Synopsis Reference Manual

Synopsis Reference Manual

Version 0.13

Table of Contents

Chapter 1. Concepts

Chapter 2. C++ API Reference

Abstract explaining the C++ API reference.

The C++ API

Namespaces

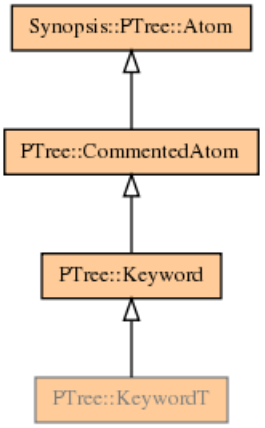
- PTree
- SymbolLookup
- TypeAnalysis

Namespace PTree

Namespaces

- PTree::Kwd

class KeywordT



KeywordT(const Token&)

```
KeywordT(const Token& tk);
```

KeywordT(const char*,size_t)

```
KeywordT(const char* ptr, size_t length);
```

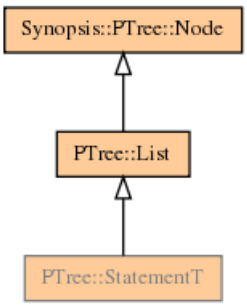
token()const

```
Token::Type token();
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class StatementT



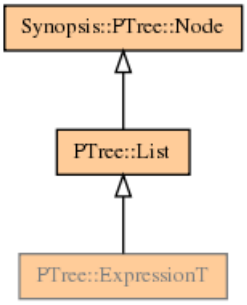
StatementT(Node*,Node*)

```
StatementT(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class ExpressionT



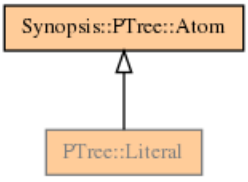
ExpressionT(Node*,Node*)

```
ExpressionT(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class Literal



Literal(const Token&)

```
Literal(const Token& tk);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

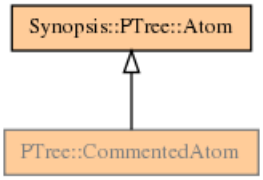
type()const

```
Token::Type type();
```

my_type

```
Token::Type my_type;
```

class CommentedAtom



CommentedAtom(const Token&,Node*)

```
CommentedAtom(const Token& tk, PTree::Node* c = 0);
```

CommentedAtom(const char*,size_t,Node*)

```
CommentedAtom(const char* p, size_t l, PTree::Node* c = 0);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

get_comments()

```
PTree::Node * get_comments();
```

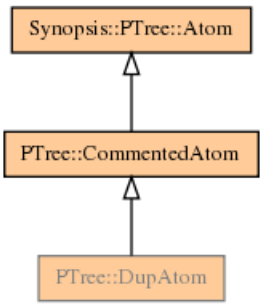
set_comments(Node*)

```
void set_comments(PTree::Node* c);
```

my_comments

```
PTree::Node * my_comments;
```

class DupAtom



DupAtom(const char*,size_t)

```
DupAtom(const char*, size_t);
```

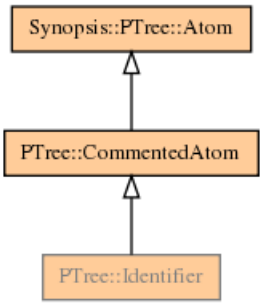
DupAtom(const char*,size_t,const char*,size_t)

```
DupAtom(const char*, size_t, const char*, size_t);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class Identifier



Identifier(const Token&)

```
Identifier(const Token& t);
```

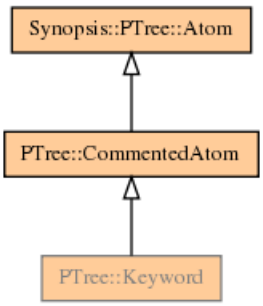
Identifier(const char*,size_t)

```
Identifier(const char* p, size_t l);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class Keyword



Keyword(const Token&)

```
Keyword(const Token& t);
```

Keyword(const char*,int)

```
Keyword(const char* str, int len);
```

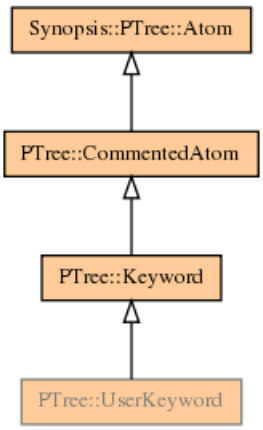
token()const

```
Token::Type token();
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class UserKeyword



UserKeyword(const Token&)

```
UserKeyword(const Token& t);
```

token()const

```
Token::Type token();
```

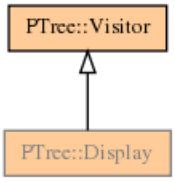
accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

my_type

```
Token::Type my_type;
```

class Display



The Display class provides an annotated view of the ptree, for debugging purposes

Display(std::ostream&,bool)

```
 Display(std::ostream& os, bool encoded);
```

display(const Node*)

```
void display(const PTree::Node*);
```

visit(Atom*)

```
void visit(PTree::Atom*);
```

visit(List*)

```
void visit(PTree::List*);
```

visit(DupAtom*)

```
void visit(PTree::DupAtom*);
```

visit(Brace*)

```
void visit(PTree::Brace*);
```

visit(Block*)

```
void visit(PTree::Block* b);
```

visit(ClassBody*)

```
void visit(PTree::ClassBody* b);
```

visit(Declarator*)

```
void visit(PTree::Declarator* l);
```

visit(Name*)

```
void visit(PTree::Name* l);
```

visit(FstyleCastExpr*)

```
void visit(PTree::FstyleCastExpr* l);
```

newline()

```
void newline();
```

too_deep()

```
bool too_deep();
```

print_encoded(List*)

```
void print_encoded(PTree::List*);
```

my_os

```
 std::ostream & my_os;
```

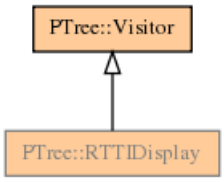
my_indent

```
size_t my_indent;
```

my_encoded

```
bool my_encoded;
```

class RTTIDisplay



RTTIDisplay(std::ostream&,bool)

```
 RTTIDisplay(std::ostream& os, bool encoded);
```

display(const Node*)

```
void display(const PTree::Node*);
```

visit(Atom*)

```
void visit(PTree::Atom*);
```

visit(List*)

```
void visit(PTree::List*);
```

visit(DupAtom*)

```
void visit(PTree::DupAtom*);
```

newline()

```
void newline();
```

my_os

```
 std::ostream & my_os;
```

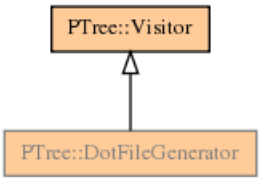
my_indent

```
size_t my_indent;
```

my_encoded

```
bool my_encoded;
```

class DotFileGenerator



DotFileGenerator(std::ostream&)

```
 DotFileGenerator(std::ostream&);
```

write(const Node*)

```
void write(const PTree::Node* ptree);
```

visit(Atom*)

```
void visit(PTree::Atom* a);
```

visit(List*)

```
void visit(PTree::List* l);
```

my_os

```
 std::ostream & my_os;
```

class Encoding

An Encoding represents a mangled (type) name. Here is a quick reference of the grammar:

