/CBFlib_NOTICES.html

 These bindings were automatically generated by SWIG, and the
 input to SWIG was automatically generated by a python script.
 We very strongly recommend you do not attempt to edit them
 by hand!



 Copyright (C) 2007    Jonathan Wright
                       ESRF, Grenoble, France
               email: wright@esrf.fr

  Revised, August 2010  Herbert J. Bernstein
    Add defines from CBFlib 0.9.1

"""
%}


// Used later to pass back binary data
%include "cstring.i"

// Attempt to autogenerate what SWIG thinks the call looks like

// Typemaps are a SWIG mechanism for many things, not least multiple
// return values
%include "typemaps.i"

// Arrays are needed
%include "carrays.i"
%array_class(double, doubleArray)
%array_class(int, intArray)
%array_class(short, shortArray)
%array_class(long, longArray)

// Following the SWIG 1.3 documentation at
// http://www.swig.org/Doc1.3/Python.html
// section 31.9.5, we map sequences of
// PyFloat, PyLong and PyInt to
// C arrays of double, long and int
//
// But with the strict checking of being a float
// commented out to allow automatic conversions
%{
static int convert_darray(PyObject *input, double *ptr, int size) {
  int i;
  if (!PySequence_Check(input)) {
      PyErr_SetString(PyExc_TypeError,"Expecting a sequence");
      return 0;
  }
  if (PyObject_Length(input) != size) {
      PyErr_SetString(PyExc_ValueError,"Sequence size mismatch");
      return 0;
  }
  for (i =0; i < size; i++) {
      PyObject *o = PySequence_GetItem(input,i);
     /*if (!PyFloat_Check(o)) {

         Py_XDECREF(o);
         PyErr_SetString(PyExc_ValueError,"Expecting a sequence of floats");
         return 0;
      }*/
      ptr[i] = PyFloat_AsDouble(o);
      Py_DECREF(o);
  }
  return 1;
}
%}

%typemap(in) double [ANY](double temp[$1_dim0]) {
    if ($input == Py_None) $1 = NULL;
    else
    if (!convert_darray($input,temp,$1_dim0)) {
      return NULL;
    }
    $1 = &temp[0];
}

%{
    static long convert_larray(PyObject *input, long *ptr, int size) {
        int i;
        if (!PySequence_Check(input)) {
            PyErr_SetString(PyExc_TypeError,"Expecting a sequence");
            return 0;
        }
        if (PyObject_Length(input) != size) {
            PyErr_SetString(PyExc_ValueError,"Sequence size mismatch");
            return 0;
        }
        for (i =0; i < size; i++) {
            PyObject *o = PySequence_GetItem(input,i);
            /*if (!PyLong_Check(o)) {
                Py_XDECREF(o);
                PyErr_SetString(PyExc_ValueError,"Expecting a sequence of long integers");
                return 0;
            }*/
            ptr[i] = PyLong_AsLong(o);
            Py_DECREF(o);
        }
        return 1;
    }
%}

%typemap(in) long [ANY](long temp[$1_dim0]) {
    if (!convert_larray($input,temp,$1_dim0)) {
        return NULL;
    }
    $1 = &temp[0];
}

%{
    static int convert_iarray(PyObject *input, int *ptr, int size) {
        int i;
        if (!PySequence_Check(input)) {
            PyErr_SetString(PyExc_TypeError,"Expecting a sequence");
            return 0;
        }
        if (PyObject_Length(input) != size) {
            PyErr_SetString(PyExc_ValueError,"Sequence size mismatch");
            return 0;
        }
        for (i =0; i < size; i++) {
            PyObject *o = PySequence_GetItem(input,i);
            /*if (!PyInt_Check(o)) {
                Py_XDECREF(o);
                PyErr_SetString(PyExc_ValueError,"Expecting a sequence of long integers");
                return 0;
            }*/
            ptr[i] = (int)PyInt_AsLong(o);
            Py_DECREF(o);
        }
        return 1;
    }
%}

%typemap(in) int [ANY](int temp[$1_dim0]) {
    if (!convert_iarray($input,temp,$1_dim0)) {
        return NULL;
    }
    $1 = &temp[0];
}


%{  // Here is the c code needed to compile the wrappers, but not
    // to be wrapped

#include "../include/cbf.h"
#include "../include/cbf_simple.h"

// Helper functions to generate error message


static int error_status = 0;
static char error_message[1024] ; // hope that is long enough

/* prototype */
void get_error_message(void);

void get_error_message(){
  sprintf(error_message,"%s","CBFlib Error(s):");
  if (error_status & CBF_FORMAT        )
    sprintf(error_message,"%s %s",error_message,"CBF_FORMAT       ");
  if (error_status & CBF_ALLOC         )
    sprintf(error_message,"%s %s",error_message,"CBF_ALLOC        ");
  if (error_status & CBF_ARGUMENT      )
    sprintf(error_message,"%s %s",error_message,"CBF_ARGUMENT     ");
  if (error_status & CBF_ASCII         )
    sprintf(error_message,"%s %s",error_message,"CBF_ASCII        ");
  if (error_status & CBF_BINARY        )
    sprintf(error_message,"%s %s",error_message,"CBF_BINARY       ");
  if (error_status & CBF_BITCOUNT      )
    sprintf(error_message,"%s %s",error_message,"CBF_BITCOUNT     ");
  if (error_status & CBF_ENDOFDATA     )
    sprintf(error_message,"%s %s",error_message,"CBF_ENDOFDATA    ");
  if (error_status & CBF_FILECLOSE     )
    sprintf(error_message,"%s %s",error_message,"CBF_FILECLOSE    ");
  if (error_status & CBF_FILEOPEN      )
    sprintf(error_message,"%s %s",error_message,"CBF_FILEOPEN     ");
  if (error_status & CBF_FILEREAD      )
    sprintf(error_message,"%s %s",error_message,"CBF_FILEREAD     ");
  if (error_status & CBF_FILESEEK      )
    sprintf(error_message,"%s %s",error_message,"CBF_FILESEEK     ");
  if (error_status & CBF_FILETELL      )
    sprintf(error_message,"%s %s",error_message,"CBF_FILETELL     ");
  if (error_status & CBF_FILEWRITE     )
    sprintf(error_message,"%s %s",error_message,"CBF_FILEWRITE    ");
  if (error_status & CBF_IDENTICAL     )
    sprintf(error_message,"%s %s",error_message,"CBF_IDENTICAL    ");
  if (error_status & CBF_NOTFOUND      )
    sprintf(error_message,"%s %s",error_message,"CBF_NOTFOUND     ");
  if (error_status & CBF_OVERFLOW      )
    sprintf(error_message,"%s %s",error_message,"CBF_OVERFLOW     ");
  if (error_status & CBF_UNDEFINED     )
    sprintf(error_message,"%s %s",error_message,"CBF_UNDEFINED    ");
  if (error_status & CBF_NOTIMPLEMENTED)
    sprintf(error_message,"%s %s",error_message,"CBF_NOTIMPLEMENTED");
  if (error_status & CBF_NOCOMPRESSION)
    sprintf(error_message,"%s %s",error_message,"CBF_NOCOMPRESSION");
}


%} // End of code which is not wrapped but needed to compile
◇
```

File defined by 5b, 9.

"pycbf.i" 9 ≡

```
  ⟨ Constants used for compression 3a ⟩

  ⟨ Constants used for headers 3b ⟩

  ⟨ Constants used to control CIF parsing 4a ⟩

  ⟨ Constants used for encoding 4b ⟩

  ⟨ Exception handling 5a ⟩

  %include "cbfgenericwrappers.i"

  // cbf_goniometer object

  %include "cbfgoniometerwrappers.i"

  %include "cbfdetectorwrappers.i"

  // cbfhandle object
  %include "cbfhandlewrappers.i"

  ◇
```

File defined by 5b, 9.

Despite the temptation to just throw everything from the c header files into the interface, a short experience suggested we are better off to pull out only the parts we want and make the calls more pythonic

The input files "CBFhandlewrappers.i", etc. are created by the make_pycbf.py script.

3.2 Exceptions

We attempt to catch the errors and pass them back to python as exceptions. This could still do with a little work to propagage back the calls causing the errors.

Currently there are two global constants defined, called error_message and error_status. These are filled out when an error occurred, converting the numerical error value into something the author can read.

There is an implicit assumption that if the library is used correctly you will not normally get exceptions. This should be addressed further in areas like file opening, proper python exceptions should be returned.

See the section on exception handling in pycbf.i, above.

Currently you get a meaningful string back. Should perhaps look into defining these as python exception classes? In any case - the SWIG exception handling is defined via the following. It could have retained the old style if(status = action) but then harder to see what to return...

4 Docstrings

The file doc/CBFlib.html is converted to a file CBFlib.txt to generate the docstrings and many of the wrappers. The conversion was done by the text-based browser, links.

This text document is then parsed by a python script called make_pycbf.py to generate the .i files which are included by the swig wrapper generator. Unfortunately this more complicated for non-python users but seemed less error prone and involved less typing for the author.

5 Wrappers

The program that does the conversion from CBFlib.txt to the SWIG input files is a python script named make_pycbf.py.

"make_pycbf.py" 10 ≡

```
print "\\begin{verbatim}"
print "This output comes from make_pycbf.py which generates the wrappers"
print "pycbf Copyright (C) 2005  Jonathan Wright, no warranty, LGPL"


######################################################################
#                                                                    #
# YOU MAY REDISTRIBUTE THE CBFLIB PACKAGE INCLUDING PYCBF UNDER THE  #
# TERMS OF THE GPL                                                   #
#                                                                    #
# ALTERNATIVELY YOU MAY REDISTRIBUTE THE CBFLIB API INCLUDING PYCBF  #
# UNDER THE TERMS OF THE LGPL                                        #
#                                                                    #
######################################################################


########################### GPL NOTICES ##############################
#                                                                    #
# This program is free software; you can redistribute it and/or      #
# modify it under the terms of the GNU General Public License as     #
# published by the Free Software Foundation; either version 2 of     #
# (the License, or (at your option) any later version.               #
#                                                                    #
# This program is distributed in the hope that it will be useful,    #
# but WITHOUT ANY WARRANTY; without even the implied warranty of     #
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the      #
# GNU General Public License for more details.                       #
#                                                                    #
# You should have received a copy of the GNU General Public License  #
# along with this program; if not, write to the Free Software        #
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA           #
# 02111-1307  USA                                                    #
#                                                                    #
######################################################################


######################### LGPL NOTICES ###############################
#                                                                    #
# This library is free software; you can redistribute it and/or      #
# modify it under the terms of the GNU Lesser General Public         #
# License as published by the Free Software Foundation; either       #
# version 2.1 of the License, or (at your option) any later version. #
#                                                                    #
# This library is distributed in the hope that it will be useful,    #
# but WITHOUT ANY WARRANTY; without even the implied warranty of     #
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU  #
# Lesser General Public License for more details.                    #
#                                                                    #
# You should have received a copy of the GNU Lesser General Public   #
# License along with this library; if not, write to the Free         #
# Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston,    #
# MA  02110-1301  USA                                                #
#                                                                    #
######################################################################


# Get the ascii text as a list of strings
lines = open("CBFlib.txt","r").readlines()

# Variables to hold the useful things we find in the file
docstring = "\n"
name=""

# Flag to indicate we have not read anything useful yet
on=0


# Dictionary of function prototypes and documentation, keyed by name in C.
name_dict = {}
i=-1
debug = 0
# Parse the text
prototypes = ""
while i<len(lines)-1:
   i=i+1
   line=lines[i]
   nfunc = 0
   if line.find("PROTOTYPE")>=0 and on==1:
      on=10 # Only try for ten lines after it say PROTOTYPE
      continue
   if line.find("#include")>=0: # why?
      continue
   if line.find("int cbf_")>=0: # We found a function
      # keep going up to DESCRIPTION
      prototypes+=""+lines[i].rstrip()+" "
      # print lines[i].rstrip()
      check=0
      while lines[i+1].find("DESCRIPTION")==-1 and lines[i+1].find("int cbf_")==-1:
         i=i+1
         prototypes+=lines[i].rstrip()+" " # lose the \n
         # print lines[i].rstrip()
         check+=1
         if check>20:
            raise Exception("Runaway prototype "+prototypes)
      on=1 # Keep reading docstring
      continue
   if on > 1: # why?
      on=on-1
   if line.find("3. File format")>=0 and on==1:
      # Stop processing at section 3
      i=len(lines)
   if on==1:
      # Docstring ends at 2.xxx for next function or see also
      # We are losing the see also information for now (needed the section
      # breaks in the rtf file)
      if len(line.strip())==0:
         docstring+="\n"
         continue
      else:
         if docstring[-1]=="\n":
            docstring += line.lstrip().rstrip()
         else:
            docstring =docstring+" "+line.lstrip().rstrip()
      if line.strip()[0] in [str(j) for j in range(9)] or \
            line.find("SEE ALSO")>=0 or \
            line.find("________")>=0 or \
            line.find("--------")>=0:
         if len(docstring)>0:
            # print "Prototypes: ",prototypes
            docstring = docstring.replace("\"", " \\\"") # escape the quotes
            for prototype in prototypes.strip().split(";")[:-1]:
                name = prototype.split("(")[0].strip()
                cname = name.split()[1].strip()
                prototype = prototype.strip()+";"
                name_dict[cname]=[prototype,docstring]
                # print "Prototype: ","::",cname,"::",name,"::", prototype
            prototypes = ""
            # print "Found ",prototype
            docstring="\n"
            prototype=""
            cname=""
            on=0
         else:
            raise Exception("bad docstring")

# End of CBFlib.txt file - now generate wrapper code for swig


def myformat(s,l,indent=0,breakon=" "):
   """
   Try to pretty print lines - this is a pain...
   """
   lines = s.rstrip().split("\n")
   out=""
   for line in lines:
      if len(line)==0:
         continue # skip blank lines
      if len(line)>l:
         words = line.split(breakon)
         newline=words[0]
         if len(words)>1:
            for word in words[1:]:
               if len(newline)+len(word)+1 < l:
                  newline=newline+breakon+word
               else:
                  out = out+newline+breakon+"\n"+indent*" "
                  newline=word
            out += newline+"\n"
         else:
            out += "\n"
      else:
         out += line+"\n" # Last one
   if out == "":
      return "\n"
   else:
      return out


def docstringwrite(pyfunc,input,output,prototype,cbflibdoc):
   doc = "%feature(\"autodoc\", \"\nReturns : "
   returns = ""
   for out in output:
      returns += out+","
   if len(returns)>0:
      doc += myformat(returns[:-1],70,indent = 10,breakon=",")
   else:
      doc += "\n"
   doc += "*args   : "
   takes = ""
   for inp in input:
      takes += inp+","
   if len(takes)>0:
      doc += myformat(takes[:-1],70,indent = 10,breakon=",")
   else:
      doc += "\n"
   doc += "\nC prototype: "+myformat(prototype,65,indent=16,breakon=",")
   doc += "\nCBFLib documentation:\n"+myformat(cbflibdoc,70)+"\")"
   doc += pyfunc+";\n"
   return doc


cbfhandle_specials = {

"cbf_get_integerarrayparameters":["""
%apply int *OUTPUT {int *compression,int *binary_id,
                    int *elsize, int *elsigned, int *elunsigned,
                    int *elements, int *minelement, int *maxelement}
                  get_integerarrayparameters;

    void get_integerarrayparameters(int *compression,int *binary_id,
                        int *elsize, int *elsigned, int *elunsigned,
                        int *elements, int *minelement, int *maxelement){
        unsigned int  comp;
        size_t elsiz, elem;
        cbf_failnez(cbf_get_integerarrayparameters(self,
         &comp,binary_id, &elsiz, elsigned, elunsigned, &elem,
          minelement, maxelement));
        *compression = comp; /* FIXME - does this convert in C? */
        *elsize = elsiz;
        *elements = elem;
        }
""","get_integerarrayparameters",[],["int compression","int binary_id",
     "int elsize", "int elsigned", "int elunsigned",
     "int elements", "int minelement", "int maxelement"]],


"cbf_get_integerarrayparameters_wdims":["""
%cstring_output_allocate_size(char **bo, int *bolen, free(*$1));
%apply int *OUTPUT {int *compression,int *binary_id,
                    int *elsize, int *elsigned, int *elunsigned,
                    int *elements, int *minelement, int *maxelement,
                    int *dimfast, int *dimmid, int *dimslow, int *padding}
                  get_integerarrayparameters_wdims;

    void get_integerarrayparameters_wdims(int *compression,int *binary_id,
                        int *elsize, int *elsigned, int *elunsigned,
                        int *elements, int *minelement, int *maxelement,
                        char **bo, int *bolen,
                        int *dimfast, int *dimmid, int *dimslow, int *padding
                        ){
        unsigned int  comp;
        size_t elsiz, elem, df,dm,ds,pd;
        const char * byteorder;
        char * bot;
        cbf_failnez(cbf_get_integerarrayparameters_wdims(self,
         &comp,binary_id, &elsiz, elsigned, elunsigned, &elem,
          minelement, maxelement, &byteorder,&df,&dm,&ds,&pd ));
        *bolen = strlen(byteorder);
        if (!(bot = (char *)malloc(*bolen))) {cbf_failnez(CBF_ALLOC)}
        strncpy(bot,byteorder,*bolen);
        *bo = bot;
        *compression = comp;
        *elsize = elsiz;
        *elements = elem;
        *dimfast = df;
        *dimmid = dm;
        *dimslow = ds;
        *padding = pd;

        }
""","get_integerarrayparameters_wdims",[],["int compression","int binary_id",
     "int elsize", "int elsigned", "int elunsigned",
     "int elements", "int minelement", "int maxelement", "char **bo", "int *bolen",
     "int dimfast", "int dimmid", "int dimslow", "int padding"]],