- *b*: boolean
- *c*: char
- *w*: wchar_t
- *i*: int (signed, unsigned)
- *s*: short (short int)
- *l*: long (long int)
- *j*: long long
- *f*: float
- *d*: double
- *r*: long double
- *v*: void
- *T*: template class (e.g. *Foo<int,char> ==> T[3]Foo[2]ic*. *[2]* means the length of *ic*. It doesn't mean the number of template arguments.
- *e*: ...
- *?*: no return type. the return type of constructors
- *: non-type template parameter
- *S*: *signed*
- *U*: *unsigned*
- *C*: *const*
- *V*: *volatile*
- *P*: pointer
- *R*: reference
- *A*: array (e.g. *char[16] ==> A16_c*)
- *F*: function (e.g. *char foo(int) ==> Fi_c*)
- *M*: pointer to member (e.g. *Type::* ==> M[4]Type*)
- *Q*: qualified class (e.g. *X::YY ==> Q[2][1]X[2]YY*, *::YY ==> Q[2][0][2]YY*)
- *[x]*: means *0x80 + x*
- *0*: means *::* (global scope)

Special function names:

- operator + ==> +
- operator new[] ==> new[]
- operator <type> ==> @<encoded type> cast operator

Code

```
typedef std::basic_string<unsigned char , PTree::Encoding::char_traits> \
Code;
```

iterator

```
typedef Code::const_iterator iterator;
```

struct char_traits

char_type

```
typedef unsigned char  char_type;
```

int_type

```
typedef unsigned long  int_type;
```

pos_type

```
typedef std::streampos pos_type;
```

off_type

```
typedef std::streamoff off_type;
```

state_type

```
typedef std::mbstate_t state_type;
```

assign(char_type&,const char_type&)

```
void assign(PTree::Encoding::char_traits::char_type& c1, const \
PTree::Encoding::char_traits::char_type& c2);
```

eq(const char_type&,const char_type&)

```
bool eq(const PTree::Encoding::char_traits::char_type& c1, const \
PTree::Encoding::char_traits::char_type& c2);
```

lt(const char_type&,const char_type&)

```
bool lt(const PTree::Encoding::char_traits::char_type& c1, const \
PTree::Encoding::char_traits::char_type& c2);
```

compare(const char_type*,const char_type*,std::size_t)

```
int compare(const PTree::Encoding::char_traits::char_type* s1, const \
PTree::Encoding::char_traits::char_type* s2, std::size_t n);
```

length(const char_type*)

```
std::size_t length(const PTree::Encoding::char_traits::char_type* s);
```

find(const char_type*,std::size_t,const char_type&)

```
const PTree::Encoding::char_traits::char_type * find(const \
PTree::Encoding::char_traits::char_type* s, std::size_t n, const \
PTree::Encoding::char_traits::char_type& a);
```

move(char_type*,const char_type*,std::size_t)

```
 PTree::Encoding::char_traits::char_type * \
move(PTree::Encoding::char_traits::char_type* s1, const \
PTree::Encoding::char_traits::char_type* s2, std::size_t n);
```

copy(char_type*,const char_type*,std::size_t)

```
 PTree::Encoding::char_traits::char_type * \
copy(PTree::Encoding::char_traits::char_type* s1, const \
PTree::Encoding::char_traits::char_type* s2, std::size_t n);
```

assign(char_type*,std::size_t,char_type)

```
 PTree::Encoding::char_traits::char_type * \
assign(PTree::Encoding::char_traits::char_type* s, std::size_t n, \
PTree::Encoding::char_traits::char_type a);
```

to_char_type(const int_type&)

```
PTree::Encoding::char_traits::char_type to_char_type(const \
PTree::Encoding::char_traits::int_type& c);
```

to_int_type(const char_type&)

```
PTree::Encoding::char_traits::int_type to_int_type(const \
PTree::Encoding::char_traits::char_type& c);
```

eq_int_type(const int_type&,const int_type&)

```
bool eq_int_type(const PTree::Encoding::char_traits::int_type& c1, \
const PTree::Encoding::char_traits::int_type& c2);
```

eof()

```
PTree::Encoding::char_traits::int_type eof();
```

not_eof(const int_type&)

```
PTree::Encoding::char_traits::int_type not_eof(const \
PTree::Encoding::char_traits::int_type& c);
```

do_init_static()

```
void do_init_static();
```

Encoding()

```
 Encoding();
```

Encoding(const Code&)

```
 Encoding(const PTree::Encoding::Code& b);
```

Encoding(const char*)

```
 Encoding(const char* b);
```

Encoding(const char*,size_t)

```
 Encoding(const char* b, size_t s);
```

Encoding(iterator,iterator)

```
 Encoding(PTree::Encoding::iterator b, PTree::Encoding::iterator e);
```

simple_name(const Atom*)

```
PTree::Encoding simple_name(const PTree::Atom* name);
```

clear()

```
void clear();
```

empty()const

```
bool empty();
```

size()const

```
size_t size();
```

begin()const

```
PTree::Encoding::iterator begin();
```

end()const

```
PTree::Encoding::iterator end();
```

front()const

```
unsigned char  front();
```

at(size_t)const

```
unsigned char  at(size_t i);
```

copy()const

```
const char * copy();
```

return a copy of the underlaying buffer FIXME: this is a temporary workaround while there are still places that use raw strings

operator==(const Encoding&)const

```
bool operator==(const PTree::Encoding& e);
```

operator==(const std::string&)const

```
bool operator==(const std::string& s);
```

operator==(const char*)const

```
bool operator==(const char* s);
```

prepend(unsigned char)

```
void prepend(unsigned char c);
```

prepend(const char*,size_t)

```
void prepend(const char* p, size_t s);
```

prepend(const Encoding&)

```
void prepend(const PTree::Encoding& e);
```

append(unsigned char)

```
void append(unsigned char c);
```

append(const char*,size_t)

```
void append(const char* p, size_t s);
```

append(const Encoding&)

```
void append(const PTree::Encoding& e);
```

append_with_length(const char*,size_t)

```
void append_with_length(const char* s, size_t n);
```

append_with_length(const Encoding&)

```
void append_with_length(const PTree::Encoding& e);
```

pop()

```
unsigned char  pop();
```

pop(size_t)

```
void pop(size_t n);
```

cv_qualify(const Node*,const Node*)

```
void cv_qualify(const PTree::Node*, const PTree::Node* = 0);
```

simple_const()

```
void simple_const();
```

global_scope()

```
void global_scope();
```

simple_name(const Node*)

```
void simple_name(const PTree::Node*);
```

anonymous()

```
void anonymous();
```

template_(const Node*,const Encoding&)

```
void template_(const PTree::Node*, const PTree::Encoding&);
```

qualified(int)

```
void qualified(int);
```

destructor(const Node*)

```
void destructor(const PTree::Node*);
```

ptr_operator(int)

```
void ptr_operator(int);
```

ptr_to_member(const Encoding&,int)

```
void ptr_to_member(const PTree::Encoding&, int);
```

cast_operator(const Encoding&)

```
void cast_operator(const PTree::Encoding&);
```

array()

```
void array();
```

array(unsigned long)

```
void array(unsigned long s);
```

function(const Encoding&)

```
void function(const PTree::Encoding& e);
```

recursion(const Encoding&)

```
void recursion(const PTree::Encoding& e);
```

start_func_args()

```
void start_func_args();
```

end_func_args()

```
void end_func_args();
```

void_()

```
void void_();
```

ellipsis_arg()

```
void ellipsis_arg();
```

no_return_type()

```
void no_return_type();
```

value_temp_param()

```
void value_temp_param();
```

get_scope()const

```
PTree::Encoding get_scope();
```

if this Encoding represents a qualified name, return the name of the outer scope

get_symbol()const

```
PTree::Encoding get_symbol();
```

if this Encoding represents a qualified name, return the name of the symbol inside the outer scope, else return the unmodified name

get_template_arguments()const

```
PTree::Encoding get_template_arguments();
```

unmangled()const

```
std::string unmangled();
```

make_name()

```
 PTree::Node * make_name();
```

make_qname()

```
 PTree::Node * make_qname();
```

make_ptree(Node*)

```
 PTree::Node * make_ptree(PTree::Node*);
```

is_simple_name()const

```
bool is_simple_name();
```

is_global_scope()const

```
bool is_global_scope();
```

is_qualified()const

```
bool is_qualified();
```

is_function()const

```
bool is_function();
```

is_template()const

```
bool is_template();
```

name_to_ptree()

```
 PTree::Node * name_to_ptree();
```

operator<(const Encoding&,const Encoding&)

```
bool operator<(const PTree::Encoding&, const PTree::Encoding&);
```

operator<<(std::ostream&,const Encoding&)

```
 std::ostream & operator<<(std::ostream&, const PTree::Encoding&);
```

bool_t

```
PTree::Node * bool_t;
```

char_t

```
PTree::Node * char_t;
```

wchar_t_t

```
PTree::Node * wchar_t_t;
```

int_t

```
PTree::Node * int_t;
```

short_t

```
PTree::Node * short_t;
```

long_t

```
PTree::Node * long_t;
```

float_t

```
PTree::Node * float_t;
```

double_t

```
PTree::Node * double_t;
```

void_t

```
PTree::Node * void_t;
```

signed_t

```
PTree::Node * signed_t;
```

unsigned_t

```
PTree::Node * unsigned_t;
```

const_t

```
PTree::Node * const_t;
```

volatile_t

```
PTree::Node * volatile_t;
```

operator_name

```
PTree::Node * operator_name;
```

new_operator

```
PTree::Node * new_operator;
```

anew_operator

```
PTree::Node * anew_operator;
```

delete_operator

```
PTree::Node * delete_operator;
```

adelete_operator

```
PTree::Node * adelete_operator;
```

star

```
PTree::Node * star;
```

ampersand

```
PTree::Node * ampersand;
```

comma

```
PTree::Node * comma;
```

dots

```
PTree::Node * dots;
```

scope

```
PTree::Node * scope;
```

tilder

```
PTree::Node * tilder;
```

left_paren

```
PTree::Node * left_paren;
```

right_paren

```
PTree::Node * right_paren;
```

left_bracket

```
PTree::Node * left_bracket;
```

right_bracket

```
PTree::Node * right_bracket;
```

left_angle

```
PTree::Node * left_angle;
```

right_angle

```
PTree::Node * right_angle;
```

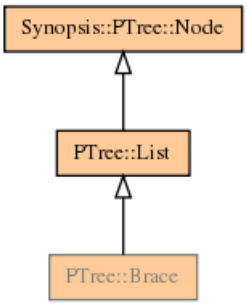
end_of_scope()const

```
PTree::Encoding::iterator end_of_scope();
```

my_buffer

```
PTree::Encoding::Code my_buffer;
```

class Brace



Brace(Node*,Node*)

```
Brace(PTree::Node* p, PTree::Node* q);
```

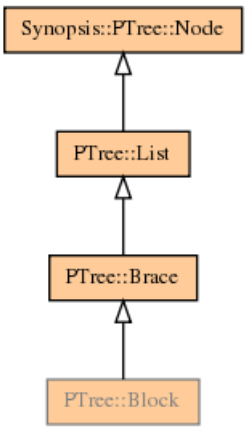
Brace(Node*,Node*,Node*)

```
Brace(PTree::Node* ob, PTree::Node* body, PTree::Node* cb);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class Block



Block(Node*,Node*)

```
Block(PTree::Node* p, PTree::Node* q);
```

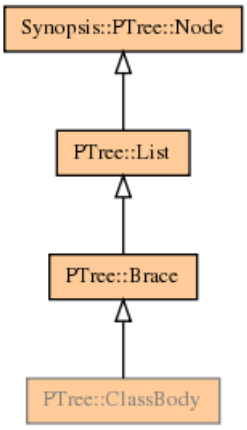
Block(Node*,Node*,Node*)

```
Block(PTree::Node* ob, PTree::Node* bdy, PTree::Node* cb);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class ClassBody



ClassBody(Node*,Node*)

```
ClassBody(PTree::Node* p, PTree::Node* q);
```

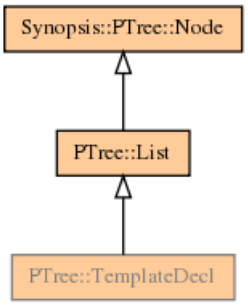
ClassBody(Node*,Node*,Node*)

```
ClassBody(PTree::Node* ob, PTree::Node* bdy, PTree::Node* cb);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class TemplateDecl



TemplateDecl(Node*,Node*)

```
TemplateDecl(PTree::Node* p, PTree::Node* q);
```

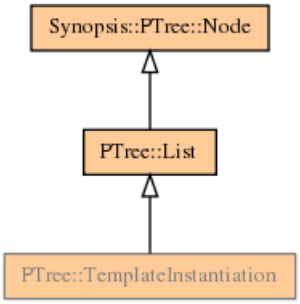
TemplateDecl(Node*)

```
TemplateDecl(PTree::Node* p);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class TemplateInstantiation



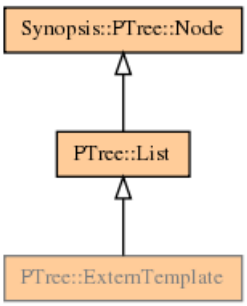
TemplateInstantiation(Node*)

```
TemplateInstantiation(PTree::Node* p);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class ExternTemplate



ExternTemplate(Node*,Node*)

```
ExternTemplate(PTree::Node* p, PTree::Node* q);
```

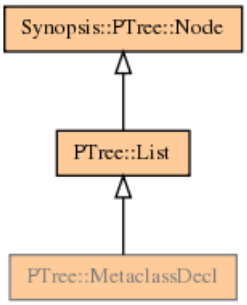
ExternTemplate(Node*)

```
ExternTemplate(PTree::Node* p);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class MetaclassDecl



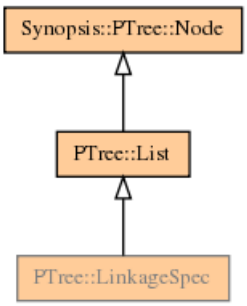
MetaclassDecl(Node*,Node*)

```
MetaclassDecl(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class LinkageSpec



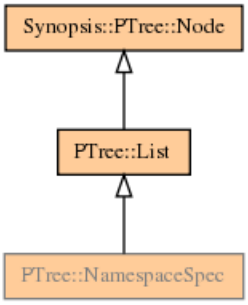
LinkageSpec(Node*,Node*)

```
LinkageSpec(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class NamespaceSpec



NamespaceSpec(Node*,Node*)

```
NamespaceSpec(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

get_comments()

```
PTree::Node * get_comments();
```

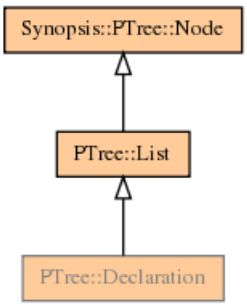
set_comments(Node*)

```
void set_comments(PTree::Node* c);
```

my_comments

```
PTree::Node * my_comments;
```

class Declaration



Declaration(Node*,Node*)