"cbf_get_integerarrayparameters_wdims_fs":["""
%cstring_output_allocate_size(char **bo, int *bolen, free(*$1));
%apply int *OUTPUT {int *compression,int *binary_id,
                   int *elsize, int *elsigned, int *elunsigned,
                   int *elements, int *minelement, int *maxelement,
                   int *dimfast, int *dimmid, int *dimslow, int *padding}
                 get_integerarrayparameters_wdims_fs;

    void get_integerarrayparameters_wdims_fs(int *compression,int *binary_id,
                       int *elsize, int *elsigned, int *elunsigned,
                       int *elements, int *minelement, int *maxelement,
                       char **bo, int *bolen,
                       int *dimfast, int *dimmid, int *dimslow, int *padding
                       ){
        unsigned int  comp;
        size_t elsiz, elem, df,dm,ds,pd;
        const char * byteorder;
        char * bot;
        cbf_failnez(cbf_get_integerarrayparameters_wdims_fs(self,
         &comp,binary_id, &elsiz, elsigned, elunsigned, &elem,
          minelement, maxelement, &byteorder,&df,&dm,&ds,&pd ));
        *bolen = strlen(byteorder);
        if (!(bot = (char *)malloc(*bolen))) {cbf_failnez(CBF_ALLOC)}
        strncpy(bot,byteorder,*bolen);
        *bo = bot;
        *compression = comp;
        *elsize = elsiz;
        *elements = elem;
        *dimfast = df;
        *dimmid = dm;
        *dimslow = ds;
        *padding = pd;

        }
""","get_integerarrayparameters_wdims_fs",[],["int compression","int binary_id",
     "int elsize", "int elsigned", "int elunsigned",
     "int elements", "int minelement", "int maxelement", "char **bo", "int *bolen",
      "int dimfast", "int dimmid", "int dimslow", "int padding"]],


"cbf_get_integerarrayparameters_wdims_sf":["""
%cstring_output_allocate_size(char **bo, int *bolen, free(*$1));
%apply int *OUTPUT {int *compression,int *binary_id,
                   int *elsize, int *elsigned, int *elunsigned,
                    int *elements, int *minelement, int *maxelement,
                    int *dimslow, int *dimmid, int *dimfast, int *padding}
                  get_integerarrayparameters_wdims_sf;

    void get_integerarrayparameters_wdims_sf(int *compression,int *binary_id,
                        int *elsize, int *elsigned, int *elunsigned,
                        int *elements, int *minelement, int *maxelement,
                        char **bo, int *bolen,
                        int *dimslow, int *dimmid, int *dimfast, int *padding
                        ){
        unsigned int  comp;
        size_t elsiz, elem, df,dm,ds,pd;
        const char * byteorder;
        char * bot;
        cbf_failnez(cbf_get_integerarrayparameters_wdims_sf(self,
         &comp,binary_id, &elsiz, elsigned, elunsigned, &elem,
          minelement, maxelement, &byteorder,&ds,&dm,&df,&pd ));
        *bolen = strlen(byteorder);
        if (!(bot = (char *)malloc(*bolen))) {cbf_failnez(CBF_ALLOC)}
        strncpy(bot,byteorder,*bolen);
        *bo = bot;
        *compression = comp;
        *elsize = elsiz;
        *elements = elem;
        *dimfast = df;
        *dimmid = dm;
        *dimslow = ds;
        *padding = pd;

        }
""","get_integerarrayparameters_wdims_sf",[],["int compression","int binary_id",
     "int elsize", "int elsigned", "int elunsigned",
     "int elements", "int minelement", "int maxelement", "char **bo", "int *bolen",
      "int dimslow", "int dimmid", "int dimfast", "int padding"]],


"cbf_get_realarrayparameters":["""
%apply int *OUTPUT {int *compression,int *binary_id,
                    int *elsize, int *elements} get_realarrayparameters;


    void get_realarrayparameters(int *compression,int *binary_id,
                                 int *elsize, int *elements){
        unsigned int  comp;
        size_t elsiz, elem;
        cbf_failnez(cbf_get_realarrayparameters(self,
                                 &comp ,binary_id, &elsiz, &elem ));
        *compression = comp; /* FIXME - does this convert in C? */
        *elsize = elsiz;
        *elements = elem;
        }
""","get_realarrayparameters",[],["int compression","int binary_id",
     "int elsize", "int elements"]],


"cbf_get_realarrayparameters_wdims":["""
%cstring_output_allocate_size(char **bo, int *bolen, free(*$1));
%apply int *OUTPUT {int *compression,int *binary_id,
                    int *elsize,
                    int *elements,
                    int *dimslow, int *dimmid, int *dimfast, int *padding}
                  get_realarrayparameters_wdims;
    void get_realarrayparameters_wdims(int *compression,int *binary_id,
                        int *elsize,
                        int *elements,
                        char **bo, int *bolen,
                        int *dimfast, int *dimmid, int *dimslow, int *padding
                        ){
        unsigned int  comp;
        size_t elsiz, elem, df,dm,ds,pd;
        const char * byteorder;
        char * bot;
        cbf_failnez(cbf_get_realarrayparameters_wdims(self,
         &comp,binary_id, &elsiz, &elem,
         &byteorder,&ds,&dm,&ds,&pd ));
        *bolen = strlen(byteorder);
        if (!(bot = (char *)malloc(*bolen))) {cbf_failnez(CBF_ALLOC)}
        strncpy(bot,byteorder,*bolen);
        *bo = bot;
        *compression = comp;
        *elsize = elsiz;
        *elements = elem;
        *dimfast = df;
        *dimmid = dm;
        *dimslow = ds;
        *padding = pd;

        }
""","get_realarrayparameters_wdims",[],["int compression","int binary_id",
     "int elsize",
     "int elements", "char **bo", "int *bolen",
     "int dimfast", "int dimmid", "int dimslow", "int padding"]],


"cbf_get_realarrayparameters_wdims_fs":["""
%cstring_output_allocate_size(char **bo, int *bolen, free(*$1));
%apply int *OUTPUT {int *compression,int *binary_id,
                    int *elsize,
                    int *elements,
                    int *dimslow, int *dimmid, int *dimfast, int *padding}
                  get_realarrayparameters_wdims_fs;

    void get_realarrayparameters_wdims_fs(int *compression,int *binary_id,
                        int *elsize,
                        int *elements,
                        char **bo, int *bolen,
                        int *dimfast, int *dimmid, int *dimslow, int *padding
                        ){
        unsigned int  comp;
        size_t elsiz, elem, df,dm,ds,pd;
        const char * byteorder;
        char * bot;
        cbf_failnez(cbf_get_realarrayparameters_wdims_fs(self,
         &comp,binary_id, &elsiz, &elem,
         &byteorder,&ds,&dm,&ds,&pd ));
        *bolen = strlen(byteorder);
        if (!(bot = (char *)malloc(*bolen))) {cbf_failnez(CBF_ALLOC)}
        strncpy(bot,byteorder,*bolen);
        *bo = bot;
        *compression = comp;
        *elsize = elsiz;
        *elements = elem;
        *dimfast = df;
        *dimmid = dm;
        *dimslow = ds;
        *padding = pd;

        }
""","get_realarrayparameters_wdims_fs",[],["int compression","int binary_id",
     "int elsize",
     "int elements", "char **bo", "int *bolen",
      "int dimfast", "int dimmid", "int dimslow", "int padding"]],


"cbf_get_realarrayparameters_wdims_sf":["""
%cstring_output_allocate_size(char **bo, int *bolen, free(*$1));
%apply int *OUTPUT {int *compression,int *binary_id,
                    int *elsize,
                    int *elements,
                    int *dimslow, int *dimmid, int *dimfast, int *padding}
                  get_realarrayparameters_wdims_sf;

    void get_realarrayparameters_wdims_sf(int *compression,int *binary_id,
                        int *elsize,
                        int *elements,
                        char **bo, int *bolen,
                        int *dimslow, int *dimmid, int *dimfast, int *padding
                        ){
        unsigned int  comp;
        size_t elsiz, elem, df,dm,ds,pd;
        const char * byteorder;
        char * bot;
        cbf_failnez(cbf_get_realarrayparameters_wdims_sf(self,
         &comp,binary_id, &elsiz, &elem,
         &byteorder,&ds,&dm,&df,&pd ));
        *bolen = strlen(byteorder);
        if (!(bot = (char *)malloc(*bolen))) {cbf_failnez(CBF_ALLOC)}
        strncpy(bot,byteorder,*bolen);
        *bo = bot;
        *compression = comp;
        *elsize = elsiz;
        *elements = elem;
        *dimfast = df;
        *dimmid = dm;
        *dimslow = ds;
        *padding = pd;

        }
""","get_realarrayparameters_wdims_sf",[],["int compression","int binary_id",
     "int elsize",
     "int elements", "char **bo", "int *bolen",
      "int dimslow", "int dimmid", "int dimfast", "int padding"]],


"cbf_get_integerarray":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_integerarray_as_string;

// Get the length correct

    void get_integerarray_as_string(char **s, int *slen){
        int binary_id, elsigned, elunsigned;
        size_t elements, elements_read, elsize;
        int minelement, maxelement;
        unsigned int compression;
        void * array;
        *slen = 0; /* Initialise in case of problems */
        cbf_failnez(cbf_get_integerarrayparameters(self, &compression,
               &binary_id, &elsize, &elsigned, &elunsigned,
               &elements, &minelement, &maxelement));

        if ((array=malloc(elsize*elements))) {
              /* cbf_failnez (cbf_select_column(cbf,colnum)) */
               cbf_failnez (cbf_get_integerarray(self, &binary_id,
                            (void *)array, elsize, elsigned,
                            elements, &elements_read));

         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*elements;
        *s = (char *) array;
      }
""","get_integerarray_as_string",[],["(Binary)String"] ],


"cbf_get_image":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_image_as_string;

// Get the length correct

    void get_image_as_string(int element_number, char **s, int *slen,
    int elsize, int elsign, int ndimslow, int ndimfast){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimslow))) {
               cbf_failnez (cbf_get_image(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize, elsign,
               (size_t) ndimslow, (size_t)ndimfast));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimslow;
        *s = (char *) array;
      }
""","get_image_as_string",["int element_number",
    "int elsize", "int elsign", "int ndimslow", "int ndimfast"],["(Binary)String"] ],


"cbf_get_image_fs":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_image_fs_as_string;

// Get the length correct

    void get_image_fs_as_string(int element_number, char **s, int *slen,
    int elsize, int elsign, int ndimfast, int ndimslow){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimslow))) {
               cbf_failnez (cbf_get_image_fs(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize, elsign,
               (size_t) ndimfast, (size_t)ndimslow));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimslow;
        *s = (char *) array;
      }
""","get_image_fs_as_string",["int element_number",
    "int elsize", "int elsign", "int ndimfast", "int ndimslow"],["(Binary)String"] ],


"cbf_get_image_sf":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_image_fs_as_string;

// Get the length correct

    void get_image_sf_as_string(int element_number, char **s, int *slen,
    int elsize, int elsign, int ndimslow, int ndimfast){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimslow))) {
               cbf_failnez (cbf_get_image_sf(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize, elsign,
               (size_t) ndimslow, (size_t)ndimfast));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimslow;
        *s = (char *) array;
      }
""","get_image_sf_as_string",["int element_number",
    "int elsize", "int elsign", "int ndimslow", "int ndimfast"],["(Binary)String"] ],


"cbf_get_real_image":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_real_image_as_string;

// Get the length correct

    void get_real_image_as_string(int element_number, char **s, int *slen,
    int elsize, int ndimslow, int ndimfast){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimslow))) {
               cbf_failnez (cbf_get_real_image(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize,
               (size_t) ndimslow, (size_t)ndimfast));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimslow;
        *s = (char *) array;
      }
""","get_real_image_as_string",["int element_number",
    "int elsize", "int ndimslow", "int ndimfast"],["(Binary)String"] ],


"cbf_get_real_image_fs":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_real_image_fs_as_string;

// Get the length correct

    void get_real_image_fs_as_string(int element_number, char **s, int *slen,
    int elsize, int ndimfast, int ndimslow){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimslow))) {
               cbf_failnez (cbf_get_real_image_fs(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize,
               (size_t) ndimfast, (size_t)ndimslow));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimslow;
        *s = (char *) array;
      }
""","get_real_image_fs_as_string",["int element_number",
    "int elsize", "int ndimfast", "int ndimslow"],["(Binary)String"] ],


"cbf_get_real_image_sf":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_real_image_sf_as_string;

// Get the length correct

    void get_real_image_sf_as_string(int element_number, char **s, int *slen,
    int elsize, int ndimslow, int ndimfast){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimslow))) {
               cbf_failnez (cbf_get_real_image_sf(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize,
               (size_t) ndimslow, (size_t)ndimfast));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimslow;
        *s = (char *) array;
      }
""","get_real_image_sf_as_string",["int element_number",
    "int elsize", "int ndimslow", "int ndimfast"],["(Binary)String"] ],


"cbf_get_3d_image":["""
```

```
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_3d_image_as_string;

// Get the length correct

    void get_3d_image_as_string(int element_number, char **s, int *slen,
    int elsize, int elsign, int ndimfast, int ndimmid, int ndimslow){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimmid*ndimslow))) {
               cbf_failnez (cbf_get_3d_image(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize, elsign,
               (size_t) ndimslow, (size_t)ndimmid, (size_t)ndimfast));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimmid*ndimslow;
        *s = (char *) array;
      }
""","get_3d_image_as_string",["int element_number",
    "int elsize", "int elsign", "int ndimslow", "int ndimmid", "int ndimfast"],["(Binary)String"] ],


"cbf_get_3d_image_fs":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_3d_image_fs_as_string;

// Get the length correct

    void get_3d_image_fs_as_string(int element_number, char **s, int *slen,
    int elsize, int elsign, int ndimfast, int ndimmid, int ndimslow){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimmid*ndimslow))) {
               cbf_failnez (cbf_get_3d_image_fs(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize, elsign,
               (size_t) ndimfast, (size_t)ndimmid, (size_t)ndimslow));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimmid*ndimslow;
        *s = (char *) array;
      }
""","get_3d_image_fs_as_string",["int element_number",
    "int elsize", "int elsign", "int ndimfast", "int ndimmid", "int ndimslow"],["(Binary)String"] ],


"cbf_get_3d_image_sf":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_3d_image_sf_as_string;

// Get the length correct
```

```
    void get_3d_image_sf_as_string(int element_number, char **s, int *slen,
    int elsize, int elsign, int ndimfast, int ndimmid, int ndimslow){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimmid*ndimslow))) {
               cbf_failnez (cbf_get_3d_image_sf(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize, elsign,
               (size_t) ndimslow, (size_t)ndimmid, (size_t)ndimfast));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimmid*ndimslow;
        *s = (char *) array;
      }
""","get_3d_image_sf_as_string",["int element_number",
    "int elsize", "int elsign", "int ndimslow", "int ndimmid", "int ndimfast"],["(Binary)String"] ],


"cbf_get_real_3d_image":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_real_3d_image_as_string;

// Get the length correct

    void get_real_3d_image_as_string(int element_number, char **s, int *slen,
    int elsize, int ndimslow, int ndimmid, int ndimfast){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimmid*ndimslow))) {
               cbf_failnez (cbf_get_real_3d_image(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize,
               (size_t) ndimslow, (size_t)ndimmid, (size_t)ndimfast));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimmid*ndimslow;
        *s = (char *) array;
      }
""","get_real_3d_image_as_string",["int element_number",
    "int elsize", "int ndimslow", "int ndimmid", "int ndimfast"],["(Binary)String"] ],


"cbf_get_real_3d_image_fs":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_real_3d_image_fs_as_string;

// Get the length correct

    void get_real_3d_image_fs_as_string(int element_number, char **s, int *slen,
    int elsize, int ndimfast, int ndimmid, int ndimslow){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimmid*ndimslow))) {
               cbf_failnez (cbf_get_real_3d_image_fs(self,
                reserved, (unsigned int)element_number,
                (void *)array, (size_t)elsize,
                (size_t) ndimfast, (size_t)ndimmid, (size_t)ndimslow));
         }else{
                cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimmid*ndimslow;
        *s = (char *) array;
      }
""","get_real_3d_image_fs_as_string",["int element_number",
    "int elsize", "int ndimfast", "int ndimmid", "int ndimslow"],["(Binary)String"] ],

"cbf_get_real_3d_image_sf":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_real_3d_image_sf_as_string;

// Get the length correct

    void get_real_3d_image_sf_as_string(int element_number, char **s, int *slen,
    int elsize, int ndimslow, int ndimmid, int ndimfast){
        void *array;
        int reserved = 0;
        *slen = 0; /* Initialise in case of problems */
        if ((array=malloc(elsize*ndimfast*ndimmid*ndimslow))) {
               cbf_failnez (cbf_get_real_3d_image_sf(self,
               reserved, (unsigned int)element_number,
               (void *)array, (size_t)elsize,
               (size_t) ndimslow, (size_t)ndimmid, (size_t)ndimfast));
         }else{
               cbf_failnez(CBF_ALLOC);
         }
        *slen = elsize*ndimfast*ndimmid*ndimslow;
        *s = (char *) array;
      }
""","get_real_3d_image_sf_as_string",["int element_number",
    "int elsize", "int ndimslow", "int ndimmid", "int ndimfast"],["(Binary)String"] ],


"cbf_get_realarray":["""
// Ensure we free the local temporary

%cstring_output_allocate_size(char ** s, int *slen, free(*$1))
       get_realarray_as_string;

// Get the length correct

    void get_realarray_as_string(char **s, int *slen){
        int binary_id;
        size_t elements, elements_read, elsize;
        unsigned int compression;
        void * array;
        *slen = 0; /* Initialise in case of problems */
        cbf_failnez(cbf_get_realarrayparameters(self, &compression,
               &binary_id, &elsize,
               &elements));

        if ((array=malloc(elsize*elements))) {
              /* cbf_failnez (cbf_select_column(cbf,colnum)) */
               cbf_failnez (cbf_get_realarray(self, &binary_id,
                            (void *)array, elsize,
                            elements, &elements_read));
```

```
          }else{
                cbf_failnez(CBF_ALLOC);
          }
         *slen = elsize*elements;
         *s = (char *) array;
      }
""","get_realarray_as_string",[],["(Binary)String"] ],


"cbf_set_integerarray":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_integerarray;

    void set_integerarray(unsigned int compression, int binary_id,
             char *data, int len, int elsize, int elsigned, int elements){
        /* safety check on args */
        size_t els, ele;
        void *array;
        if(len == elsize*elements){
           array = data;
           els = elsize;
           ele = elements;
           cbf_failnez(cbf_set_integerarray (self, compression, binary_id,
           (void *) data,  (size_t) elsize, elsigned, (size_t) elements));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_integerarray",
[ "int compression", "int binary_id","(binary) String data",
 "int elsize", "int elsigned","int elements"],[]],


"cbf_set_integerarray_wdims":["""
    /* CBFlib must NOT modify the data string nor the byteorder string
       which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_integerarray_wdims;
%apply (char *STRING, int LENGTH) { (char *bo, int bolen) } set_integerarray_wdims;

    void set_integerarray_wdims(unsigned int compression, int binary_id,
             char *data, int len, int elsize, int elsigned, int elements,
             char *bo, int bolen, int dimfast, int dimmid, int dimslow, int padding){
        /* safety check on args */
        size_t els, ele;
        void *array;
        char byteorder[15];
        if(len == elsize*elements && elements==dimfast*dimmid*dimslow){
           array = data;
           els = elsize;
           ele = elements;
           strncpy(byteorder,bo,bolen<15?bolen:14);
           byteorder[bolen<15?14:bolen] = 0;
           cbf_failnez(cbf_set_integerarray_wdims (self, compression, binary_id,
           (void *) data,  (size_t) elsize, elsigned, (size_t) elements, (const char *)byteorder,
           (size_t)dimfast, (size_t)dimmid, (size_t)dimslow, (size_t)padding));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_integerarray_wdims",
[ "int compression", "int binary_id","(binary) String data",
 "int elsize","int elements", "String byteorder", "int dimfast", "int dimmid", "int dimslow", "int padding"],[]


"cbf_set_integerarray_wdims_sf":["""
    /* CBFlib must NOT modify the data string nor the byteorder string
       which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_integerarray_wdims_sf;
%apply (char *STRING, int LENGTH) { (char *bo, int bolen) } set_integerarray_wdims_sf;

    void set_integerarray_wdims_sf(unsigned int compression, int binary_id,
             char *data, int len, int elsize, int elsigned, int elements,
             char *bo, int bolen, int dimslow, int dimmid, int dimfast, int padding){
        /* safety check on args */
        size_t els, ele;
        void *array;
        char byteorder[15];
        if(len == elsize*elements && elements==dimfast*dimmid*dimslow){
           array = data;
           els = elsize;
           ele = elements;
           strncpy(byteorder,bo,bolen<15?bolen:14);
           byteorder[bolen<15?14:bolen] = 0;
           cbf_failnez(cbf_set_integerarray_wdims_sf (self, compression, binary_id,
           (void *) data,  (size_t) elsize, elsigned, (size_t) elements, (const char *)byteorder,
           (size_t)dimslow, (size_t)dimmid, (size_t)dimfast, (size_t)padding));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_integerarray_wdims_sf",
[ "int compression", "int binary_id","(binary) String data",
 "int elsize","int elements", "String byteorder", "int dimslow", "int dimmid", "int dimfast", "int padding"],[]

"cbf_set_integerarray_wdims_fs":["""
    /* CBFlib must NOT modify the data string nor the byteorder string
       which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_integerarray_wdims_fs;
%apply (char *STRING, int LENGTH) { (char *bo, int bolen) } set_integerarray_wdims_fs;

    void set_integerarray_wdims_fs(unsigned int compression, int binary_id,
             char *data, int len, int elsize, int elsigned, int elements,
             char *bo, int bolen, int dimfast, int dimmid, int dimslow, int padding){
        /* safety check on args */
        size_t els, ele;
        void *array;
        char byteorder[15];
        if(len == elsize*elements && elements==dimfast*dimmid*dimslow){
           array = data;
           els = elsize;
           ele = elements;
           strncpy(byteorder,bo,bolen<15?bolen:14);
           byteorder[bolen<15?14:bolen] = 0;
           cbf_failnez(cbf_set_integerarray_wdims_fs (self, compression, binary_id,
           (void *) data,  (size_t) elsize, elsigned, (size_t) elements, (const char *)byteorder,
           (size_t)dimfast, (size_t)dimmid, (size_t)dimslow, (size_t)padding));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_integerarray_wdims_fs",
[ "int compression", "int binary_id","(binary) String data",
 "int elsize","int elements", "String byteorder", "int dimfast", "int dimmid", "int dimslow", "int padding"],[]


"cbf_set_realarray":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_realarray;

    void set_realarray(unsigned int compression, int binary_id,
             char *data, int len, int elsize, int elements){
        /* safety check on args */
        size_t els, ele;
        void *array;
        if(len == elsize*elements){
           array = data;
           els = elsize;
           ele = elements;
           cbf_failnez(cbf_set_realarray (self, compression, binary_id,
           (void *) data,  (size_t) elsize, (size_t) elements));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_realarray",
[ "int compression", "int binary_id","(binary) String data",
 "int elsize","int elements"],[]],


"cbf_set_realarray_wdims":["""
    /* CBFlib must NOT modify the data string nor the byteorder string
       which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_realarray_wdims;
%apply (char *STRING, int LENGTH) { (char *bo, int bolen) } set_realarray_wdims;

    void set_realarray_wdims(unsigned int compression, int binary_id,
             char *data, int len, int elsize, int elements,
             char *bo, int bolen, int dimfast, int dimmid, int dimslow, int padding){
        /* safety check on args */
        size_t els, ele;
        void *array;
        char byteorder[15];
        if(len == elsize*elements && elements==dimfast*dimmid*dimslow){
           array = data;
           els = elsize;
           ele = elements;
           strncpy(byteorder,bo,bolen<15?bolen:14);
           byteorder[bolen<15?14:bolen] = 0;
           cbf_failnez(cbf_set_realarray_wdims (self, compression, binary_id,
           (void *) data,  (size_t) elsize, (size_t) elements, (const char *)byteorder,
           (size_t)dimfast, (size_t)dimmid, (size_t)dimslow, (size_t)padding));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_realarray_wdims",
[ "int compression", "int binary_id","(binary) String data",
 "int elsize","int elements", "String byteorder", "int dimfast", "int dimmid", "int dimslow", "int padding"],[]


"cbf_set_realarray_wdims_sf":["""
    /* CBFlib must NOT modify the data string nor the byteorder string
       which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_realarray_wdims_sf;
%apply (char *STRING, int LENGTH) { (char *bo, int bolen) } set_realarray_wdims_sf;

    void set_realarray_wdims_sf(unsigned int compression, int binary_id,
             char *data, int len, int elsize, int elements,
             char *bo, int bolen, int dimslow, int dimmid, int dimfast, int padding){
        /* safety check on args */
        size_t els, ele;
        void *array;
        char byteorder[15];
        if(len == elsize*elements && elements==dimfast*dimmid*dimslow){
           array = data;
           els = elsize;
           ele = elements;
           strncpy(byteorder,bo,bolen<15?bolen:14);
           byteorder[bolen<15?14:bolen] = 0;
           cbf_failnez(cbf_set_realarray_wdims_sf (self, compression, binary_id,
           (void *) data,  (size_t) elsize, (size_t) elements, (const char *)byteorder,
           (size_t) dimslow, (size_t) dimmid, (size_t) dimfast, (size_t)padding));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_realarray_wdims_sf",
[ "int compression", "int binary_id","(binary) String data",
 "int elsize","int elements", "String byteorder", "int dimslow", "int dimmid", "int dimfast", "int padding"],[]


"cbf_set_realarray_wdims_fs":["""
    /* CBFlib must NOT modify the data string nor the byteorder string
       which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_realarray_wdims_fs;
%apply (char *STRING, int LENGTH) { (char *bo, int bolen) } set_realarray_wdims_fs;

    void set_realarray_wdims_fs(unsigned int compression, int binary_id,
             char *data, int len, int elsize, int elements,
             char *bo, int bolen, int dimfast, int dimmid, int dimslow, int padding){
        /* safety check on args */
        size_t els, ele;
        void *array;
        char byteorder[15];
        if(len == elsize*elements && elements==dimfast*dimmid*dimslow){
           array = data;
           els = elsize;
           ele = elements;
           strncpy(byteorder,bo,bolen<15?bolen:14);
           byteorder[bolen<15?14:bolen] = 0;
           cbf_failnez(cbf_set_realarray_wdims_fs (self, compression, binary_id,
           (void *) data,  (size_t) elsize, (size_t) elements, (const char *)byteorder,
           (size_t) dimfast, (size_t) dimmid, (size_t) dimslow, (size_t)padding));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_realarray_wdims_fs",
[ "int compression", "int binary_id","(binary) String data",
 "int elsize","int elements", "String byteorder", "int dimfast", "int dimmid", "int dimslow", "int padding"],[]


"cbf_set_image":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_image;

    void set_image(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int elsign, int ndimslow, int ndimfast){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimfast){
           array = data;
           els = elsize;
           reserved = 0;
           cbf_failnez(cbf_set_image (self, reserved, element_number, compression,
           (void *) data,  (size_t) elsize, elsign, (size_t) ndimslow, (size_t)ndimfast));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_image",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int elsign", "int dimslow", "int dimfast"],[]],


"cbf_set_image_fs":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_image;

    void set_image_fs(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int elsign, int ndimfast, int ndimslow){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimfast){
           array = data;
           els = elsize;
           reserved = 0;
           cbf_failnez(cbf_set_image (self, reserved, element_number, compression,
           (void *) data,  (size_t) elsize, elsign, (size_t) ndimfast, (size_t)ndimslow));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_image_fs",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int elsign", "int dimfast", "int dimslow"],[]],
"cbf_set_image_sf":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_image_sf;

    void set_image_sf(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int elsign, int ndimslow, int ndimfast){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimfast){
           array = data;
           els = elsize;
           reserved = 0;
           cbf_failnez(cbf_set_image_sf (self, reserved, element_number, compression,
           (void *) data,  (size_t) elsize, elsign, (size_t) ndimslow, (size_t)ndimfast));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_image_sf",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int elsign", "int dimslow", "int dimfast"],[]],


"cbf_set_real_image":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_real_image;

    void set_real_image(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int ndimslow, int ndimfast){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimfast){
           array = data;
           els = elsize;
           reserved = 0;
           cbf_failnez(cbf_set_real_image (self, reserved, element_number, compression,
           (void *) data,  (size_t) elsize, (size_t) ndimslow, (size_t)ndimfast));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_real_image",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int dimslow", "int dimfast"],[]],


"cbf_set_real_image_fs":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_real_image;

    void set_real_image_fs(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int ndimfast, int ndimslow){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimfast){
           array = data;
           els = elsize;
           reserved = 0;
           cbf_failnez(cbf_set_real_image_fs (self, reserved, element_number, compression,
           (void *) data,  (size_t) elsize, (size_t) ndimfast, (size_t)ndimslow));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""",
"set_real_image_fs",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int dimfast", "int dimslow"],[]],


"cbf_set_real_image_sf":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_real_image_sf;

    void set_real_image_sf(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int ndimslow, int ndimfast){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimfast){
           array = data;
           els = elsize;
           reserved = 0;
           cbf_failnez(cbf_set_real_image_sf (self, reserved, element_number, compression,
           (void *) data,  (size_t) elsize, (size_t) ndimslow, (size_t)ndimfast));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""",
"set_real_image_sf",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int dimslow", "int dimfast"],[]],


"cbf_set_3d_image":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_3d_image;

    void set_3d_image(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int elsign, int ndimslow, int ndimmid, int ndimfast){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimmid*ndimfast){
```

```
            array = data;
            els = elsize;
            reserved = 0;
            cbf_failnez(cbf_set_3d_image (self, reserved, element_number, compression,
            (void *) data,  (size_t) elsize, elsign, (size_t) ndimslow, (size_t) ndimmid, (size_t)ndimfast));
         }else{
            cbf_failnez(CBF_ARGUMENT);
         }
     }
""","set_3d_image",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int elsign", "int dimslow", "int dimmid", "int dimfast"],[]],


"cbf_set_3d_image_fs":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_3d_image;

    void set_3d_image_fs(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int elsign, int ndimfast, int ndimmid, int ndimslow){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimmid*ndimfast){
            array = data;
            els = elsize;
            reserved = 0;
            cbf_failnez(cbf_set_3d_image_fs (self, reserved, element_number, compression,
            (void *) data,  (size_t) elsize, elsign, (size_t) ndimfast, (size_t) ndimmid, (size_t)ndimslow));
        }else{
            cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_3d_image_fs",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int elsign", "int dimfast", "int dimmid", "int dimslow"],[]],


"cbf_set_3d_image_sf":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_3d_image;

    void set_3d_image_sf(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int elsign, int ndimslow, int ndimmid, int ndimfast){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimmid*ndimfast){
            array = data;
            els = elsize;
            reserved = 0;
            cbf_failnez(cbf_set_3d_image_sf (self, reserved, element_number, compression,
            (void *) data,  (size_t) elsize, elsign, (size_t) ndimslow, (size_t) ndimmid, (size_t)ndimfast));
        }else{
            cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_3d_image_sf",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int elsign", "int dimslow", "int dimmid", "int dimfast"],[]],


"cbf_set_real_3d_image":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_real_3d_image_sf;

    void set_real_3d_image(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int ndimslow, int ndimmid, int ndimfast){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimmid*ndimfast){
           array = data;
           els = elsize;
           reserved = 0;
           cbf_failnez(cbf_set_real_3d_image (self, reserved, element_number, compression,
           (void *) data,  (size_t) elsize, (size_t) ndimslow, (size_t)ndimmid, (size_t)ndimfast));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_real_3d_image",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int dimslow", "int dimmid", "int dimfast"],[]],


"cbf_set_real_3d_image_fs":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_real_3d_image_fs;

    void set_real_3d_image_fs(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int ndimfast, int ndimmid, int ndimslow){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimmid*ndimfast){
           array = data;
           els = elsize;
           reserved = 0;
           cbf_failnez(cbf_set_real_3d_image_fs (self, reserved, element_number, compression,
           (void *) data,  (size_t) elsize, (size_t) ndimfast, (size_t)ndimmid, (size_t)ndimslow));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_real_3d_image_fs",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int dimfast", "int dimmid", "int dimslow"],[]],
"cbf_set_real_3d_image_sf":["""
    /* CBFlib must NOT modify the data string which belongs to the scripting
       language we will get and check the length via a typemap */

%apply (char *STRING, int LENGTH) { (char *data, int len) } set_real_3d_image_sf;

    void set_real_3d_image_sf(unsigned int element_number,
             unsigned int compression,
             char *data, int len, int elsize, int ndimslow, int ndimmid, int ndimfast){
        /* safety check on args */
        size_t els;
        unsigned int reserved;
        void *array;
        if(len == elsize*ndimslow*ndimmid*ndimfast){
           array = data;
           els = elsize;
           reserved = 0;
           cbf_failnez(cbf_set_real_3d_image_sf (self, reserved, element_number, compression,
           (void *) data,  (size_t) elsize, (size_t) ndimslow, (size_t)ndimmid, (size_t)ndimfast));
        }else{
           cbf_failnez(CBF_ARGUMENT);
        }
    }
""","set_real_3d_image_sf",
[ "int element_number","int compression","(binary) String data",
 "int elsize", "int dimslow", "int dimmid", "int dimfast"],[]],


"cbf_get_image_size": ["""
%apply int *OUTPUT {int *ndimslow, int *ndimfast} get_image_size;
     void get_image_size(unsigned int element_number, int *ndimslow, int *ndimfast){
        unsigned int reserved;
        size_t inslow, infast;
        reserved = 0;
        cbf_failnez(cbf_get_image_size(self,reserved,element_number,&inslow,&infast));
        *ndimslow = (int)inslow;
        *ndimfast = (int)infast;
        }
""","get_image_size",["Integer element_number"],["size_t ndim1","size_t ndim2"]],


"cbf_get_image_size_fs": ["""
%apply int *OUTPUT {int *ndimfast, int *ndimslow} get_image_size_fs;
     void get_image_size_fs(unsigned int element_number, int *ndimfast, int *ndimslow){
        unsigned int reserved;
        size_t infast, inslow;
        reserved = 0;
        cbf_failnez(cbf_get_image_size_fs(self,reserved,element_number,&infast,&inslow));
        *ndimfast = (int)infast; /* FIXME - is that how to convert? */
        *ndimslow = (int)inslow;
        }
""","get_image_size_fs",["Integer element_number"],["size_t ndimfast","size_t ndimslow"]],


"cbf_get_image_size_sf": ["""
%apply int *OUTPUT {int *ndimslow, int *ndimfast} get_image_size_sf;
     void get_image_size_sf(unsigned int element_number, int *ndimslow, int *ndimfast){
        unsigned int reserved;
        size_t inslow, infast;
        reserved = 0;
        cbf_failnez(cbf_get_image_size(self,reserved,element_number,&inslow,&infast));
        *ndimslow = (int)inslow;
        *ndimfast = (int)infast;
        }
""","get_image_size_sf",["Integer element_number"],["size_t ndimslow","size_t ndimfast"]],


"cbf_get_3d_image_size": ["""
%apply int *OUTPUT {int *ndimslow, int *ndimmid, int *ndimfast} get_3d_image_size;
     void get_3d_image_size(unsigned int element_number, int *ndimslow, int *ndimmid, int *ndimfast){
        unsigned int reserved;
        size_t inslow, inmid, infast;
        reserved = 0;
        cbf_failnez(cbf_get_3d_image_size(self,reserved,element_number,&inslow,&inmid,&infast));
        *ndimslow = (int)inslow; /* FIXME - is that how to convert? */
        *ndimmid = (int)inmid;
        *ndimfast = (int)infast;
        }
""","get_3d_image_size",["Integer element_number"],["size_t ndimslow","size_t ndimmid","size_t ndimfast"]],


"cbf_get_3d_image_size_fs": ["""
%apply int *OUTPUT {int *ndimslow, int *ndimmid, int *ndimfast} get_3d_image_size;
     void get_3d_image_size_fs(unsigned int element_number, int *ndimfast, int *ndimmid, int *ndimslow){
        unsigned int reserved;
        size_t inslow, inmid, infast;
        reserved = 0;
        cbf_failnez(cbf_get_3d_image_size_fs(self,reserved,element_number,&infast,&inmid,&inslow));
        *ndimslow = (int)inslow; /* FIXME - is that how to convert? */
        *ndimmid = (int)inmid;
        *ndimfast = (int)infast;
        }
""","get_3d_image_size",["Integer element_number"],["size_t ndimfast","size_t ndimmid","size_t ndimslow"]],