```
Declaration(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

get_comments()

```
PTree::Node * get_comments();
```

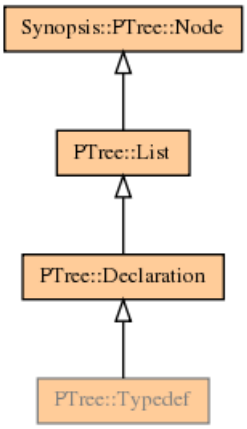
set_comments(Node*)

```
void set_comments(PTree::Node* c);
```

my_comments

```
PTree::Node * my_comments;
```

class Typedef



Typedef(Node*)

```
Typedef(PTree::Node* p);
```

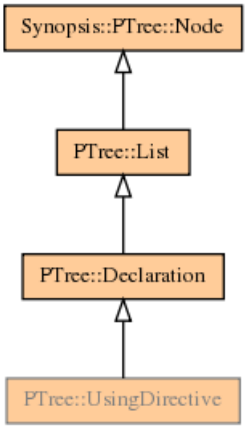
Typedef(Node*,Node*)

```
Typedef(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class UsingDirective



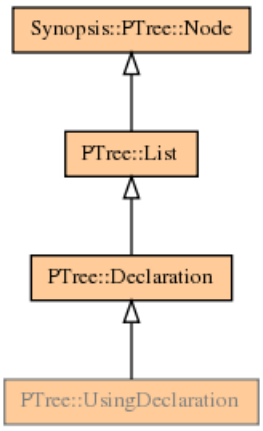
UsingDirective(Node*)

```
UsingDirective(PTree::Node* p);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class UsingDeclaration



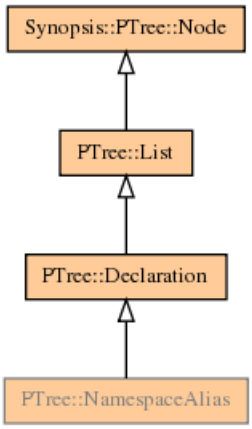
UsingDeclaration(Node*,Node*)

```
UsingDeclaration(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class NamespaceAlias



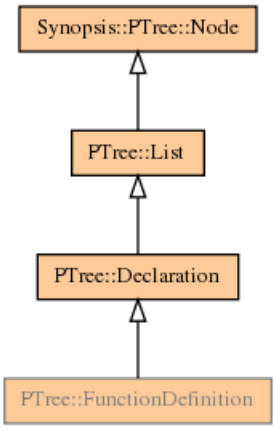
NamespaceAlias(Node*,Node*)

```
NamespaceAlias(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class FunctionDefinition



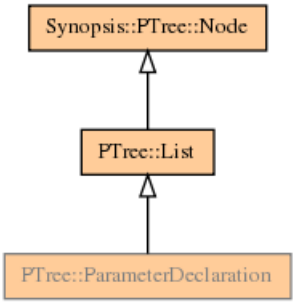
FunctionDefinition(Node*,Node*)

```
FunctionDefinition(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class ParameterDeclaration



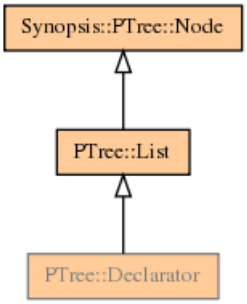
ParameterDeclaration(Node*,Node*,Node*)

```
 ParameterDeclaration(PTree::Node* mod, PTree::Node* type, PTree::Node* \
decl);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class Declarator



Declarator(Node*)

```
 Declarator(PTree::Node*);
```

Declarator(Node*,const Encoding&,const Encoding&,Node*)

```
 Declarator(PTree::Node*, const PTree::Encoding&, const \
PTree::Encoding&, PTree::Node*);
```

Declarator(const Encoding&,const Encoding&,Node*)

```
 Declarator(const PTree::Encoding&, const PTree::Encoding&, \
PTree::Node*);
```

Declarator(Node*,Node*,const Encoding&,const Encoding&,Node*)

```
 Declarator(PTree::Node*, PTree::Node*, const PTree::Encoding&, const \
PTree::Encoding&, PTree::Node*);
```

Declarator(Node*,const Encoding&)

```
Declarator(PTree::Node*, const PTree::Encoding&);
```

Declarator(const Encoding&)

```
Declarator(const PTree::Encoding&);
```

Declarator(Declarator*,Node*,Node*)

```
Declarator(PTree::Declarator*, PTree::Node*, PTree::Node*);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

encoded_type()const

```
PTree::Encoding encoded_type();
```

encoded_name()const

```
PTree::Encoding encoded_name();
```

set_encoded_type(const Encoding&)

```
void set_encoded_type(const PTree::Encoding& t);
```

name()

```
PTree::Node * name();
```

initializer()

```
PTree::Node * initializer();
```

get_comments()

```
PTree::Node * get_comments();
```

set_comments(Node*)

```
void set_comments(PTree::Node* c);
```

my_type

```
PTree::Encoding my_type;
```

my_name

```
PTree::Encoding my_name;
```

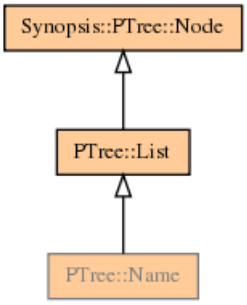
my_declared_name

```
PTree::Node * my_declared_name;
```

my_comments

```
PTree::Node * my_comments;
```

class Name



Name(Node*,const Encoding&)

```
Name(PTree::Node*, const PTree::Encoding&);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

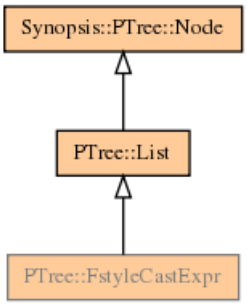
encoded_name()const

```
PTree::Encoding encoded_name();
```

my_name

```
PTree::Encoding my_name;
```

class FstyleCastExpr



FstyleCastExpr(const Encoding&,Node*,Node*)

```
FstyleCastExpr(const PTree::Encoding&, PTree::Node*, PTree::Node*);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

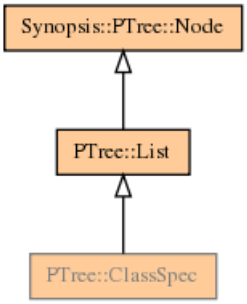
encoded_type()const

```
PTree::Encoding encoded_type();
```

my_type

```
PTree::Encoding my_type;
```

class ClassSpec



ClassSpec(Node*,Node*,Node*)

```
 ClassSpec(PTree::Node*, PTree::Node*, PTree::Node*);
```

ClassSpec(const Encoding&,Node*,Node*,Node*)

```
 ClassSpec(const PTree::Encoding&, PTree::Node*, PTree::Node*, \
PTree::Node*);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

encoded_name()const

```
PTree::Encoding encoded_name();
```

set_encoded_name(const Encoding&)

```
void set_encoded_name(const PTree::Encoding& n);
```

get_comments()

```
 PTree::Node * get_comments();
```

base_clause()const

```
const PTree::Node * base_clause();
```

The list of base classes, i.e. [: [public A] , [public virtual B] ...]

body()

```
PTree::ClassBody * body();
```

The following assumes proper C++, i.e. no OpenC++ extension.

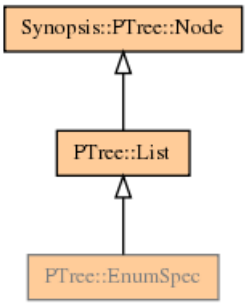
my_name

```
PTree::Encoding my_name;
```

my_comments

```
PTree::Node * my_comments;
```

class EnumSpec



EnumSpec(Node*)

```
EnumSpec(PTree::Node* head);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

encoded_name()const

```
PTree::Encoding encoded_name();
```

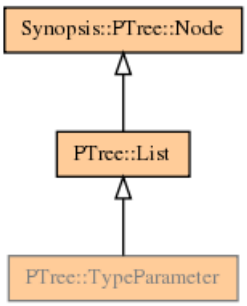
set_encoded_name(const Encoding&)

```
void set_encoded_name(const PTree::Encoding& n);
```

my_name

```
PTree::Encoding my_name;
```

class TypeParameter



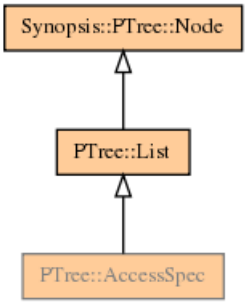
TypeParameter(Node*,Node*)

```
TypeParameter(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class AccessSpec



AccessSpec(Node*,Node*,Node*)

```
AccessSpec(PTree::Node* p, PTree::Node* q, PTree::Node* c);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

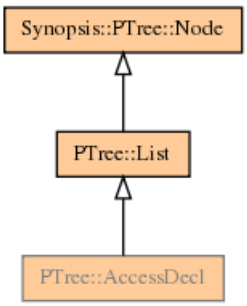
get_comments()

```
PTree::Node * get_comments();
```

my_comments

```
PTree::Node * my_comments;
```

class AccessDecl



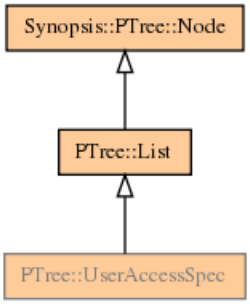
AccessDecl(Node*,Node*)

```
AccessDecl(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class UserAccessSpec



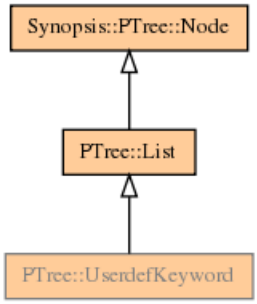
UserAccessSpec(Node*,Node*)

```
UserAccessSpec(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class UserdefKeyword



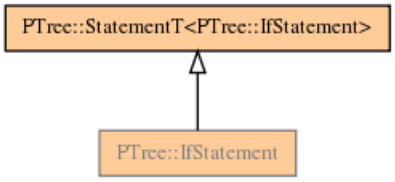
UserdefKeyword(Node*,Node*)

```
UserdefKeyword(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

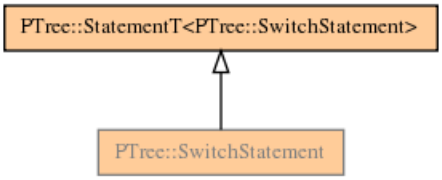
class IfStatement



IfStatement(Node*,Node*)

```
IfStatement(PTree::Node* p, PTree::Node* q);
```

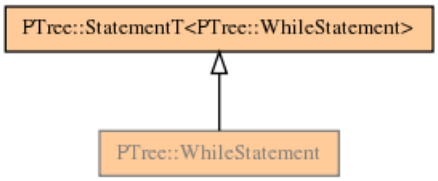
class SwitchStatement



SwitchStatement(Node*,Node*)

```
SwitchStatement(PTree::Node* p, PTree::Node* q);
```

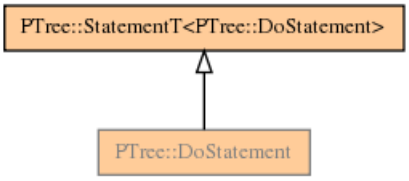
class WhileStatement



WhileStatement(Node*,Node*)

```
WhileStatement(PTree::Node* p, PTree::Node* q);
```

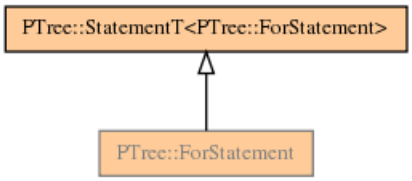
class DoStatement



DoStatement(Node*,Node*)

```
DoStatement(PTree::Node* p, PTree::Node* q);
```

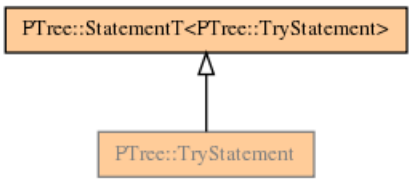
class ForStatement



ForStatement(Node*,Node*)

```
ForStatement(PTree::Node* p, PTree::Node* q);
```

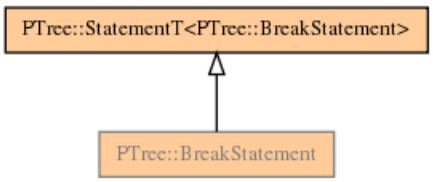
class TryStatement



TryStatement(Node*,Node*)

```
TryStatement(PTree::Node* p, PTree::Node* q);
```

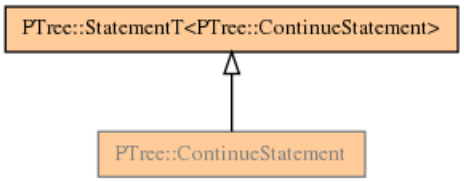
class BreakStatement



BreakStatement(Node*,Node*)

```
BreakStatement(PTree::Node* p, PTree::Node* q);
```

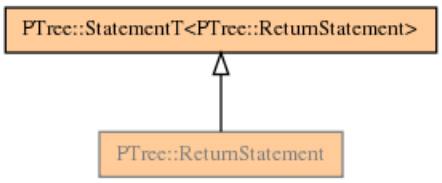
class ContinueStatement



ContinueStatement(Node*,Node*)

```
ContinueStatement(PTree::Node* p, PTree::Node* q);
```

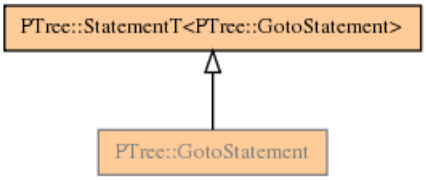
class ReturnStatement



ReturnStatement(Node*,Node*)

```
ReturnStatement(PTree::Node* p, PTree::Node* q);
```

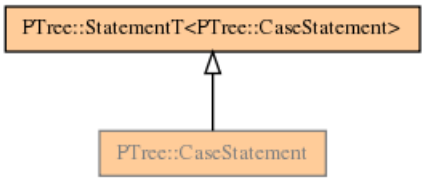
class GotoStatement



GotoStatement(Node*,Node*)

```
GotoStatement(PTree::Node* p, PTree::Node* q);
```

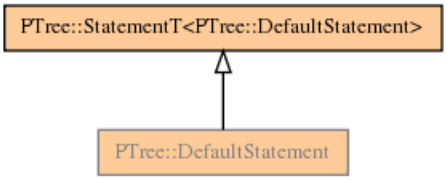
class CaseStatement



CaseStatement(Node*,Node*)

```
CaseStatement(PTree::Node* p, PTree::Node* q);
```

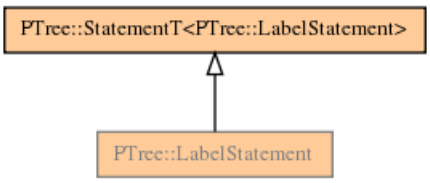
class DefaultStatement



DefaultStatement(Node*,Node*)

```
DefaultStatement(PTree::Node* p, PTree::Node* q);
```