"cbf_get_3d_image_size_sf": ["""
%apply int *OUTPUT {int *ndimslow, int *ndimmid, int *ndimfast} get_3d_image_size_sf;
     void get_3d_image_size_sf(unsigned int element_number, int *ndimslow, int *ndimmid, int *ndimfast){
        unsigned int reserved;
        size_t inslow, inmid, infast;
        reserved = 0;
        cbf_failnez(cbf_get_3d_image_size_sf(self,reserved,element_number,&inslow,&inmid,&infast));
        *ndimslow = (int)inslow; /* FIXME - is that how to convert? */
        *ndimmid = (int)inmid;
        *ndimfast = (int)infast;
        }
""","get_3d_image_size_sf",["Integer element_number"],["size_t ndimslow","size_t ndimmid","size_t ndimfast"]],


"cbf_get_pixel_size" : ["""
%apply double *OUTPUT {double *psize} get_pixel_size;
    void get_pixel_size(unsigned int element_number,
                        unsigned int axis_number, double *psize){
        cbf_failnez(cbf_get_pixel_size(self,
                                       element_number,
                                       axis_number,
                                       psize));
    }
""","get_pixel_size",["Int element_number","Int axis_number"],
                     ["Float pixel_size"]] ,


"cbf_get_pixel_size_fs" : ["""
%apply double *OUTPUT {double *psize} get_pixel_size;
    void get_pixel_size_fs(unsigned int element_number,
                        unsigned int axis_number, double *psize){
```

```
        cbf_failnez(cbf_get_pixel_size_fs(self,
                                       element_number,
                                       axis_number,
                                       psize));
    }
""","get_pixel_size_fs",["Int element_number","Int axis_number"],
                     ["Float pixel_size"]] ,


"cbf_get_pixel_size_sf" : ["""
%apply double *OUTPUT {double *psize} get_pixel_size;
    void get_pixel_size_sf(unsigned int element_number,
                        unsigned int axis_number, double *psize){
        cbf_failnez(cbf_get_pixel_size_sf(self,
                                       element_number,
                                       axis_number,
                                       psize));
    }
""","get_pixel_size_sf",["Int element_number","Int axis_number"],
                     ["Float pixel_size"]] ,


"cbf_set_pixel_size":["""
     void set_pixel_size (unsigned int element_number,
                          unsigned int axis_number, double psize){
         cbf_failnez(cbf_set_pixel_size(self,
                                        element_number,
                                        axis_number,
                                        psize));
     }
""","set_pixel_size",
   ["Int element_number","Int axis_number","Float pixel size"],[]],


"cbf_set_pixel_size_fs":["""
     void set_pixel_size_fs (unsigned int element_number,
                          unsigned int axis_number, double psize){
         cbf_failnez(cbf_set_pixel_size_fs(self,
                                        element_number,
                                        axis_number,
                                        psize));
     }
""","set_pixel_size_fs",
   ["Int element_number","Int axis_number","Float pixel size"],[]],


"cbf_set_pixel_size_sf":["""
     void set_pixel_size_sf (unsigned int element_number,
                          unsigned int axis_number, double psize){
         cbf_failnez(cbf_set_pixel_size_sf(self,
                                        element_number,
                                        axis_number,
                                        psize));
     }
""","set_pixel_size_sf",
   ["Int element_number","Int axis_number","Float pixel size"],[]],


"cbf_write_file" : ["""
    void write_file(const char* filename, int ciforcbf, int headers,
                    int encoding){
       FILE *stream;
       int readable;
```

```
        /* Make the file non-0 to make CBFlib close the file */
        readable = 1;
        if ( ! ( stream = fopen (filename, "w+b")) ){
          cbf_failnez(CBF_FILEOPEN);
         }
         else{
         cbf_failnez(cbf_write_file(self, stream, readable,
                     ciforcbf, headers, encoding));

         }
        }
""","write_file",["String filename","Integer ciforcbf","Integer Headers",
                  "Integer encoding"],[]],


"cbf_write_widefile" : ["""
    void write_widefile(const char* filename, int ciforcbf, int headers,
                     int encoding){
        FILE *stream;
        int readable;
        /* Make the file non-0 to make CBFlib close the file */
        readable = 1;
        if ( ! ( stream = fopen (filename, "w+b")) ){
          cbf_failnez(CBF_FILEOPEN);
         }
         else{
         cbf_failnez(cbf_write_widefile(self, stream, readable,
                     ciforcbf, headers, encoding));

         }
        }
""","write_widefile",["String filename","Integer ciforcbf","Integer Headers",
                  "Integer encoding"],[]],


"cbf_read_template":["""
    void read_template(char* filename){
        /* CBFlib needs a stream that will remain open
         hence DO NOT open from python */
        FILE *stream;
        if ( ! ( stream = fopen (filename, "rb")) ){
          cbf_failnez(CBF_FILEOPEN);
         }
         else{
         cbf_failnez(cbf_read_template (self, stream)); }
    }

""","read_template",["String filename"],[]],


"cbf_read_file" : ["""
    void read_file(char* filename, int headers){
        /* CBFlib needs a stream that will remain open
           hence DO NOT open from python */
        FILE *stream;
        if ( ! ( stream = fopen (filename, "rb")) ){
          cbf_failnez(CBF_FILEOPEN);
         }
         else{
          cbf_failnez(cbf_read_file(self, stream, headers));
    }
        }
""","read_file",["String filename","Integer headers"],[]],
"cbf_read_widefile" : ["""
    void read_widefile(char* filename, int headers){
        /* CBFlib needs a stream that will remain open
           hence DO NOT open from python */
        FILE *stream;
        if ( ! ( stream = fopen (filename, "rb")) ){
          cbf_failnez(CBF_FILEOPEN);
         }
         else{
          cbf_failnez(cbf_read_widefile(self, stream, headers));
    }
        }
""","read_widefile",["String filename","Integer headers"],[]],


"cbf_set_doublevalue":["""
     void set_doublevalue(const char *format, double number){
         cbf_failnez(cbf_set_doublevalue(self,format,number));}
""","set_doublevalue",["String format","Float number"],[]],


"cbf_require_integervalue":["""
%apply int *OUTPUT {int *number} require_integervalue;

     void require_integervalue(int *number, int thedefault){

     cbf_failnez(cbf_require_integervalue(self,number,thedefault));

     }
""","require_integervalue", ["Int thedefault"],["Int number"]],


"cbf_require_doublevalue":["""
%apply double *OUTPUT {double *number} require_doublevalue;
void require_doublevalue(double *number, double defaultvalue){
   cbf_failnez(cbf_require_doublevalue(self,number,defaultvalue));
}
""","require_doublevalue",["Float Default"],["Float Number"]],


"cbf_require_column_value":["""
 const char* require_column_value(const char *columnname,
                                  const char *defaultvalue){
   const char * result;
   cbf_failnez(cbf_require_column_value(self,columnname,
                                    &result,defaultvalue));
   return result;
}
""","require_column_value",
    ["String columnnanme","String Default"],["String Name"]],


"cbf_require_column_doublevalue":["""
%apply double *OUTPUT { double *number} require_column_doublevalue;
void require_column_doublevalue(const char *columnname, double * number,
             const double defaultvalue){
    cbf_failnez(cbf_require_column_doublevalue(self,
                  columnname,number,defaultvalue));
    }
""","require_column_doublevalue",["String columnname","Float Value"],
                                ["Float defaultvalue"]],


"cbf_require_column_integervalue":["""
%apply int *OUTPUT {int *number}  require_column_integervalue;
void require_column_integervalue(const char *columnname,
                       int *number, const int defaultvalue){
    cbf_failnez(cbf_require_column_integervalue(self,
           columnname, number,defaultvalue));
    }
""","require_column_integervalue",["String Columnvalue","Int default"],
 ["Int Value"]],





"cbf_require_value" : ["""

   const char* require_value(const char* defaultvalue){
     const char * result;
     cbf_failnez(cbf_require_value(self, &result, defaultvalue));
     return result;
    }
""","require_value",["String defaultvalue"],['String Value']],


"cbf_require_diffrn_id":["""
   const char* require_diffrn_id(const char* defaultid){
     const char * id;
     cbf_failnez(cbf_require_diffrn_id(self,&id,defaultid));
     return id;
     }
""","require_diffrn_id", ["String Default_id"],["String diffrn_id"]],



"cbf_get_polarization":["""
     /* Returns a pair of double values */
%apply double *OUTPUT { double *in1, double *in2 };
     void get_polarization(double *in1,double *in2){
        cbf_failnez(cbf_get_polarization (self, in1, in2));
     }
""","get_polarization",[],
    ["float polarizn_source_ratio","float polarizn_source_norm"]],


"cbf_set_polarization":["""
     void set_polarization (double polarizn_source_ratio,
                            double polarizn_source_norm){
         cbf_failnez(cbf_set_polarization(self,
                         polarizn_source_ratio,
                         polarizn_source_norm));
     }
""","set_polarization",
   ["Float polarizn_source_ratio","Float polarizn_source_norm"],[]],


"cbf_get_divergence":["""
%apply double *OUTPUT {double *div_x_source, double *div_y_source,
                       double *div_x_y_source } get_divergence;
    void get_divergence(double *div_x_source, double *div_y_source,
       double *div_x_y_source){
       cbf_failnez(cbf_get_divergence(self,
                                    div_x_source,
                                    div_y_source,
                                    div_x_y_source));
       }
""","get_divergence",[],
     ["Float div_x_source","Float div_y_source","Float div_x_y_source"]],


"cbf_set_divergence":["""
   void set_divergence ( double div_x_source, double div_y_source,
                         double div_x_y_source){
      cbf_failnez(cbf_set_divergence (self, div_x_source,
                               div_y_source,div_x_y_source));
      }
""","set_divergence",
    ["Float div_x_source","Float div_y_source","Float div_x_y_source"],[]],

"cbf_get_gain":["""
%apply double *OUTPUT {double *gain, double *gain_esd} get_gain;
    void get_gain (unsigned int element_number, double *gain,
                   double *gain_esd){
        cbf_failnez(cbf_get_gain (self, element_number, gain, gain_esd));
        }
""","get_gain",
    [],["Float gain", "Float gain_esd"]],


"cbf_set_gain":["""
    void set_gain (unsigned int element_number, double gain, double gain_esd){
        cbf_failnez(cbf_set_gain (self, element_number, gain, gain_esd));
        }
""","set_gain",["Float gain", "Float gain_esd"],[]],

"cbf_get_element_id":["""
   const char * get_element_id(unsigned int element_number){
       const char * result;
       cbf_failnez(cbf_get_element_id (self, element_number, &result));
       return result;
       }
""","get_element_id", ["Integer element_number"],["String"]],


"cbf_set_axis_setting":["""
   void set_axis_setting(const char *axis_id,
                    double start, double increment){
        unsigned int reserved;
        reserved = 0;
        cbf_failnez(cbf_set_axis_setting(self,reserved,
                         axis_id,start,increment));
        }
""","set_axis_setting",["String axis_id", "Float start", "Float increment"],
 []],


"cbf_get_axis_setting":["""
%apply double *OUTPUT {double *start, double *increment} get_axis_setting;
   void get_axis_setting(const char *axis_id,
                    double *start, double *increment){
        unsigned int reserved;
        reserved = 0;
        cbf_failnez(cbf_get_axis_setting(self,reserved,axis_id,
                         start,increment));
        }
```

```
""","get_axis_setting",["String axis_id"],["Float start", "Float increment"],],



"cbf_get_datestamp":["""
%apply int *OUTPUT {int *year, int *month, int *day, int *hour,
                    int *minute, double *second, int *timezone} get_datestamp;
   void get_datestamp(int *year, int *month, int *day, int *hour,
                      int *minute, double *second, int *timezone){
        unsigned int reserved;
        reserved = 0;
        cbf_failnez(cbf_get_datestamp(self,reserved,
              year,month,day,hour,minute,second,timezone));
        }
""","get_datestamp",[],["int year", "int month", "int day", "int hour",
"int minute", "double second", "int timezone"]],



"cbf_set_datestamp":["""
   void set_datestamp(int year, int month, int day, int hour,
                      int minute, double second, int timezone,
                      double precision){
        unsigned int reserved;
        reserved = 0;
        cbf_failnez(cbf_set_datestamp(self,reserved,
              year,month,day,hour,minute,second,timezone,precision));
        }
""","set_datestamp",["int year", "int month", "int day", "int hour",
"int minute", "double second", "int timezone","Float precision"],[]],


"cbf_get_timestamp":["""
%apply double *OUTPUT {double *time} get_timestamp;
%apply int *OUTPUT {int *timezone} get_timestamp;
    void get_timestamp(double *time, int *timezone){
        unsigned int reserved;
        reserved = 0;
        cbf_failnez(cbf_get_timestamp(self,reserved,time,timezone));
        }
""","get_timestamp",[],["Float time","Integer timezone"]],



"cbf_set_timestamp":["""
    void set_timestamp(double time, int timezone, double precision){
        unsigned int reserved;
        reserved = 0;
        cbf_failnez(cbf_set_timestamp(self,reserved,time,timezone,precision));
        }
""","set_timestamp",["Float time","Integer timezone","Float precision"],[]],


"cbf_set_current_timestamp":["""
    void set_current_timestamp(int timezone){
        unsigned int reserved;
        reserved = 0;
        cbf_failnez(cbf_set_current_timestamp(self,reserved,timezone));
        }
""","set_current_timestamp",["Integer timezone"],[]],
"cbf_get_overload":["""
%apply double *OUTPUT {double *overload} get_overload;
   void get_overload(unsigned int element_number, double *overload){
        cbf_failnez(cbf_get_overload(self,element_number,overload));
        }
""","get_overload",["Integer element_number"],["Float overload"]],

"cbf_set_overload":["""
   void set_overload(unsigned int element_number, double overload){
        cbf_failnez(cbf_set_overload(self,element_number,overload));
        }
""","set_overload",["Integer element_number","Float overload"],[]],


"cbf_set_integration_time":["""
   void set_integration_time(double time){
        unsigned int reserved;
        reserved = 0;
        cbf_failnez(cbf_set_integration_time(self,reserved,time));
        }
""","set_integration_time",["Float time"],[]],



"cbf_get_integration_time":["""
%apply double *OUTPUT {double *time} get_integration_time;
   void get_integration_time( double *time ){
        unsigned int reserved;
        double tim;
        reserved = 0;
        cbf_failnez(cbf_get_integration_time(self,reserved,&tim));
        *time = tim;
        }
""","get_integration_time",[],["Float time"]],

"cbf_get_orientation_matrix":["""
%apply double *OUTPUT {double *m0,double *m1,double *m2,
double *m3,double *m4, double *m5,double *m6,
double *m7,double *m8  } get_orientation_matrix;
   void get_orientation_matrix(  double *m0,double *m1,
double *m2,double *m3,double *m4,double *m5,double *m6,
double *m7,double *m8){
        double m[9];
        cbf_failnez(cbf_get_orientation_matrix(self,m));
        *m0 = m[0]; *m1=m[1] ; *m2=m[2] ;
        *m3 = m[3]; *m4=m[4] ; *m5=m[5] ;
        *m6 = m[6]; *m7=m[7] ; *m8=m[8] ;
        }
""","get_orientation_matrix",
    [],[ "Float matrix_%d"%(ind) for ind in range(9) ]],

"cbf_get_unit_cell":["""
%apply double *OUTPUT {double *a, double *b, double *c,
  double *alpha, double *beta, double *gamma} get_unit_cell;
     void get_unit_cell(double *a, double *b, double *c,
  double *alpha, double *beta, double *gamma) {
     double cell[6];
     cbf_failnez(cbf_get_unit_cell(self,cell,NULL));
     *a = cell[0];
     *b = cell[1];
     *c = cell[2];
     *alpha = cell[3];
     *beta = cell[4];
```

```
      *gamma = cell[5];
    }
"""","get_unit_cell",
    [],["Float a", "Float b", "Float c", "Float alpha", "Float beta", "Float gamma" ] ],

"cbf_get_unit_cell_esd":["""
%apply double *OUTPUT {double *a_esd, double *b_esd, double *c_esd,
  double *alpha_esd, double *beta_esd, double *gamma_esd} get_unit_cell_esd;
     void get_unit_cell_esd(double *a_esd, double *b_esd, double *c_esd,
  double *alpha_esd, double *beta_esd, double *gamma_esd) {
     double cell_esd[6];
     cbf_failnez(cbf_get_unit_cell(self,NULL,cell_esd));
     *a_esd = cell_esd[0];
     *b_esd = cell_esd[1];
     *c_esd = cell_esd[2];
     *alpha_esd = cell_esd[3];
     *beta_esd = cell_esd[4];
     *gamma_esd = cell_esd[5];
    }
"""","get_unit_cell",
    [],["doubleArray cell"] ],


"cbf_get_reciprocal_cell":["""
%apply double *OUTPUT {double *astar, double *bstar, double *cstar,
  double *alphastar, double *betastar, double *gammastar} get_reciprocal_cell;
     void get_reciprocal_cell(double *astar, double *bstar, double *cstar,
  double *alphastar, double *betastar, double *gammastar) {
     double rcell[6];
     cbf_failnez(cbf_get_reciprocal_cell(self,rcell,NULL));
    *astar =      rcell[0];
    *bstar =      rcell[1];
    *cstar =      rcell[2];
    *alphastar =  rcell[3];
    *betastar =   rcell[4];
    *gammastar =  rcell[5];
    }
"""","get_reciprocal_cell",
    [],["Float astar", "Float bstar", "Float cstar", "Float alphastar", "Float betastar", "Float gammastar"] ],

"cbf_get_reciprocal_cell_esd":["""
%apply double *OUTPUT {double *a_esd, double *b_esd, double *c_esd,
  double *alpha_esd, double *beta_esd, double *gamma_esd} get_reciprocal_cell_esd;
     void get_reciprocal_cell_esd(double *a_esd, double *b_esd, double *c_esd,
  double *alpha_esd, double *beta_esd, double *gamma_esd) {
     double cell_esd[6];
     cbf_failnez(cbf_get_reciprocal_cell(self,NULL,cell_esd));
     *a_esd = cell_esd[0];
     *b_esd = cell_esd[1];
     *c_esd = cell_esd[2];
     *alpha_esd = cell_esd[3];
     *beta_esd = cell_esd[4];
     *gamma_esd = cell_esd[5];
    }
"""","get_reciprocal_cell",
    [],["doubleArray cell"] ],



"cbf_set_unit_cell":["""
    void set_unit_cell(double cell[6]) {
      cbf_failnez(cbf_set_unit_cell(self,cell,NULL));
    }
```

```
""","set_unit_cell",
    ["double cell[6]"],[] ],

"cbf_set_unit_cell_esd":["""
   void set_unit_cell_esd(double cell_esd[6]) {
     cbf_failnez(cbf_set_unit_cell(self,NULL,cell_esd));
   }
""","set_unit_cell_esd",
    ["double cell_esd[6]"],[] ],


"cbf_set_reciprocal_cell":["""
   void set_reciprocal_cell(double cell[6]) {
     cbf_failnez(cbf_set_reciprocal_cell(self,cell,NULL));
   }
""","set_reciprocal_cell",
    ["double cell[6]"],[] ],

"cbf_set_reciprocal_cell_esd":["""
   void set_reciprocal_cell_esd(double cell_esd[6]) {
     cbf_failnez(cbf_set_reciprocal_cell(self,NULL,cell_esd));
   }
""","set_reciprocal_cell_esd",
    ["double cell_esd[6]"],[] ],


"cbf_set_tag_category":["""
   void set_tag_category(const char *tagname, const char* categoryname_in){
     cbf_failnez(cbf_set_tag_category(self,tagname, categoryname_in));
     }
""","set_tag_category",["String tagname","String categoryname_in"],[] ],



"cbf_find_tag_category":["""

   const char * find_tag_category(const char *tagname){
     const char * result;
     cbf_failnez(cbf_find_tag_category(self,tagname, &result));
     return result;
     }
""","find_tag_category",["String tagname"],["String categoryname"] ],


"cbf_require_tag_root":["""
const char* require_tag_root(const char* tagname){
 const char* result;
 cbf_failnez(cbf_require_tag_root(self,tagname,&result));
 return result;
 }
""","require_tag_root",["String tagname"],["String tagroot"]],

"cbf_find_tag_root":["""
const char * find_tag_root(const char* tagname){
   const char* result;
   cbf_failnez(cbf_find_tag_root(self,tagname,&result));
   return result;
}
""","find_tag_root",["String tagname"],["String tagroot"]],


"cbf_set_tag_root":["""
void  set_tag_root(const char* tagname, const char* tagroot_in){
   cbf_failnez(cbf_set_tag_root(self,tagname,tagroot_in));
}
""","set_tag_root",["String tagname","String tagroot_in"],[]],

"cbf_set_category_root":["""
void  set_category_root(const char* categoryname, const char* categoryroot){
   cbf_failnez(cbf_set_category_root(self,categoryname,categoryroot));
}
""","set_category_root",["String categoryname","String categoryroot"],[]],


"cbf_find_category_root":["""
const char*  find_category_root(const char* categoryname){
   const char * result;
   cbf_failnez(cbf_find_category_root(self,categoryname,&result));
   return result;
}
""","find_category_root",["String categoryname"],["String categoryroot"]],







"cbf_require_category_root":["""
const char* require_category_root (const char* categoryname){
  const char* result;
  cbf_failnez(cbf_require_category_root(self,categoryname, &result));
  return result;
}
""","cbf_require_category_root",["String Categoryname"],["String categoryroot"]],


"cbf_set_orientation_matrix":["""
   void set_orientation_matrix(  double m0,double m1,
double  m2,double  m3,double  m4,double m5,double m6,
double  m7,double  m8){
        double m[9];
        m[0] = m0; m[1]=m1 ; m[2]=m2 ;
        m[3] = m3; m[4]=m4 ; m[5]=m5 ;
        m[6] = m6; m[7]=m7 ; m[8]=m8 ;
        cbf_failnez(cbf_get_orientation_matrix(self,m));
        }
""","set_orientation_matrix",
    [ "Float matrix_%d"%(ind) for ind in range(9) ] ,[]],

"cbf_set_bin_sizes":["""
   void set_bin_sizes( int element_number, double slowbinsize_in, double fastbinsize_in) {
     cbf_failnez(cbf_set_bin_sizes(self,element_number,slowbinsize_in,fastbinsize_in));
   }
""","set_bin_sizes",["Integer element_number","Float slowbinsize_in","Float fastbinsize_in"],[] ],


"cbf_get_bin_sizes":["""
%apply double *OUTPUT {double *slowbinsize,double *fastbinsize};
  void get_bin_sizes(int element_number, double *slowbinsize, double *fastbinsize) {
    cbf_failnez(cbf_get_bin_sizes (self, (unsigned int)element_number, slowbinsize, fastbinsize));
  }
""","get_bin_sizes",["Integer element_number"],["Float slowbinsize","Float fastbinsize"] ],



# cbfhandle dict functions UNTESTED
```

```
"cbf_require_dictionary":["""
cbf_handle require_dictionary(){
   cbf_handle temp;
   cbf_failnez(cbf_require_dictionary(self,&temp));
   return temp;
}
""","require_dictionary",[],["CBFHandle dictionary"]],


"cbf_get_dictionary":["""
cbf_handle get_dictionary(){
   cbf_handle temp;
   cbf_failnez(cbf_get_dictionary(self,&temp));
   return temp;
}
""","get_dictionary",[],["CBFHandle dictionary"]],



"cbf_set_dictionary":["""
void set_dictionary(cbf_handle other){
   cbf_failnez(cbf_set_dictionary(self,other));
}
""","set_dictionary",["CBFHandle dictionary"],[]],



"cbf_convert_dictionary":["""
void convert_dictionary(cbf_handle other){
   cbf_failnez(cbf_convert_dictionary(self,other));
}
""","convert_dictionary",["CBFHandle dictionary"],[]],


"cbf_construct_detector":["""
 cbf_detector construct_detector(unsigned int element_number){
    cbf_detector detector;
    cbf_failnez(cbf_construct_detector(self,&detector,element_number));
    return detector;
    }
""","construct_detector",["Integer element_number"],["pycbf detector object"]],

"cbf_construct_reference_detector":["""
 cbf_detector construct_reference_detector(unsigned int element_number){
    cbf_detector detector;
    cbf_failnez(cbf_construct_reference_detector(self,&detector,element_number));
    return detector;
    }
""","construct_reference_detector",["Integer element_number"],["pycbf detector object"]],

"cbf_require_reference_detector":["""
 cbf_detector require_reference_detector(unsigned int element_number){
    cbf_detector detector;
    cbf_failnez(cbf_require_reference_detector(self,&detector,element_number));
    return detector;
    }
""","require_reference_detector",["Integer element_number"],["pycbf detector object"]],


# Prelude to the next section of the nuweb doc
```

```
"cbf_construct_goniometer":["""
 cbf_goniometer construct_goniometer(){
    cbf_goniometer goniometer;
    cbf_failnez(cbf_construct_goniometer(self,&goniometer));
    return goniometer;
    }
""","construct_goniometer",[],["pycbf goniometer object"]],

}


class cbfhandlewrapper:
   def __init__(self):
      self.code = """
// Tell SWIG not to make constructor for these objects
%nodefault cbf_handle;
%nodefault cbf_handle_struct;
%nodefault cbf_node;

// A couple of blockitem functions return CBF_NODETYPE
typedef enum
{
  CBF_UNDEFNODE,        /* Undefined */
  CBF_LINK,             /* Link      */
  CBF_ROOT,             /* Root      */
  CBF_DATABLOCK,        /* Datablock */
  CBF_SAVEFRAME,        /* Saveframe */
  CBF_CATEGORY,         /* Category  */
  CBF_COLUMN            /* Column    */
}
CBF_NODETYPE;


// Tell SWIG what the object is, so we can build the class

typedef struct
{
  cbf_node *node;

  int row, search_row;
}  cbf_handle_struct;

typedef cbf_handle_struct *cbf_handle;

typedef cbf_handle_struct handle;
%feature("autodoc","1");

%extend cbf_handle_struct{   // Tell SWIG to attach functions to the structure

    cbf_handle_struct(){  // Constructor
       cbf_handle handle;
       cbf_failnez(cbf_make_handle(&handle));
       return handle;
       }

    ~cbf_handle_struct(){ // Destructor
       cbf_failnez(cbf_free_handle(self));
       }
"""
      self.tail = """
}; // End of cbf_handle_struct
"""
   # End of init function
```

```
def get_code(self):
    return self.code+self.tail
def wrap(self,cfunc,prototype,args,docstring):
    # print "cfunc: ", cfunc
    pyfunc = cfunc.replace("cbf_","")
    # Insert a comment for debugging this script
    code = "\n/* cfunc %s   pyfunc %s  \n"%(cfunc,pyfunc)
    for a in args:
       code += "   arg %s "%(a)
    code += "*/\n\n"
    # Make and free handle are done in the header so skip
    if cfunc.find("cbf_make_handle")>-1 or cfunc.find("cbf_free_handle")>-1:
       # Constructor and destructor done in headers
       return
    if args[0] != "cbf_handle handle": # Must be for cbfhandle
       print "problem",cfunc,pyfunc,args
       return
    if len(args)==1: # Only takes CBFhandle arg
       code+= docstringwrite(pyfunc,[],[],prototype,docstring)
       code+= "    void %s(void){\n"%(pyfunc)
       code+= "      cbf_failnez(%s(self));}\n"%(cfunc)
       self.code=self.code+code
       return
    # Now case by case rather than writing a proper parser
    # Special cases ...
    not_found=0
    try:
        code, pyname, input, output = cbfhandle_specials[cfunc]
        self.code +=  docstringwrite(pyname,input,output,
                                     prototype,docstring)+ code
        return
    except KeyError:
        not_found = 1
        # print "KeyError"
    except ValueError:
        print "problem in",cfunc
        for item in cbfhandle_specials[cfunc]:
           print "***",item
        raise
    if len(args)==2:
       if args[1].find("const char")>-1 and \
          args[1].find("*")>-1          and \
          args[1].find("**")==-1            :
          # 1 input string
          code += docstringwrite(pyfunc,[],["string"],prototype,docstring)
          code += "    void %s(const char* arg){\n"%(pyfunc)
          code +="      cbf_failnez(%s(self,arg));}\n"%(cfunc)
          self.code=self.code+code
          return
       if args[1].find("const char")>-1 and \
          args[1].find("**")>-1                :# return string
          code += docstringwrite(pyfunc,["string"],[],prototype,docstring)
          code += "    const char* %s(void){\n"%(pyfunc)
          code += "    const char* result;\n"
          code += "    cbf_failnez(%s(self, &result));\n"%(cfunc)
          code += "    return result;}\n"
          self.code=self.code+code
          return
       if args[1].find("unsigned int")>-1 and args[1].find("*")==-1:
          # set uint
          if args[1].find("reserved")>-1:
             raise Exception("Setting reserved??? %s %s %s"%(pyfunc,
                                                 cfunc,str(args)))
```

```
            code += docstringwrite(pyfunc,["Integer"],[],prototype,docstring)
            code +="    void %s(unsigned int arg){\n"%(pyfunc)
            code +="      cbf_failnez(%s(self,arg));}\n"%(cfunc)
            self.code=self.code+code
            return
         if args[1].find("unsigned int *")>-1 and args[1].find("**")==-1:
            # output uint
            if args[1].find("reserved")>-1:
               raise Exception("Setting reserved??? %s %s %s"%(pyfunc,
                                                       cfunc,str(args)))
            code += docstringwrite(pyfunc,[],["Integer"],prototype,docstring)
            code +="    unsigned int %s(void){\n"%(pyfunc)
            code +="      unsigned int result;\n"
            code +="      cbf_failnez(%s(self,&result));\n"%(cfunc)
            code +="      return result;}\n"
            self.code=self.code+code
            return
         # For the rest attempt to guess
         if args[1].find("cbf")==-1: # but do not try the goniometer constructor
            if args[1].find("*")>-1 and args[1].find("cbf")==-1:
               # pointer used for returning something
               type = args[1].split(" ")[0]
               code += docstringwrite(pyfunc,[],[type.replace("*","")],
                                                      prototype,docstring)
               code+= "    "+type+" "+pyfunc+"(void){\n"
               code+= "     "+type+" result;\n"
               code+= "       cbf_failnez(%s(self,&result));\n"%(cfunc)
               code+= "       return result;}\n"
               self.code=self.code+code
               return
            else:
               var = args[1].split(" ")[-1]
               code += docstringwrite(pyfunc,[],[args[1]],prototype,docstring)
               code+= "     void %s(%s){\n"%(pyfunc,args[1])
               code +="        cbf_failnez(%s(self,%s));}\n"%(cfunc,var)
               self.code=self.code+code
               return
      if not_found:
            code+= "     void %s(void){\n"%(pyfunc)
            code +="        cbf_failnez(CBF_NOTIMPLEMENTED);}\n"
            self.code=self.code+code
            print "Have not implemented: cbfhandle.%s"%(pyfunc)
            print "   ",cfunc
            print "    args:"
            for a in args:
                print "       ",a
            print
            return



cbf_handle_wrapper = cbfhandlewrapper()


cbf_goniometer_specials = {
"cbf_get_rotation_range":["""
%apply double *OUTPUT {double *start,double *increment};

    void get_rotation_range(double *start,double *increment){
       unsigned int reserved;
       reserved = 0;
       cbf_failnez(cbf_get_rotation_range (self,reserved, start,increment));
    }
""","get_rotation_range",[],["Float start","Float increment"]],

"cbf_rotate_vector":["""

%apply double *OUTPUT {double *final1, double *final2, double *final3};

    void rotate_vector (double ratio, double initial1,double initial2,
         double initial3, double *final1, double *final2, double *final3){
       unsigned int reserved;
       reserved = 0;
       cbf_failnez(cbf_rotate_vector (self, reserved, ratio, initial1,
         initial2, initial3, final1, final2, final3));
    }
""", "rotate_vector",
 [ "double ratio", "double initial1","double initial2", "double initial3" ] ,
                  [ "double final1"  ,"double final2"  , "double final3" ] ],



"cbf_get_reciprocal":["""
%apply double *OUTPUT {double *reciprocal1,double *reciprocal2,
              double *reciprocal3};

    void get_reciprocal (double ratio,double wavelength,
                         double real1, double real2, double real3,
                         double *reciprocal1,double *reciprocal2,
                         double *reciprocal3){
        unsigned int reserved;
        reserved = 0;
        cbf_failnez(cbf_get_reciprocal(self,reserved, ratio, wavelength,
                         real1, real2, real3,reciprocal1,
                         reciprocal2,reciprocal3));
    }
""", "get_reciprocal",
    ["double ratio","double wavelength",
     "double real1","double real2","double real3"],
    ["double reciprocal1","double reciprocal2", "double reciprocal3" ]],

"cbf_get_rotation_axis":["""
%apply double *OUTPUT {double *vector1,double *vector2, double *vector3};

void get_rotation_axis (double *vector1, double *vector2, double *vector3){
     unsigned int reserved;
     reserved = 0;
     cbf_failnez(cbf_get_rotation_axis (self, reserved,
                                   vector1, vector2, vector3));
    }
""","get_rotation_axis", [] ,
 ["double vector1", "double vector2", "double vector3"] ],

}



class cbfgoniometerwrapper:
   def __init__(self):
      self.code = """
// Tell SWIG not to make constructor for these objects
%nodefault cbf_positioner_struct;
%nodefault cbf_goniometer;
%nodefault cbf_axis_struct;

// Tell SWIG what the object is, so we can build the class
```

```
typedef struct
{
  double matrix [3][4];

  cbf_axis_struct *axis;

  size_t axes;

  int matrix_is_valid, axes_are_connected;
}
cbf_positioner_struct;

typedef cbf_positioner_struct *cbf_goniometer;


%feature("autodoc","1");

%extend cbf_positioner_struct{// Tell SWIG to attach functions to the structure

    cbf_positioner_struct(){  // Constructor
       // DO NOT CONSTRUCT WITHOUT A CBFHANDLE
       cbf_failnez(CBF_ARGUMENT);
       return NULL; /* Should never be executed */
       }

    ~cbf_positioner_struct(){ // Destructor
       cbf_failnez(cbf_free_goniometer(self));
       }
"""
      self.tail = """
}; // End of cbf_positioner
"""
   def wrap(self,cfunc,prototype,args,docstring):
     if cfunc.find("cbf_free_goniometer")>-1:
        return
     try:
        code, pyname, input, output = cbf_goniometer_specials[cfunc]
        self.code +=  docstringwrite(pyname,input,output,
                                     prototype,docstring)+ code
     except KeyError:
        print "TODO: Goniometer:",prototype
   def get_code(self):
     return self.code+self.tail


cbf_goniometer_wrapper = cbfgoniometerwrapper()



cbf_detector_specials = {
"cbf_get_pixel_normal":["""
%apply double *OUTPUT {double *normal1,double *normal2, double *normal3};
   void get_pixel_normal ( double index1, double index2,
                          double *normal1,double *normal2, double *normal3){
       cbf_failnez(cbf_get_pixel_normal(self,
                                    index1,index2,normal1,normal2,normal3));
   }

""","get_pixel_normal",["double index1","double index2"] ,
 ["double normal1","double normal2", "double normal3" ] ],

"cbf_get_pixel_normal_fs":["""
%apply double *OUTPUT {double *normalfast,double *normalslow, double *normal3};
```

```
   void get_pixel_normal_fs ( double indexfast, double indexslow,
                       double *normal1,double *normal2, double *normal3){
       cbf_failnez(cbf_get_pixel_normal_fs(self,
                                 indexfast,indexslow,normal1,normal2,normal3));
   }

""","get_pixel_normal_fs",["double indexfast","double indexslow"] ,
 ["double normal1","double normal2", "double normal3" ] ],


"cbf_get_pixel_normal_sf":["""
%apply double *OUTPUT {double *normalslow,double *normalfast, double *normal3};
   void get_pixel_normal_sf ( double indexslow, double indexfast,
                       double *normal1,double *normal2, double *normal3){
       cbf_failnez(cbf_get_pixel_normal_sf(self,
                                 indexslow,indexfast,normal1,normal2,normal3));
   }

""","get_pixel_normal_sf",["double indexslow","double indexfast"] ,
 ["double normal1","double normal2", "double normal3" ] ],


"cbf_get_pixel_area":["""
%apply double *OUTPUT{double *area,double *projected_area};
    void get_pixel_area(double index1, double index2,
                      double *area,double *projected_area){
       cbf_failnez(cbf_get_pixel_area (self,
                                    index1, index2, area,projected_area));
      }
""","get_pixel_area",["double index1", "double index2"],
     ["double area","double projected_area"] ],


"cbf_get_pixel_area_fs":["""
%apply double *OUTPUT{double *area,double *projected_area};
    void get_pixel_area_fs(double indexfast, double indexslow,
                      double *area,double *projected_area){
       cbf_failnez(cbf_get_pixel_area_fs (self,
                                    indexfast, indexslow, area,projected_area));
      }
""","get_pixel_area_fs",["double indexfast", "double indexslow"],
     ["double area","double projected_area"] ],