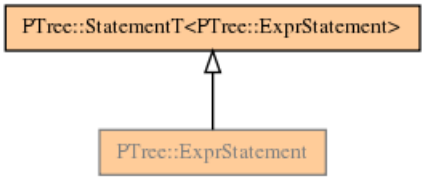
class LabelStatement



LabelStatement(Node*,Node*)

```
LabelStatement(PTree::Node* p, PTree::Node* q);
```

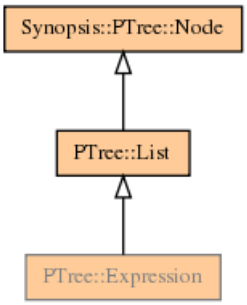
class ExprStatement



ExprStatement(Node*,Node*)

```
ExprStatement(PTree::Node* p, PTree::Node* q);
```

class Expression



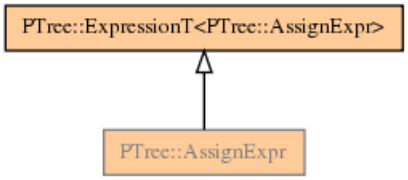
Expression(Node*,Node*)

```
Expression(PTree::Node* p, PTree::Node* q);
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

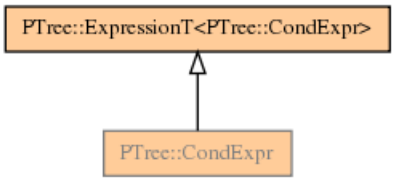
class AssignExpr



AssignExpr(Node*,Node*)

```
AssignExpr(PTree::Node* p, PTree::Node* q);
```

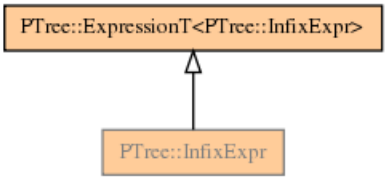
class CondExpr



CondExpr(Node*,Node*)

```
CondExpr(PTree::Node* p, PTree::Node* q);
```

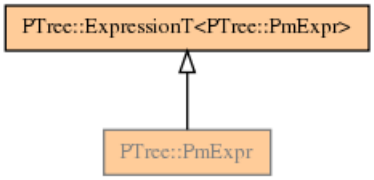
class InfixExpr



InfixExpr(Node*,Node*)

```
InfixExpr(PTree::Node* p, PTree::Node* q);
```

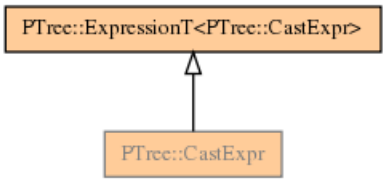
class PmExpr



PmExpr(Node*,Node*)

```
PmExpr(PTree::Node* p, PTree::Node* q);
```

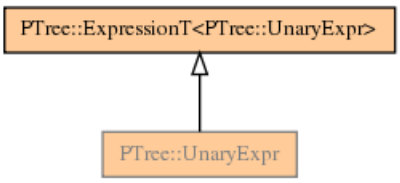
class CastExpr



CastExpr(Node*,Node*)

```
CastExpr(PTree::Node* p, PTree::Node* q);
```

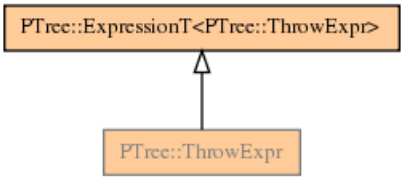
class UnaryExpr



UnaryExpr(Node*,Node*)

```
UnaryExpr(PTree::Node* p, PTree::Node* q);
```

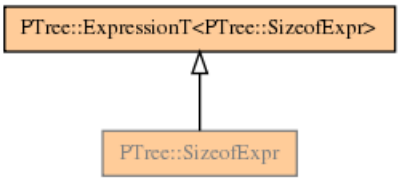
class ThrowExpr



ThrowExpr(Node*,Node*)

```
ThrowExpr(PTree::Node* p, PTree::Node* q);
```

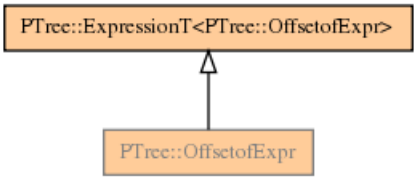
class SizeofExpr



SizeofExpr(Node*,Node*)

```
SizeofExpr(PTree::Node* p, PTree::Node* q);
```

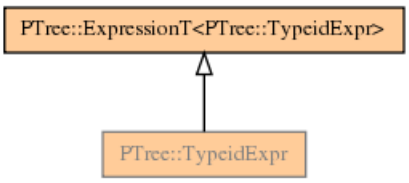
class OffsetofExpr



OffsetofExpr(Node*,Node*)

```
OffsetofExpr(PTree::Node* p, PTree::Node* q);
```

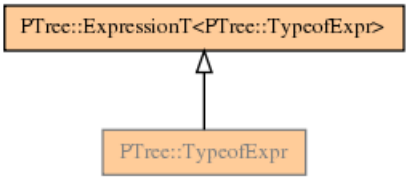
class TypeidExpr



TypeidExpr(Node*,Node*)

```
TypeidExpr(PTree::Node* p, PTree::Node* q);
```

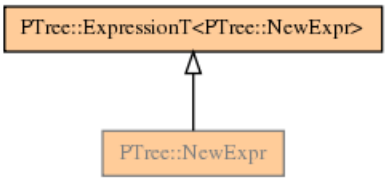
class TypeofExpr



TypeofExpr(Node*,Node*)

```
TypeofExpr(PTree::Node* p, PTree::Node* q);
```

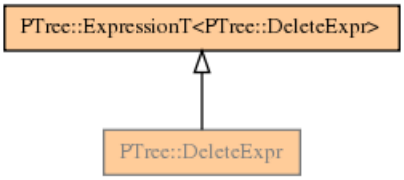
class NewExpr



NewExpr(Node*,Node*)

```
NewExpr(PTree::Node* p, PTree::Node* q);
```

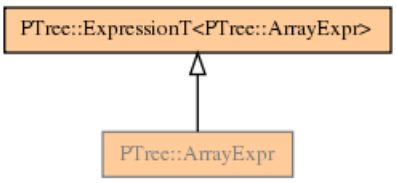
class DeleteExpr



DeleteExpr(Node*,Node*)

```
DeleteExpr(PTree::Node* p, PTree::Node* q);
```

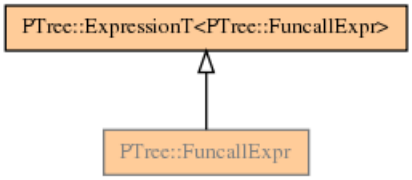
class ArrayExpr



ArrayExpr(Node*,Node*)

```
ArrayExpr(PTree::Node* p, PTree::Node* q);
```

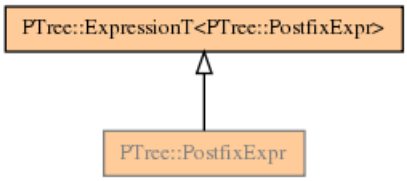
class FuncallExpr



FuncallExpr(Node*,Node*)

```
FuncallExpr(PTree::Node* p, PTree::Node* q);
```

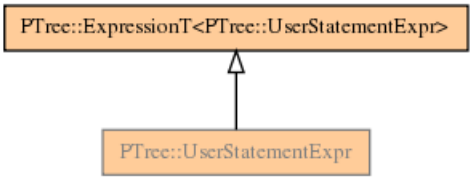
class PostfixExpr



PostfixExpr(Node*,Node*)