"cbf_get_pixel_area_sf":["""
%apply double *OUTPUT{double *area,double *projected_area};
    void get_pixel_area_sf(double indexslow, double indexfast,
                      double *area,double *projected_area){
       cbf_failnez(cbf_get_pixel_area_sf (self,
                                    indexslow, indexfast, area,projected_area));
      }
""","get_pixel_area_sf",["double indexslow", "double indexfast"],
     ["double area","double projected_area"] ],


"cbf_get_detector_distance":["""
%apply double *OUTPUT {double *distance};
 void get_detector_distance (double *distance){
  cbf_failnez(cbf_get_detector_distance(self,distance));
  }
""","get_detector_distance",[],["double distance"]],
"cbf_get_detector_normal":["""
%apply double *OUTPUT {double *normal1, double *normal2, double *normal3};
   void get_detector_normal(double *normal1,
                            double *normal2,
                            double *normal3){
     cbf_failnez(cbf_get_detector_normal(self,
                    normal1, normal2, normal3));
   }
""","get_detector_normal",[],
["double normal1", "double normal2", "double normal3"]],


"cbf_get_pixel_coordinates":["""
%apply double *OUTPUT {double *coordinate1,
         double *coordinate2, double *coordinate3};
   void get_pixel_coordinates(double index1, double index2,
             double *coordinate1,
             double *coordinate2,
             double *coordinate3){
      cbf_failnez(cbf_get_pixel_coordinates(self, index1, index2,
             coordinate1, coordinate2, coordinate3));
   }
""","get_pixel_coordinates",["double index1","double index2"],
["double coordinate1", "double coordinate2", "double coordinate3"] ],


"cbf_get_pixel_coordinates_fs":["""
%apply double *OUTPUT {double *coordinate1,
         double *coordinate2, double *coordinate3};
   void get_pixel_coordinates_fs(double indexfast, double indexslow,
             double *coordinate1,
             double *coordinate2,
             double *coordinate3){
      cbf_failnez(cbf_get_pixel_coordinates_fs(self, indexfast, indexslow, coordinate1, coordinate2, coordinate
   }
""","get_pixel_coordinates_fs",["double indexfast","double indexslow"],
["double coordinate1", "double coordinate2", "double coordinate3"] ],


"cbf_get_pixel_coordinates_sf":["""
%apply double *OUTPUT {double *coordinate1,
         double *coordinate2, double *coordinate3};
   void get_pixel_coordinates_sf(double indexslow, double indexfast,
             double *coordinate1,
             double *coordinate2,
             double *coordinate3){
      cbf_failnez(cbf_get_pixel_coordinates_sf(self, indexslow, indexfast, coordinate1, coordinate2, coordinate
   }
""","get_pixel_coordinates_sf",["double indexslow","double indexfast"],
["double coordinate1", "double coordinate2", "double coordinate3"] ],


"cbf_get_beam_center":["""
%apply double *OUTPUT {double *index1, double *index2,
 double *center1,double *center2};
    void get_beam_center(double *index1, double *index2,
                         double *center1,double *center2){
        cbf_failnez(cbf_get_beam_center(self, index1, index2,
                                        center1, center2));
        }
""","get_beam_center",[],
["double index1", "double index2", "double center1","double center2"]],
"cbf_get_beam_center_fs":["""
%apply double *OUTPUT {double *indexfast, double *indexslow,
 double *centerfast,double *centerslow};
    void get_beam_center_fs(double *indexfast, double *indexslow,
                         double *centerfast,double *centerslow){
        cbf_failnez(cbf_get_beam_center_fs(self, indexfast, indexslow,
                                       centerfast, centerslow));
        }
""","get_beam_center_fs",[],
["double indexfast", "double indexslow", "double centerfast","double centerslow"]],


"cbf_get_beam_center_sf":["""
%apply double *OUTPUT {double *indexslow, double *indexfast,
 double *centerslow,double *centerfast};
    void get_beam_center_sf(double *indexslow, double *indexfast,
                         double *centerslow,double *centerfast){
        cbf_failnez(cbf_get_beam_center_sf(self, indexslow, indexfast,
                                       centerslow, centerfast));
        }
""","get_beam_center_sf",[],
["double indexslow", "double indexfast", "double centerslow","double centerfast"]],


"cbf_set_beam_center":["""
    void set_beam_center(double *indexslow, double *indexfast,
                         double *centerslow,double *centerfast){
        cbf_failnez(cbf_set_beam_center(self, indexslow, indexfast,
                                       centerslow, centerfast));
        }
""","set_beam_center",
["double indexslow", "double indexfast", "double centerslow","double centerfast"],[]],


"cbf_set_beam_center_fs":["""
    void set_beam_center_fs(double *indexfast, double *indexslow,
                         double *centerfast,double *centerslow){
        cbf_failnez(cbf_set_beam_center_fs(self, indexfast, indexslow,
                                       centerfast, centerslow));
        }
""","set_beam_center_fs",
["double indexfast", "double indexslow", "double centerfast","double centerslow"],[]],


"cbf_set_beam_center_sf":["""
    void set_beam_center_sf(double *indexslow, double *indexfast,
                         double *centerslow,double *centerfast){
        cbf_failnez(cbf_set_beam_center_sf(self, indexslow, indexfast,
                                       centerslow, centerfast));
        }
""","set_beam_center_sf",
["double indexslow", "double indexfast", "double centerslow","double centerfast"],[]],


"cbf_set_reference_beam_center":["""
    void set_reference_beam_center(double *indexslow, double *indexfast,
                         double *centerslow,double *centerfast){
        cbf_failnez(cbf_set_reference_beam_center(self, indexslow, indexfast,
                                       centerslow, centerfast));
        }
""","set_reference_beam_center",
["double indexslow", "double indexfast", "double centerslow","double centerfast"],[]],
"cbf_set_reference_beam_center_fs":["""
    void set_reference_beam_center_fs(double *indexfast, double *indexslow,
                         double *centerfast,double *centerslow){
        cbf_failnez(cbf_set_reference_beam_center_fs(self, indexfast, indexslow,
                                        centerfast, centerslow));
        }
""","set_reference_beam_center_fs",
["double indexfast", "double indexslow", "double centerfast","double centerslow"],[]],


"cbf_set_reference_beam_center_sf":["""
    void set_reference_beam_center_sf(double *indexslow, double *indexfast,
                         double *centerslow,double *centerfast){
        cbf_failnez(cbf_set_reference_beam_center_sf(self, indexslow, indexfast,
                                        centerslow, centerfast));
        }
""","set_reference_beam_center_sf",
["double indexslow", "double indexfast", "double centerslow","double centerfast"],[]],


"cbf_get_inferred_pixel_size" : ["""
%apply double *OUTPUT { double *psize } get_inferred_pixel_size;
void get_inferred_pixel_size(unsigned int axis_number, double* psize){
   cbf_failnez(cbf_get_inferred_pixel_size(self, axis_number, psize));
   }
""","get_inferred_pixel_size",["Int axis_number"],["Float pixel size"] ],


"cbf_get_inferred_pixel_size_fs" : ["""
%apply double *OUTPUT { double *psize } get_inferred_pixel_size;
void get_inferred_pixel_size_fs(unsigned int axis_number, double* psize){
   cbf_failnez(cbf_get_inferred_pixel_size_fs(self, axis_number, psize));
   }
""","get_inferred_pixel_size_fs",["Int axis_number"],["Float pixel size"] ],


"cbf_get_inferred_pixel_size_sf" : ["""
%apply double *OUTPUT { double *psize } get_inferred_pixel_size;
void get_inferred_pixel_size_sf(unsigned int axis_number, double* psize){
   cbf_failnez(cbf_get_inferred_pixel_size_sf(self, axis_number, psize));
   }
""","get_inferred_pixel_size_sf",["Int axis_number"],["Float pixel size"] ]


}


class cbfdetectorwrapper:
   def __init__(self):
      self.code = """
// Tell SWIG not to make constructor for these objects
%nodefault cbf_detector_struct;
%nodefault cbf_detector;

// Tell SWIG what the object is, so we can build the class
typedef struct
{
  cbf_positioner positioner;

  double displacement [2], increment [2];
  size_t axes, index [2];
}
cbf_detector_struct;

typedef cbf_detector_struct *cbf_detector;

%feature("autodoc","1");

%extend cbf_detector_struct{// Tell SWIG to attach functions to the structure

    cbf_detector_struct(){  // Constructor
       // DO NOT CONSTRUCT WITHOUT A CBFHANDLE
       cbf_failnez(CBF_ARGUMENT);
       return NULL; /* Should never be executed */
       }

    ~cbf_detector_struct(){ // Destructor
       cbf_failnez(cbf_free_detector(self));
       }
"""
      self.tail = """
}; // End of cbf_detector
"""
   def wrap(self,cfunc,prototype,args,docstring):
     if cfunc.find("cbf_free_detector")>-1:
        return
     try:
        code, pyname, input, output = cbf_detector_specials[cfunc]
        self.code +=  docstringwrite(pyname,input,output,
                                     prototype,docstring)+ code
     except KeyError:
        print "TODO: Detector:",prototype
   def get_code(self):
     return self.code+self.tail


cbf_detector_wrapper = cbfdetectorwrapper()


cbfgeneric_specials = {
"cbf_get_local_integer_byte_order":["""
%cstring_output_allocate_size(char **bo, int *bolen, free(*$1));
  %inline {
  void get_local_integer_byte_order(char **bo, int *bolen) {
        char * byteorder;
        char * bot;
        error_status = cbf_get_local_integer_byte_order(&byteorder);
        *bolen = strlen(byteorder);
        if (!(bot = (char *)malloc(*bolen))) {cbf_failnez(CBF_ALLOC)}
        strncpy(bot,byteorder,*bolen);
        *bo = bot;
  }
  }
""","get_local_integer_byte_order",[],["char **bo", "int *bolen"]],


"cbf_get_local_real_format":["""
%cstring_output_allocate_size(char **rf, int *rflen, free(*$1));
  %inline {
  void get_local_real_format(char **rf, int *rflen) {
        char * real_format;
        char * rft;
        error_status = cbf_get_local_real_format(&real_format);
        *rflen = strlen(real_format);
        if (!(rft = (char *)malloc(*rflen))) {cbf_failnez(CBF_ALLOC)}
        strncpy(rft,real_format,*rflen);
        *rf = rft;
  }
  }
""","get_local_real_format",[],["char **rf", "int *rflen"]],


"cbf_get_local_real_byte_order":["""
%cstring_output_allocate_size(char **bo, int *bolen, free(*$1));
  %inline {
  void get_local_real_byte_order(char **bo, int *bolen) {
        char * byteorder;
        char * bot;
        error_status = cbf_get_local_real_byte_order(&byteorder);
        *bolen = strlen(byteorder);
        if (!(bot = (char *)malloc(*bolen))) {cbf_failnez(CBF_ALLOC)}
        strncpy(bot,byteorder,*bolen);
        *bo = bot;
  }
  }
""","get_local_real_byte_order",[],["char **bo", "int *bolen"]],


"cbf_compute_cell_volume":["""

%apply double *OUTPUT {double *volume};
  %inline {
  void compute_cell_volume(double cell[6], double *volume) {
  cbf_failnez(cbf_compute_cell_volume(cell,volume));
  }
  }
""","compute_cell_volume",["double cell[6]"],["Float volume"]],


"cbf_compute_reciprocal_cell":["""
%apply double *OUTPUT {double *astar, double *bstar, double *cstar,
  double *alphastar, double *betastar, double *gammastar};
  %inline {
  void compute_reciprocal_cell(double cell[6], double *astar, double *bstar, double *cstar,
  double *alphastar, double *betastar, double *gammastar) {
    double rcell[6];
    cbf_failnez(cbf_compute_reciprocal_cell(cell,rcell));
    *astar =      rcell[0];
    *bstar =      rcell[1];
    *cstar =      rcell[2];
    *alphastar =  rcell[3];
    *betastar =   rcell[4];
    *gammastar =  rcell[5];
  }
  }

""","compute_reciprocal_cell",["double cell[6]"],
["Float astar", "Float bstar", "Float cstar", "Float alphastar", "Float betastar", "Float gammastar"] ]

}


class genericwrapper:
   def __init__(self):
       self.code = """
// Start of generic functions
```

```
%feature("autodoc","1");
"""
       self.tail = "// End of generic functions\n"
   def get_code(self):
       return self.code + self.tail
   def wrap(self,cfunc,prototype,args,docstring):
       pyfunc = cfunc.replace("cbf_","")
       # Insert a comment for debugging this script
       code = "\n/* cfunc %s   pyfunc %s  \n"%(cfunc,pyfunc)
       for a in args:
           code += "   arg %s "%(a)
       code += "*/\n\n"
       self.code+=code
       code = ""
       not_found = 0
       try:
           code, pyname, input, output = cbfgeneric_specials[cfunc]
           self.code +=  docstringwrite(pyname,input,output,
                                              prototype,docstring)+ code
           return
       except KeyError:
           not_found = 1
           # print "KeyError"
       except ValueError:
           print "problem in generic",cfunc
           for item in cbfgeneric_specials[cfunc]:
              print "***",item
           raise
       if len(args)==1 and args[0].find("char")>-1 and \
                         args[0].find("**")>-1                :# return string
           # first write the c code and inline it
           code += docstringwrite(pyfunc,[],["string"],prototype,docstring)
           code += "%%inline %%{\n   char* %s(void);\n"%(pyfunc)
           code += "   char* %s(void){\n"%(pyfunc)
           code += "      char *r;\n"
           code += "      error_status = %s(&r);\n"%(cfunc)
           code += "      return r; }\n%}\n"
           # now the thing to wrap is:
           code += "char* %s(void);"%(pyfunc)
           self.code=self.code+code
           return

#        code+= "     void %s(void){\n"%(pyfunc)
#        code +="        cbf_failnez(CBF_NOTIMPLEMENTED);}\n"
#        self.code=self.code+code
       print "Have not implemented:"
       for s in [cfunc, pyfunc] + args:
           print "\t",s
       print
       return


generic_wrapper = genericwrapper()


def generate_wrappers(name_dict):
   names = name_dict.keys()
   for cname in names:
      prototype = name_dict[cname][0]
      docstring = name_dict[cname][1]
      # print "Generate wrappers: ", "::",cname,"::", prototype,"::", docstring
      # Check prototype begins with "int cbf_"
      if prototype.find("int cbf_")!=0:
         print "problem with:",prototype
      # Get arguments from prototypes
      try:
         args = prototype.split("(")[1].split(")")[0].split(",")
         args = [ s.lstrip().rstrip() for s in args ] # strip spaces off ends
         # print "Args: ", args
      except:
         # print cname
         # print prototype
         raise
      if args[0].find("cbf_handle")>=0: # This is for the cbfhandle object
         cbf_handle_wrapper.wrap(cname,prototype,args,docstring)
         if (cname=="cbf_get_unit_cell"):
           cbf_handle_wrapper.wrap("cbf_get_unit_cell_esd",prototype,args,docstring)
         if (cname=="cbf_get_reciprocal_cell"):
           cbf_handle_wrapper.wrap("cbf_get_reciprocal_cell_esd",prototype,args,docstring)
         if (cname=="cbf_set_unit_cell"):
           cbf_handle_wrapper.wrap("cbf_set_unit_cell_esd",prototype,args,docstring)
         if (cname=="cbf_set_reciprocal_cell"):
           cbf_handle_wrapper.wrap("cbf_set_reciprocal_cell_esd",prototype,args,docstring)
         continue
      if args[0].find("cbf_goniometer")>=0: # This is for the cbfgoniometer
         cbf_goniometer_wrapper.wrap(cname,prototype,args,docstring)
         continue
      if args[0].find("cbf_detector")>=0: # This is for the cbfdetector
         cbf_detector_wrapper.wrap(cname,prototype,args,docstring)
         continue
      generic_wrapper.wrap(cname,prototype,args,docstring)


generate_wrappers(name_dict)
open("cbfgoniometerwrappers.i","w").write(cbf_goniometer_wrapper.get_code())
open("cbfdetectorwrappers.i","w").write(cbf_detector_wrapper.get_code())
open("cbfhandlewrappers.i","w").write(cbf_handle_wrapper.get_code())
open("cbfgenericwrappers.i","w").write(generic_wrapper.get_code())

print "End of output from make_pycbf.py"
print "\\end{verbatim}"
```
◇

6 Building python extensions - the setup file

Based on the contents of the makefile for CBFlib we will just pull in all of the library for now. We use the distutils approach.

"setup.py" 58 ≡

```


# Import the things to build python binary extensions

from distutils.core import setup, Extension

# Make our extension module

e = Extension('_pycbf',
              sources = ["pycbf_wrap.c","../src/cbf_simple.c"],
         extra_compile_args=["-g"],
         library_dirs=["../lib/"],
         libraries=["cbf"],
         include_dirs = ["../include"] )

# Build it
setup(name="_pycbf",ext_modules=[e],)
```

◇

7 Building and testing the resulting package

Aim to build and test in one go (so that the source and the binary match!!)

"win32.bat" 59a ≡

```
nuweb pycbf
latex pycbf
nuweb pycbf
latex pycbf
dvipdfm pycbf
nuweb pycbf
C:\python24\python make_pycbf.py > TODO.txt
"C:\program files\swigwin-1.3.31\swig.exe" -python pycbf.i
C:\python24\python setup.py build --compiler=mingw32
copy build\lib.win32-2.4\_pycbf.pyd .
REM C:\python24\python pycbf_test1.py
C:\python24\python pycbf_test2.py
C:\python24\python pycbf_test3.py
C:\python24\lib\pydoc.py -w pycbf
C:\python24\python makeflatascii.py pycbf_ascii_help.txt
◇
```

"linux.sh" 59b ≡

```
nuweb pycbf
latex pycbf
nuweb pycbf
latex pycbf
dvipdfm pycbf
nuweb pycbf
lynx -dump CBFlib.html > CBFlib.txt
python make_pycbf.py
swig -python pycbf.i
python setup.py build
rm _pycbf.so
cp build/lib.linux-i686-2.4/_pycbf.so .
python pycbf_test1.py
python pycbf_test2.py
pydoc -w pycbf
python makeflatascii.py pycbf_ascii_help.txt
◇
```

This still gives bold in the ascii (=sucks)

"makeflatascii.py" 59c ≡

```
import pydoc, pycbf, sys
f = open(sys.argv[1],"w")
pydoc.pager=lambda text: f.write(text)
pydoc.TextDoc.bold = lambda self,text : text
pydoc.help(pycbf)
◇
```

8 Debugging compiled extensions

Since it can be a bit of a pain to see where things go wrong here is a quick recipe for poking around with a debugger:

```
amber $> gdb /bliss/users//blissadm/python/bliss_python/suse82/bin/python
GNU gdb 5.3
Copyright 2002 Free Software Foundation, Inc.
```

```
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i586-suse-linux"...
(gdb) br _PyImport_LoadDynamicModule
Breakpoint 1 at 0x80e4199: file Python/importdl.c, line 28.
```

This is how to get a breakpoint when loading the module

```
(gdb) run
Starting program: /mntdirect/_bliss/users/blissadm/python/bliss_python/suse82/bin/python
[New Thread 16384 (LWP 18191)]
Python 2.4.2 (#3, Feb 17 2006, 09:12:13)
[GCC 3.3 20030226 (prerelease) (SuSE Linux)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pycbf
[Switching to Thread 16384 (LWP 18191)]

Breakpoint 1, _PyImport_LoadDynamicModule (name=0xbfffd280 "_pycbf.so",
    pathname=0xbfffd280 "_pycbf.so", fp=0x819e208) at Python/importdl.c:28
28              if ((m = _PyImport_FindExtension(name, pathname)) != NULL) {
(gdb) finish
Run till exit from #0  _PyImport_LoadDynamicModule (
    name=0xbfffd280 "_pycbf.so", pathname=0xbfffd280 "_pycbf.so", fp=0x819e208)
    at Python/importdl.c:28
load_module (name=0xbfffd710 "_pycbf", fp=0x819e208,
    buf=0xbfffd280 "_pycbf.so", type=3, loader=0x405b44f4)
    at Python/import.c:1678
1678                    break;
Value returned is $1 = (PyObject *) 0x405662fc
(gdb) break cbf_read_file
Breakpoint 2 at 0x407f0508: file ../src/cbf.c, line 221.
(gdb) cont
Continuing.
```

We now have a breakpoint where we wanted inside the dynamically loaded file.