```
PostfixExpr(PTree::Node* p, PTree::Node* q);
```

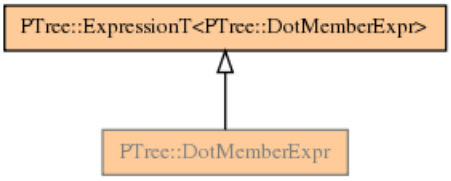
class UserStatementExpr



UserStatementExpr(Node*,Node*)

```
UserStatementExpr(PTree::Node* p, PTree::Node* q);
```

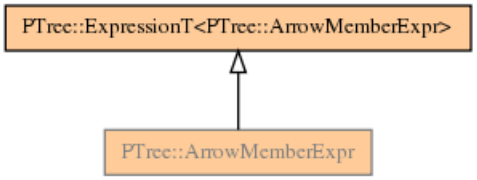
class DotMemberExpr



DotMemberExpr(Node*,Node*)

```
DotMemberExpr(PTree::Node* p, PTree::Node* q);
```

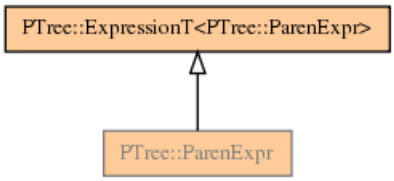
class ArrowMemberExpr



ArrowMemberExpr(Node*,Node*)

```
ArrowMemberExpr(PTree::Node* p, PTree::Node* q);
```

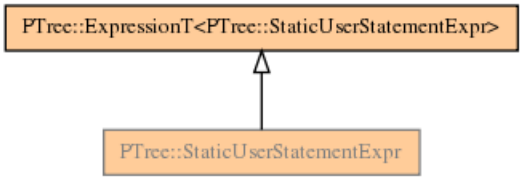
class ParenExpr



ParenExpr(Node*,Node*)

```
ParenExpr(PTree::Node* p, PTree::Node* q);
```

class StaticUserStatementExpr



StaticUserStatementExpr(Node*,Node*)

```
StaticUserStatementExpr(PTree::Node* p, PTree::Node* q);
```

class Node

~Node()

```
 ~Node();
```

is_atom()const

```
bool is_atom();
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

begin()const

```
const char * begin();
```

return the start address of this Ptree in the buffer

end()const

```
const char * end();
```

return the one-past-the-end address of this Ptree in the buffer

position()const

```
const char * position();
```

length()const

```
size_t length();
```

car()const

```
const PTree::Node * car();
```

car()

```
 PTree::Node * car();
```

cdr()const

```
const PTree::Node * cdr();
```

cdr()

```
 PTree::Node * cdr();
```

set_car(Node*)

```
void set_car(PTree::Node* p);
```

set_cdr(Node*)

```
void set_cdr(PTree::Node* p);
```

encoded_type()const

```
PTree::Encoding encoded_type();
```

encoded_name()const

```
PTree::Encoding encoded_name();
```

Node(const char*,size_t)

```
 Node(const char* ptr, size_t len);
```

used by Atom

Node(Node*,Node*)

```
 Node(PTree::Node* p, PTree::Node* q);
```

used by List

union `0106

struct `0107

child

```
 PTree::Node * child;
```

next

```
 PTree::Node * next;
```

struct `0108

position

```
const char * position;
```

length

```
int length;
```

nonleaf

```
PTree::Node::`0106::`0107 nonleaf;
```

leaf

```
PTree::Node::`0106::`0108 leaf;
```

my_data

```
PTree::Node::`0106 my_data;
```

class Iterator

Iterator(Node*)

```
 Iterator(PTree::Node* p);
```

operator()()

```
 PTree::Node * operator()();
```

pop()

```
 PTree::Node * pop();
```

next(Node*&)

```
bool next(PTree::Node*&);
```

reset(Node*)

```
void reset(PTree::Node* p);
```

get()

```
 PTree::Node * get();
```

operator*()

```
 PTree::Node * operator*();
```

operator++()

```
 PTree::Node * operator++();
```

operator++(int)

```
 PTree::Node * operator++(int);
```

empty()

```
bool empty();
```

ptree

```
 PTree::Node * ptree;
```

class Array

Array(size_t)

```
Array(size_t = 8);
```

number()

```
size_t number();
```

operator[](size_t)

```
PTree::Node * & operator[](size_t index);
```

ref(size_t)

```
PTree::Node * & ref(size_t index);
```

append(Node*)

```
void append(PTree::Node*);
```

clear()

```
void clear();
```

all()

```
PTree::Node * all();
```

num

```
size_t num;
```

size

```
size_t size;
```

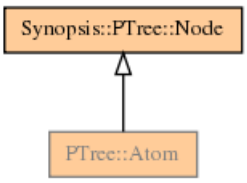
array

```
PTree::Node * * array;
```

default_buf

```
[8] PTree::Node * default_buf;
```

class Atom



Atom(const char*,size_t)

```
Atom(const char* p, size_t l);
```

Atom(const Token&)

```
Atom(const Token& t);
```

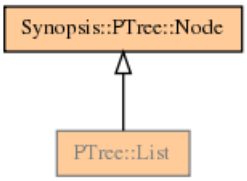
is_atom()const

```
bool is_atom();
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class List



List(Node*,Node*)

```
List(PTree::Node* p, PTree::Node* q);
```

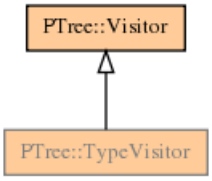
is_atom()const

```
bool is_atom();
```

accept(Visitor*)

```
void accept(PTree::Visitor* visitor);
```

class TypeVisitor



TypeVisitor()

```
 TypeVisitor();
```

type_of(Node*)

```
Token::Type type_of(PTree::Node* node);
```

visit(Literal*)

```
void visit(PTree::Literal*);
```

visit(Identifier*)

```
void visit(PTree::Identifier*);
```

visit(Keyword*)

```
void visit(PTree::Keyword* kwd);
```

visit(Typedef*)

```
void visit(PTree::Typedef*);
```

visit(TemplateDecl*)

```
void visit(PTree::TemplateDecl*);
```

visit(TemplateInstantiation*)

```
void visit(PTree::TemplateInstantiation*);
```

visit(ExternTemplate*)

```
void visit(PTree::ExternTemplate*);
```

visit(MetaclassDecl*)

```
void visit(PTree::MetaclassDecl*);
```

visit(ParameterDeclaration*)

```
void visit(PTree::ParameterDeclaration*);
```

visit(LinkageSpec*)

```
void visit(PTree::LinkageSpec*);
```

visit(NamespaceSpec*)

```
void visit(PTree::NamespaceSpec*);
```

visit(NamespaceAlias*)

```
void visit(PTree::NamespaceAlias*);
```

visit(UsingDirective*)

```
void visit(PTree::UsingDirective*);
```

visit(Declaration*)

```
void visit(PTree::Declaration*);
```

visit(UsingDeclaration*)

```
void visit(PTree::UsingDeclaration*);
```

visit(Declarator*)

```
void visit(PTree::Declarator*);
```

visit(Name*)

```
void visit(PTree::Name*);
```

visit(FstyleCastExpr*)

```
void visit(PTree::FstyleCastExpr*);
```

visit(ClassSpec*)

```
void visit(PTree::ClassSpec*);
```

visit(EnumSpec*)

```
void visit(PTree::EnumSpec*);
```

visit(TypeParameter*)

```
void visit(PTree::TypeParameter*);
```

visit(AccessSpec*)