```
>>> o=pycbf.cbf_handle_struct()
>>> o.read_file("../img2cif_packed.cif",pycbf.MSG_DIGEST)

Breakpoint 2, cbf_read_file (handle=0x81f7c08, stream=0x8174f58,
    headers=136281096) at ../src/cbf.c:221
221       if (!handle)
(gdb)
```

Now you can step through the c...

9 Things which are currently missing

This is the to do list. Obviously we could benefit a lot from more extensive testing and checking of the docstrings etc.

```
This output comes from make_pycbf.py which generates the wrappers
End of output from make_pycbf.py
```

10 Testing

Some test programs to see if anything appears to work. Eventually it would be good to write a proper unit test suite.

10.1 Read a file based on cif2cbf.c

This is a pretty ugly translation of the program cif2cbf.c skipping all of the writing parts. It appeared to work with the file img2cif_packed.cif which is built when you build CBFlib, hence that file is hardwired in.

"pycbf_test1.py" 61 ≡

```
import pycbf
object = pycbf.cbf_handle_struct() # FIXME
object.read_file("../img2cif_packed.cif",pycbf.MSG_DIGEST)
object.rewind_datablock()
print "Found",object.count_datablocks(),"blocks"
object.select_datablock(0)
print "Zeroth is named",object.datablock_name()
object.rewind_category()
categories = object.count_categories()
for i in range(categories):
    print "Category:",i,
    object.select_category(i)
    category_name = object.category_name()
    print "Name:",category_name,
    rows=object.count_rows()
    print "Rows:",rows,
    cols = object.count_columns()
    print "Cols:",cols
    loop=1
    object.rewind_column()
    while loop is not 0:
        column_name = object.column_name()
        print "column name \"",column_name,"\"",
        try:
           object.next_column()
        except:
           break
    print
    for j in range(rows):
        object.select_row(j)
        object.rewind_column()
        print "row:",j
        for k in range(cols):
            name=object.column_name()
            print "col:",name,
            object.select_column(k)
            typeofvalue=object.get_typeofvalue()
            print "type:",typeofvalue
            if typeofvalue.find("bnry") > -1:
                print "Found the binary!!",
                s=object.get_integerarray_as_string()
                print type(s)
                print dir(s)
                print len(s)
                try:
                   import Numeric
                   d = Numeric.fromstring(s,Numeric.UInt32)
                   # Hard wired Unsigned Int32
                   print d.shape
                   print d[0:10],d[d.shape[0]/2],d[-1]
                   d=Numeric.reshape(d,(2300,2300))
#                   from matplotlib import pylab
#                   pylab.imshow(d,vmin=0,vmax=1000)
#                   pylab.show()
                except ImportError:
                   print "You need to get Numeric and matplotlib to see the data"
            else:
                value=object.get_value()
                print "Val:",value,i
    print
del(object)
```

```
#
print dir()
#object.free_handle(handle)
◇
```

10.2 Try to test the goniometer and detector

Had some initial difficulties but then downloaded an input cbf file which defines a goniometer and detector. The file was found in the example data which comes with CBFlib.

This test is clearly minimalistic for now - it only checks the objects for apparent existence of a single member function.

"pycbf_test2.py" 62a ≡

```
import pycbf
obj = pycbf.cbf_handle_struct()
obj.read_file("../adscconverted.cbf",0)
obj.select_datablock(0)
g = obj.construct_goniometer()
print "Rotation axis is",g.get_rotation_axis()
d = obj.construct_detector(0)
print "Beam center is",d.get_beam_center()
◇
```

It appears to work - eventually. Surprising

10.3 Test cases for the generics

"pycbf_test3.py" 62b ≡

```
import pycbf, unittest
class GenericTests(unittest.TestCase):

    def test_get_local_integer_byte_order(self):
        self.assertEqual( pycbf.get_local_integer_byte_order(),
                          'little_endian')

    def test_get_local_real_byte_order(self):
        self.assertEqual( pycbf.get_local_real_byte_order() ,
                          'little_endian')

    def test_get_local_real_format(self):
        self.assertEqual( pycbf.get_local_real_format(),
                          'ieee 754-1985')

    def test_compute_cell_volume(self):
        self.assertEqual( pycbf.compute_cell_volume((2.,3.,4.,90.,90.,90.)),
                           24.0)
if __name__=="__main__":
    unittest.main()

◇
```

11 Worked example 1 : xmas beamline + mar ccd detector at the ESRF

Now for the interesting part. We will attempt to actually use pycbf for a real dataprocessing task. Crazy you might think.

The idea is the following - we want to take the header information from some mar ccd files (and eventually also the user or the spec control system) and pass this information into cif headers which can be read by fit2d (etc).

11.1 Reading marccd headers

Some relatively ugly code which parses a c header and then tries to interpret the mar ccd header format.

FIXME : byteswapping and ends???

"xmas/readmarheader.py" 63 ≡

```
#!/usr/bin/env python
import struct

# Convert mar c header file types to python struct module types
mar_c_to_python_struct = {
    "INT32"  : "i",
    "UINT32" : "I",
    "char"   : "c",
    "UINT16" : "H"
    }

# Sizes (bytes) of mar c header objects
mar_c_sizes = {
    "INT32"  : 4,
    "UINT32" : 4,
    "char"   : 1,
    "UINT16" : 2
    }

# This was worked out by trial and error from a trial image I think
MAXIMAGES=9



def make_format(cdefinition):
    """
    Reads the header definition in c and makes the format
    string to pass to struct.unpack
    """
    lines = cdefinition.split("\n")
    fmt = ""
    names = []
    expected = 0
    for line in lines:
        if line.find(";")==-1:
            continue
        decl  = line.split(";")[0].lstrip().rstrip()
        try:
            [type, name] = decl.split()
        except:
            #print "skipping:",line
            continue
        #        print "type:",type,"  name:",name

        if name.find("[")>-1:
            # repeated ... times
            try:
                num = name.split("[")[1].split("]")[0]
                num = num.replace("MAXIMAGES",str(MAXIMAGES))
                num = num.replace("sizeof(INT32)","4")
                times = eval(num)
            except:
                print "Please decode",decl
                raise
        else:
            times=1
        try:
            fmt   += mar_c_to_python_struct[type]*times
            names += [name]*times
            expected += mar_c_sizes[type]*times
        except:
            #print "skipping",line
            continue
        #print "%4d %4d"%(mar_c_sizes[type]*times,expected),name,":",times,line
    #print struct.calcsize(fmt),expected
    return names, fmt

def read_mar_header(filename):
    """
    Get the header from a binary file
    """
    f = open(filename,"rb")
    f.seek(1024)
    header=f.read(3072)
    f.close()
    return header


def interpret_header(header, fmt, names):
    """
    given a format and header interpret it
    """
    values = struct.unpack(fmt,header)
    dict = {}
    i=0
    for name in names:
        if dict.has_key(name):
            if type(values[i]) == type("string"):
                 dict[name] = dict[name]+values[i]
            else:
                 try:
                     dict[name].append(values[i])
                 except:
                     dict[name] = [dict[name],values[i]]
        else:
            dict[name] = values[i]
        i=i+1

    return dict


# Now for the c definition (found on mar webpage)
# The following string is therefore copyrighted by Mar I guess

cdefinition = """
typedef struct frame_header_type {
         /* File/header format parameters (256 bytes) */
         UINT32        header_type;      /* flag for header type
                                           (can be  used as magic number) */
         char header_name[16];           /* header name (MMX) */
         UINT32        header_major_version;     /* header_major_version  (n.) */
         UINT32        header_minor_version;     /* header_minor_version  (.n) */
         UINT32        header_byte_order;/* BIG_ENDIAN (Motorola,MIPS);
                                            LITTLE_ENDIAN (DEC, Intel) */
         UINT32        data_byte_order;  /* BIG_ENDIAN (Motorola,MIPS);
                                            LITTLE_ENDIAN (DEC, Intel) */
         UINT32        header_size;      /* in bytes                     */
         UINT32        frame_type;       /* flag for frame type */
         UINT32        magic_number;     /* to be used as a flag -
                                            usually  to indicate new file */
```

```
UINT32       compression_type; /* type of image compression    */
UINT32       compression1;     /* compression parameter 1 */
UINT32       compression2;     /* compression parameter 2 */
UINT32       compression3;     /* compression parameter 3 */
UINT32       compression4;     /* compression parameter 4 */
UINT32       compression5;     /* compression parameter 4 */
UINT32       compression6;     /* compression parameter 4 */
UINT32       nheaders;         /* total number of headers      */
UINT32       nfast;            /* number of pixels in one line */
UINT32       nslow;            /* number of lines in image     */
UINT32       depth;            /* number of bytes per pixel    */
UINT32       record_length;    /* number of pixels between
                                  succesive rows */
UINT32       signif_bits;      /* true depth of data, in bits  */
UINT32       data_type;        /* (signed,unsigned,float...) */
UINT32       saturated_value;  /* value marks pixel as saturated */
UINT32       sequence;         /* TRUE or FALSE */
UINT32       nimages;          /* total number of images - size of
                                  each is nfast*(nslow/nimages) */
UINT32       origin;           /* corner of origin             */
UINT32       orientation;      /* direction of fast axis       */
UINT32       view_direction;   /* direction to view frame      */
UINT32       overflow_location;/* FOLLOWING_HEADER,  FOLLOWING_DATA */
UINT32       over_8_bits;      /* # of pixels with counts  255 */
UINT32       over_16_bits;     /* # of pixels with count  65535 */
UINT32       multiplexed;      /* multiplex flag */
UINT32       nfastimages;      /* # of images in fast direction */
UINT32       nslowimages;      /* # of images in slow direction */
UINT32       background_applied; /* flags correction has been applied -
                                    hold magic number ? */
UINT32       bias_applied;       /* flags correction has been applied -
                                    hold magic number ? */
UINT32       flatfield_applied;  /* flags correction has been applied -
                                    hold magic number ? */
UINT32       distortion_applied; /* flags correction has been applied -
                                    hold magic number ? */
UINT32       original_header_type;     /* Header/frame type from  file
                                          that frame is read from */
UINT32       file_saved;         /* Flag that file has been  saved,
                                    should be zeroed if modified */
char reserve1[(64-40)*sizeof(INT32)-16];

/* Data statistics (128) */
UINT32       total_counts[2];  /* 64 bit integer range = 1.85E19*/
UINT32       special_counts1[2];
UINT32       special_counts2[2];
UINT32       min;
UINT32       max;
UINT32       mean;
UINT32       rms;
UINT32       p10;
UINT32       p90;
UINT32       stats_uptodate;
UINT32       pixel_noise[MAXIMAGES]; /* 1000*base noise value (ADUs) */
char reserve2[(32-13-MAXIMAGES)*sizeof(INT32)];

/* More statistics (256) */
UINT16 percentile[128];


/* Goniostat parameters (128 bytes) */
INT32 xtal_to_detector;  /* 1000*distance in millimeters */
INT32 beam_x;            /* 1000*x beam position (pixels) */
```

```
INT32 beam_y;            /* 1000*y beam position (pixels) */
INT32 integration_time;  /* integration time in  milliseconds */
INT32 exposure_time;     /* exposure time in milliseconds */
INT32 readout_time;      /* readout time in milliseconds */
INT32 nreads;            /* number of readouts to get this  image */
INT32 start_twotheta;    /* 1000*two_theta angle */
INT32 start_omega;       /* 1000*omega angle */
INT32 start_chi;         /* 1000*chi angle */
INT32 start_kappa;       /* 1000*kappa angle */
INT32 start_phi;         /* 1000*phi angle */
INT32 start_delta;       /* 1000*delta angle */
INT32 start_gamma;       /* 1000*gamma angle */
INT32 start_xtal_to_detector; /* 1000*distance in mm (dist in um)*/
INT32 end_twotheta;           /* 1000*two_theta angle */
INT32 end_omega;              /* 1000*omega angle */
INT32 end_chi;                /* 1000*chi angle */
INT32 end_kappa;              /* 1000*kappa angle */
INT32 end_phi;                /* 1000*phi angle */
INT32 end_delta;              /* 1000*delta angle */
INT32 end_gamma;              /* 1000*gamma angle */
INT32 end_xtal_to_detector;   /* 1000*distance in mm (dist in um)*/
INT32 rotation_axis;          /* active rotation axis */
INT32 rotation_range;         /* 1000*rotation angle */
INT32 detector_rotx;          /* 1000*rotation of detector  around X */
INT32 detector_roty;          /* 1000*rotation of detector  around Y */
INT32 detector_rotz;          /* 1000*rotation of detector  around Z */
char reserve3[(32-28)*sizeof(INT32)];

/* Detector parameters (128 bytes) */
INT32 detector_type;            /* detector type */
INT32 pixelsize_x;              /* pixel size (nanometers) */
INT32 pixelsize_y;              /* pixel size (nanometers) */
INT32 mean_bias;                        /* 1000*mean bias value */
INT32 photons_per_100adu;       /* photons / 100 ADUs */
INT32 measured_bias[MAXIMAGES]; /* 1000*mean bias value for each image*/
INT32 measured_temperature[MAXIMAGES];  /* Temperature of each
                                           detector in milliKelvins */
INT32 measured_pressure[MAXIMAGES]; /* Pressure of each  chamber
                                      in microTorr */
/* Retired reserve4 when MAXIMAGES set to 9 from 16 and
   two fields removed, and temp and pressure added
 char reserve4[(32-(5+3*MAXIMAGES))*sizeof(INT32)]
*/

/* X-ray source and optics parameters (128 bytes) */
/* X-ray source parameters (8*4 bytes) */
INT32 source_type;              /* (code) - target, synch. etc */
INT32 source_dx;                /* Optics param. - (size  microns) */
INT32 source_dy;                /* Optics param. - (size  microns) */
INT32 source_wavelength;        /* wavelength  (femtoMeters) */
INT32 source_power;             /* (Watts) */
INT32 source_voltage;           /* (Volts) */
INT32 source_current;           /* (microAmps) */
INT32 source_bias;              /* (Volts) */
INT32 source_polarization_x;    /* () */
INT32 source_polarization_y;    /* () */
char reserve_source[4*sizeof(INT32)];

/* X-ray optics_parameters (8*4 bytes) */
INT32 optics_type;              /* Optics type (code)*/
INT32 optics_dx;                /* Optics param. - (size  microns) */
INT32 optics_dy;                /* Optics param. - (size  microns) */
INT32 optics_wavelength;        /* Optics param. - (size  microns) */
```

```
         INT32 optics_dispersion;        /* Optics param. - (*10E6) */
         INT32 optics_crossfire_x;       /* Optics param. - (microRadians) */
         INT32 optics_crossfire_y;       /* Optics param. - (microRadians) */
         INT32 optics_angle;             /* Optics param. - (monoch.
                                                    2theta - microradians) */
         INT32 optics_polarization_x;    /* () */
         INT32 optics_polarization_y;    /* () */
         char reserve_optics[4*sizeof(INT32)];

         char reserve5[((32-28)*sizeof(INT32))];

         /* File parameters (1024 bytes) */
         char filetitle[128];            /*  Title                  */
         char filepath[128];             /* path name for data  file  */
         char filename[64];              /* name of data  file  */
         char acquire_timestamp[32];     /* date and time of  acquisition */
         char header_timestamp[32];      /* date and time of header  update  */
         char save_timestamp[32];        /* date and time file  saved */
         char file_comments[512];        /* comments, use as desired   */
         char reserve6[1024-(128+128+64+(3*32)+512)];

         /* Dataset parameters (512 bytes) */
         char dataset_comments[512];     /* comments, used as desired   */
         /* pad out to  3072 bytes */
         char pad[3072-(256+128+256+(3*128)+1024+512)];

         } frame_header;
"""



class marheaderreader:
    """
    Class to sit and read a series of images (makes format etc only once)
    """
    def __init__(self):
        """
        Initialise internal stuff
        """
        self.names , self.fmt = make_format(cdefinition)
    def get_header(self,filename):
        """
        Reads a header from file filename
        """
        h=read_mar_header(filename)
        dict = interpret_header(h,self.fmt,self.names)
        # Append ESRF formatted stuff
        items = self.readesrfstring(dict["dataset_comments[512]"])
        for pair in items:
            dict[pair[0]]=pair[1]
        items = self.readesrfstring(dict["file_comments[512]"])
        for pair in items:
            dict[pair[0]]=pair[1]
        dict["pixelsize_x_mm"]= str(float(dict["pixelsize_x"])/1e6)
        dict["pixelsize_y_mm"]= str(float(dict["pixelsize_y"])/1e6)
        dict["integration_time_sec"]= str(float(dict["integration_time"])/1e3)
        dict["beam_y_mm"]= str(float(dict["pixelsize_y_mm"])*
                                     float(dict["beam_y"])/1000.)
        dict["beam_x_mm"]= str(float(dict["pixelsize_x_mm"])*
                                     float(dict["beam_x"])/1000.)

        return dict
    def readesrfstring(self,s):
        """
        Interpret the so called "esrf format" header lines
        which are in comment sections
        """
        s=s.replace("\000","")
        items = filter(None, [len(x)>1 and x or None for x in [
            item.split("=") for item in s.split(";")]])
        return items


if __name__=="__main__":
    """
    Make a little program to process files
    """
    import sys
    print "Starting"
    names,fmt = make_format(cdefinition)
    print "Names and format made"
    h = read_mar_header(sys.argv[1])
    print "Read header, interpreting"
    d = interpret_header(h,fmt,names)
    printed = {}
    for name in names:
        if printed.has_key(name):
            continue
        print name,":",d[name]
        printed[name]=1
```

◇

11.2 Writing out cif files for fit2d/xmas

A script which is supposed to pick up some header information from the mar images, some more infomation from the user and the create cif files.

This relies on a "template" cif file to get it started (avoids me programming everything).

"xmas/xmasheaders.py" 68 ≡

```
    #!/usr/bin/env python


    import pycbf

    # Some cbf helper functions - obj would be a cbf_handle_struct object

    def writewavelength(obj,wavelength):
        obj.set_wavelength(float(wavelength))

    def writecellpar(obj,cifname,value):
        obj.find_category("cell")
        obj.find_column(cifname)
        obj.set_value(value)

    def writecell(obj,cell):
        """
        call with cell = (a,b,c,alpha,beta,gamma)
        """
        obj.find_category("cell")
        obj.find_column("length_a")
        obj.set_value(str(cell[0]))
        obj.find_column("length_b")
        obj.set_value(str(cell[1]))
        obj.find_column("length_c")
        obj.set_value(str(cell[2]))
        obj.find_column("angle_alpha")
        obj.set_value(str(cell[3]))
        obj.find_column("angle_beta")
        obj.set_value(str(cell[4]))
        obj.find_column("angle_gamma")
        obj.set_value(str(cell[5]))

    def writeUB(obj,ub):
        """
        call with ub that can be indexed ub[i][j]
        """
        obj.find_category("diffrn_orient_matrix")
        for i in (1,2,3):
            for j in (1,2,3):
                obj.find_column("UB[%d][%d]"%(i,j))
                obj.set_value(str(ub[i-1][j-1]))

    def writedistance(obj,distance):
        obj.set_axis_setting("DETECTOR_Z",float(distance),0.)


    def writebeam_x_mm(obj,cen):
        obj.set_axis_setting("DETECTOR_X",float(cen),0.)

    def writebeam_y_mm(obj,cen):
        obj.set_axis_setting("DETECTOR_Y",float(cen),0.)

    def writeSPECcmd(obj,s):
        obj.find_category("diffrn_measurement")
        obj.find_column("details")
        obj.set_value(s)

    def writeSPECscan(obj,s):
        obj.find_category("diffrn_scan")
        obj.find_column("id")
        obj.set_value("SCAN%s"%(s))
        obj.find_category("diffrn_scan_axis")
        obj.find_column("scan_id")
        obj.rewind_row()
        for i in range(obj.count_rows()):
            obj.select_row(i)
            obj.set_value("SCAN%s"%(s))
        obj.find_category("diffrn_scan_frame")
        obj.find_column("scan_id")
        obj.rewind_row()
        obj.set_value("SCAN%s"%(s))


    def writepixelsize_y_mm(obj,s):
        """
        Units are mm for cif
        """
        # element number  = assume this is first and only detector
        element_number = 0
        # axis number = faster or slower... ? Need to check precedence ideally...
        obj.find_category("array_structure_list")
        obj.find_column("axis_set_id")
        obj.find_row("ELEMENT_Y")
        obj.find_column("precedence")
        axis_number = obj.get_integervalue()

        obj.set_pixel_size(element_number, axis_number, float(s) )
        obj.find_category("array_structure_list_axis")
        obj.find_column("axis_id")
        obj.find_row("ELEMENT_Y")
        obj.find_column("displacement")
        obj.set_doublevalue("%.6g",float(s)/2.0)
        obj.find_column("displacement_increment")
        obj.set_doublevalue("%.6g",float(s))

    def writepixelsize_x_mm(obj,s):
        # element number  = assume this is first and only detector
        element_number = 0
        # axis number = faster or slower... ? Need to check precedence ideally...
        obj.find_category("array_structure_list")
        obj.find_column("axis_set_id")
        obj.find_row("ELEMENT_X")
        obj.find_column("precedence")
        axis_number = obj.get_integervalue()

        obj.set_pixel_size(element_number, axis_number, float(s) )

        obj.find_category("array_structure_list_axis")
        obj.find_column("axis_id")
        obj.find_row("ELEMENT_X")
        obj.find_column("displacement")
        obj.set_doublevalue("%.6g",float(s)/2.0)
        obj.find_column("displacement_increment")
        obj.set_doublevalue("%.6g",float(s))

    def writeintegrationtime(obj,s):
        obj.find_category("diffrn_scan_frame")
        obj.find_column("integration_time")
        obj.set_value(str(s).replace("\000",""))

    def writenfast(obj,s):
        obj.find_category("array_structure_list")
        obj.find_column("index")
        obj.find_row("1")
        obj.find_column("dimension")
        obj.set_value(str(s))

    def writenslow(obj,s):
        obj.find_category("array_structure_list")
        obj.find_column("index")
        obj.find_row("2")
        obj.find_column("dimension")
        obj.set_value(str(s))


    functiondict = {
        "lambda"   : writewavelength,
        "beam_x_mm"   : writebeam_x_mm,
        "beam_y_mm"   : writebeam_y_mm,
        "distance" : writedistance,
        "UB"       : writeUB,
        "cell"     : writecell,
        "cmd"      : writeSPECcmd,
        "scan"     : writeSPECscan,
        "nfast"    : writenfast,
        "nslow"    : writenslow,
        "pixelsize_y_mm" : writepixelsize_y_mm,
        "pixelsize_x_mm" : writepixelsize_x_mm,
        "integration_time_sec" : writeintegrationtime,
        "tth"      : lambda obj,value : obj.set_axis_setting(
                                "DETECTOR_TWO_THETA_VERTICAL",float(value),0.),
    "chi"      : lambda obj,value : obj.set_axis_setting(
                                    "GONIOMETER_CHI",float(value),0.),
    "th"       : lambda obj,value : obj.set_axis_setting(
                                    "GONIOMETER_THETA",float(value),0.),
    "phi"      : lambda obj,value : obj.set_axis_setting(
                                    "GONIOMETER_PHI",float(value),0.),
    "lc_a"     : lambda obj,value : writecellpar(obj,"length_a",value),
    "lc_b"     : lambda obj,value : writecellpar(obj,"length_b",value),
    "lc_c"     : lambda obj,value : writecellpar(obj,"length_c",value),
    "lc_al"    : lambda obj,value : writecellpar(obj,"angle_alpha",value),
    "lc_be"    : lambda obj,value : writecellpar(obj,"angle_beta",value),
    "lc_ga"    : lambda obj,value : writecellpar(obj,"angle_gamma",value)
    }

"""
    #
    # Not implementing these for now
    lc_ra
    lc_rc 0.4742
    lc_rb 1.16
    energy 13
    cp_phi -180
    alpha 7.3716
    lc_ral 90
    cp_tth -180
    lc_rga 90
    beta 17.572
    omega -2.185
    h 0.21539
    k 0.01957
    l 5.9763
    cp_chi -180
    lc_rbe 90
    cp_th -180
    azimuth 0
"""

# Finally a class for creating header files.
# It reads a template and then offers a processfile command
# for running over a file series

class cifheader:

    def __init__(self,templatefile):
        self.cbf=pycbf.cbf_handle_struct()
        self.cbf.read_template(templatefile)
        from readmarheader import marheaderreader
        self.marheaderreader = marheaderreader()


    def processfile(self,filename, outfile=None,
                    format="mccd",
                    **kwds):
        outfile=outfile.replace(format,"cif")

        if format == "mccd":
            items = self.marheaderreader.get_header(filename)

        if format == "bruker":
            pass
        if format == "edf":
            pass
        self.items=items

        # Take the image header items as default
        self.updateitems(items)

        # Allow them to be overridden
        self.updateitems(kwds)

        # Write the file
        self.writefile(outfile)



    def writefile(self,filename):
        self.cbf.write_file(filename,pycbf.CIF,pycbf.MIME_HEADERS,
                            pycbf.ENC_BASE64)


    def updateitems(self,dict):
        names = dict.keys()
        for name in names:
            value = dict[name]
            # use a dictionary of functions
            if functiondict.has_key(name):
                # print "calling",functiondict[name],value
                apply(functiondict[name],(self.cbf,value))
            else:
                #print "ignoring",name,value
                pass


if __name__=="__main__":
    import sys

    obj=cifheader("xmas_cif_template.cif")

    ub = [[0.11, 0.12, 0.13] , [0.21, 0.22, 0.23], [0.31, 0.32, 0.33]]

    for filename in sys.argv[1:]:
        fileout = filename.split("/")[-1]
        obj.processfile(filename, outfile=fileout, UB=ub, distance=123.456)
◇
```

11.3 A template cif file for the xmas beamline

This was sort of copied and modified from an example file. It has NOT been checked. Hopefully the four circle geometry at least vaguely matches what is at the beamline.

"xmas/xmas_cif_template.cif" 72 ≡

```
###CBF: VERSION 0.6
# CBF file written by cbflib v0.6



data_image_1



loop_
_diffrn.id
_diffrn.crystal_id
 DS1 DIFFRN_CRYSTAL_ID
```

```
loop_
_cell.length_a                   5.959(1)
_cell.length_b                   14.956(1)
_cell.length_c                   19.737(3)
_cell.angle_alpha                90
_cell.angle_beta                 90
_cell.angle_gamma                90


loop_
_diffrn_orient_matrix.id 'DS1'
_diffrn_orient_matrix.type
; reciprocal axis matrix, multiplies hkl vector to generate
  diffractometer xyz vector and diffractometer angles
;
_diffrn_orient_matrix.UB[1][1]            0.11
_diffrn_orient_matrix.UB[1][2]            0.12
_diffrn_orient_matrix.UB[1][3]            0.13
_diffrn_orient_matrix.UB[2][1]            0.21
_diffrn_orient_matrix.UB[2][2]            0.22
_diffrn_orient_matrix.UB[2][3]            0.23
_diffrn_orient_matrix.UB[3][1]            0.31
_diffrn_orient_matrix.UB[3][2]            0.32
_diffrn_orient_matrix.UB[3][3]            0.33




loop_
_diffrn_source.diffrn_id
_diffrn_source.source
_diffrn_source.current
_diffrn_source.type
 DS1 synchrotron 200.0 'XMAS beamline bm28 ESRF'

loop_
_diffrn_radiation.diffrn_id
_diffrn_radiation.wavelength_id
_diffrn_radiation.probe
_diffrn_radiation.monochromator
_diffrn_radiation.polarizn_source_ratio
_diffrn_radiation.polarizn_source_norm
_diffrn_radiation.div_x_source
_diffrn_radiation.div_y_source
_diffrn_radiation.div_x_y_source
_diffrn_radiation.collimation
 DS1 WAVELENGTH1 x-ray 'Si 111' 0.8 0.0 0.08 0.01 0.00 '0.20 mm x 0.20 mm'

loop_
_diffrn_radiation_wavelength.id
_diffrn_radiation_wavelength.wavelength
_diffrn_radiation_wavelength.wt
 WAVELENGTH1 1.73862 1.0

loop_
_diffrn_detector.diffrn_id
_diffrn_detector.id
_diffrn_detector.type
_diffrn_detector.details
_diffrn_detector.number_of_axes
 DS1 MAR 'MAR XMAS' 'slow mode' 5
```

```
loop_
_diffrn_detector_axis.detector_id
_diffrn_detector_axis.axis_id
 MAR DETECTOR_TWO_THETA_VERTICAL
 MAR DETECTOR_X
 MAR DETECTOR_Y
 MAR DETECTOR_Z
 MAR DETECTOR_PITCH

loop_
_diffrn_detector_element.id
_diffrn_detector_element.detector_id
 ELEMENT1 MAR

loop_
_diffrn_data_frame.id
_diffrn_data_frame.detector_element_id
_diffrn_data_frame.array_id
_diffrn_data_frame.binary_id
 FRAME1 ELEMENT1 ARRAY1 1

loop_
_diffrn_measurement.diffrn_id
_diffrn_measurement.id
_diffrn_measurement.number_of_axes
_diffrn_measurement.method
_diffrn_measurement.details
 DS1 GONIOMETER 3 rotation
 'i0=1.000 i1=1.000 i2=1.000 ib=1.000 beamstop=20 mm 0% attenuation'

loop_
_diffrn_measurement_axis.measurement_id
_diffrn_measurement_axis.axis_id
 GONIOMETER GONIOMETER_PHI
 GONIOMETER GONIOMETER_CHI
 GONIOMETER GONIOMETER_THETA


loop_
_diffrn_scan.id
_diffrn_scan.frame_id_start
_diffrn_scan.frame_id_end
_diffrn_scan.frames
 SCAN1 FRAME1 FRAME1 1

loop_
_diffrn_scan_axis.scan_id
_diffrn_scan_axis.axis_id
_diffrn_scan_axis.angle_start
_diffrn_scan_axis.angle_range
_diffrn_scan_axis.angle_increment
_diffrn_scan_axis.displacement_start
_diffrn_scan_axis.displacement_range
_diffrn_scan_axis.displacement_increment
 SCAN1 GONIOMETER_THETA 0.0 0.0 0.0 0.0 0.0 0.0
 SCAN1 GONIOMETER_CHI 0.0 0.0 0.0 0.0 0.0 0.0
 SCAN1 GONIOMETER_PHI 185 1 1 0.0 0.0 0.0
 SCAN1 DETECTOR_TWO_THETA_VERTICAL 0.0 0.0 0.0 0.0 0.0 0.0
 SCAN1 DETECTOR_Z 0.0 0.0 0.0 103.750 0 0
 SCAN1 DETECTOR_Y 0.0 0.0 0.0 0.0 0.0 0.0
 SCAN1 DETECTOR_X 0.0 0.0 0.0 0.0 0.0 0.0
 SCAN1 DETECTOR_PITCH 0.0 0.0 0.0 0.0 0.0 0.0
```

```
loop_
_diffrn_scan_frame.frame_id
_diffrn_scan_frame.frame_number
_diffrn_scan_frame.integration_time
_diffrn_scan_frame.scan_id
_diffrn_scan_frame.date
 FRAME1 1 360 SCAN1 1997-12-04T10:23:48

loop_
_diffrn_scan_frame_axis.frame_id
_diffrn_scan_frame_axis.axis_id
_diffrn_scan_frame_axis.angle
_diffrn_scan_frame_axis.displacement
 FRAME1 GONIOMETER_THETA 0.0 0.0
 FRAME1 GONIOMETER_CHI 0.0 0.0
 FRAME1 GONIOMETER_PHI 185 0.0
 FRAME1 DETECTOR_TWO_THETA_VERTICAL 185 0.0
 FRAME1 DETECTOR_Z 0.0 103.750
 FRAME1 DETECTOR_Y 0.0 0.0
 FRAME1 DETECTOR_X 0.0 0.0
 FRAME1 DETECTOR_PITCH 0.0 0.0

loop_
_axis.id
_axis.type
_axis.equipment
_axis.depends_on
_axis.vector[1]
_axis.vector[2]
_axis.vector[3]
_axis.offset[1]
_axis.offset[2]
_axis.offset[3]
 GONIOMETER_THETA rotation goniometer . 1 0 0 . . .
 GONIOMETER_CHI rotation goniometer GONIOMETER_THETA 0 0 1 . . .
 GONIOMETER_PHI rotation goniometer GONIOMETER_PHI 1 0 0 . . .
 SOURCE general source . 0 0 1 . . .
 GRAVITY general gravity . 0 -1 0 . . .
 DETECTOR_TWO_THETA_VERTICAL rotation goniometer . 1 0 0 . . .
 DETECTOR_Z translation detector DETECTOR_TWO_THETA_VERTICAL 0 0 -1 0 0 0
 DETECTOR_Y translation detector DETECTOR_Z 0 1 0 0 0 0
 DETECTOR_X translation detector DETECTOR_Y 1 0 0 0 0 0
 DETECTOR_PITCH rotation detector DETECTOR_X 0 1 0 0 0 0
 ELEMENT_X translation detector DETECTOR_PITCH 1 0 0 -94.0032 94.0032 0
 ELEMENT_Y translation detector ELEMENT_X 0 1 0 0 0 0

loop_
_array_structure_list.array_id
_array_structure_list.index
_array_structure_list.dimension
_array_structure_list.precedence
_array_structure_list.direction
_array_structure_list.axis_set_id
 ARRAY1 1 2049 1 increasing ELEMENT_X
 ARRAY1 2 2049 2 increasing ELEMENT_Y

loop_
_array_structure_list_axis.axis_set_id
_array_structure_list_axis.axis_id
_array_structure_list_axis.displacement
_array_structure_list_axis.displacement_increment
 ELEMENT_X ELEMENT_X 0.0408 0.0816
 ELEMENT_Y ELEMENT_Y -0.0408 -0.0816
```

```
loop_
_array_intensities.array_id
_array_intensities.binary_id
_array_intensities.linearity
_array_intensities.gain
_array_intensities.gain_esd
_array_intensities.overload
_array_intensities.undefined_value
 ARRAY1 1 linear 0.30 0.03 65000 0

loop_
_array_structure.id
_array_structure.encoding_type
_array_structure.compression_type
_array_structure.byte_order
 ARRAY1 "signed 32-bit integer" packed little_endian
```

◇