```
void visit(PTree::AccessSpec*);
```

visit(AccessDecl*)

```
void visit(PTree::AccessDecl*);
```

visit(UserAccessSpec*)

```
void visit(PTree::UserAccessSpec*);
```

visit(IfStatement*)

```
void visit(PTree::IfStatement*);
```

visit(SwitchStatement*)

```
void visit(PTree::SwitchStatement*);
```

visit(WhileStatement*)

```
void visit(PTree::WhileStatement*);
```

visit(DoStatement*)

```
void visit(PTree::DoStatement*);
```

visit(ForStatement*)

```
void visit(PTree::ForStatement*);
```

visit(TryStatement*)

```
void visit(PTree::TryStatement*);
```

visit(BreakStatement*)

```
void visit(PTree::BreakStatement*);
```

visit(ContinueStatement*)

```
void visit(PTree::ContinueStatement*);
```

visit(ReturnStatement*)

```
void visit(PTree::ReturnStatement*);
```

visit(GotoStatement*)

```
void visit(PTree::GotoStatement*);
```

visit(CaseStatement*)

```
void visit(PTree::CaseStatement*);
```

visit(DefaultStatement*)

```
void visit(PTree::DefaultStatement*);
```

visit(LabelStatement*)

```
void visit(PTree::LabelStatement*);
```

visit(ExprStatement*)

```
void visit(PTree::ExprStatement*);
```

visit(Expression*)

```
void visit(PTree::Expression*);
```

visit(AssignExpr*)

```
void visit(PTree::AssignExpr*);
```

visit(CondExpr*)

```
void visit(PTree::CondExpr*);
```

visit(InfixExpr*)

```
void visit(PTree::InfixExpr*);
```

visit(PmExpr*)

```
void visit(PTree::PmExpr*);
```

visit(CastExpr*)

```
void visit(PTree::CastExpr*);
```

visit(UnaryExpr*)

```
void visit(PTree::UnaryExpr*);
```

visit(ThrowExpr*)

```
void visit(PTree::ThrowExpr*);
```

visit(SizeofExpr*)

```
void visit(PTree::SizeofExpr*);
```

visit(TypeidExpr*)

```
void visit(PTree::TypeidExpr*);
```

visit(TypeofExpr*)

```
void visit(PTree::TypeofExpr*);
```

visit(NewExpr*)

```
void visit(PTree::NewExpr*);
```

visit(DeleteExpr*)

```
void visit(PTree::DeleteExpr*);
```

visit(ArrayExpr*)

```
void visit(PTree::ArrayExpr*);
```

visit(FuncallExpr*)

```
void visit(PTree::FuncallExpr*);
```

visit(PostfixExpr*)

```
void visit(PTree::PostfixExpr*);
```

visit(DotMemberExpr*)

```
void visit(PTree::DotMemberExpr*);
```

visit(ArrowMemberExpr*)

```
void visit(PTree::ArrowMemberExpr*);
```

visit(ParenExpr*)

```
void visit(PTree::ParenExpr*);
```

my_type

```
Token::Type my_type;
```

class Visitor

The Visitor class is used to dynamically resolve type information about a given Node. The default implementation does nothing, so you only need to implement the methods you actually need. Any types for which no corresponding 'visit' methods exist will be caught by the 'visit' of the closest parent.

~Visitor()

```
 ~Visitor();
```

visit(Node*)

```
void visit(PTree::Node*);
```

visit(Atom*)

```
void visit(PTree::Atom*);
```

visit(List*)

```
void visit(PTree::List*);
```

visit(Literal*)

```
void visit(PTree::Literal*);
```

visit(CommentedAtom*)

```
void visit(PTree::CommentedAtom*);
```

visit(DupAtom*)

```
void visit(PTree::DupAtom*);
```

visit(Identifier*)

```
void visit(PTree::Identifier*);
```

visit(Keyword*)

```
void visit(PTree::Keyword*);
```

visit(Kwd::Auto*)

```
void visit(PTree::Kwd::Auto*);
```

visit(Kwd::Break*)

```
void visit(PTree::Kwd::Break*);
```

visit(Kwd::Bool*)

```
void visit(PTree::Kwd::Bool*);
```

visit(Kwd::Case*)

```
void visit(PTree::Kwd::Case*);
```

visit(Kwd::Catch*)

```
void visit(PTree::Kwd::Catch*);
```

visit(Kwd::Char*)

```
void visit(PTree::Kwd::Char*);
```

visit(Kwd::Class*)

```
void visit(PTree::Kwd::Class*);
```

visit(Kwd::Continue*)

```
void visit(PTree::Kwd::Continue*);
```

visit(Kwd::Const*)

```
void visit(PTree::Kwd::Const*);
```

visit(Kwd::Default*)

```
void visit(PTree::Kwd::Default*);
```

visit(Kwd::Delete*)

```
void visit(PTree::Kwd::Delete*);
```

visit(Kwd::Double*)

```
void visit(PTree::Kwd::Double*);
```

visit(Kwd::Do*)

```
void visit(PTree::Kwd::Do*);
```

visit(Kwd::Else*)

```
void visit(PTree::Kwd::Else*);
```

visit(Kwd::Extern*)

```
void visit(PTree::Kwd::Extern*);
```

visit(Kwd::Float*)

```
void visit(PTree::Kwd::Float*);
```

visit(Kwd::For*)

```
void visit(PTree::Kwd::For*);
```

visit(Kwd::Friend*)

```
void visit(PTree::Kwd::Friend*);
```

visit(Kwd::Goto*)

```
void visit(PTree::Kwd::Goto*);
```

visit(Kwd::Inline*)

```
void visit(PTree::Kwd::Inline*);
```

visit(Kwd::If*)

```
void visit(PTree::Kwd::If*);
```

visit(Kwd::Int*)

```
void visit(PTree::Kwd::Int*);
```

visit(Kwd::Long*)

```
void visit(PTree::Kwd::Long*);
```

visit(Kwd::Mutable*)

```
void visit(PTree::Kwd::Mutable*);
```

visit(Kwd::Namespace*)

```
void visit(PTree::Kwd::Namespace*);
```

visit(Kwd::New*)

```
void visit(PTree::Kwd::New*);
```

visit(Kwd::Operator*)

```
void visit(PTree::Kwd::Operator*);
```

visit(Kwd::Private*)

```
void visit(PTree::Kwd::Private*);
```

visit(Kwd::Protected*)

```
void visit(PTree::Kwd::Protected*);
```

visit(Kwd::Public*)

```
void visit(PTree::Kwd::Public*);
```

visit(Kwd::Register*)

```
void visit(PTree::Kwd::Register*);
```

visit(Kwd::Return*)

```
void visit(PTree::Kwd::Return*);
```

visit(Kwd::Short*)

```
void visit(PTree::Kwd::Short*);
```

visit(Kwd::Signed*)

```
void visit(PTree::Kwd::Signed*);
```

visit(Kwd::Static*)

```
void visit(PTree::Kwd::Static*);
```

visit(Kwd::Struct*)

```
void visit(PTree::Kwd::Struct*);
```

visit(Kwd::Switch*)

```
void visit(PTree::Kwd::Switch*);
```

visit(Kwd::Template*)

```
void visit(PTree::Kwd::Template*);
```

visit(Kwd::This*)

```
void visit(PTree::Kwd::This*);
```

visit(Kwd::Throw*)

```
void visit(PTree::Kwd::Throw*);
```

visit(Kwd::Try*)

```
void visit(PTree::Kwd::Try*);
```

visit(Kwd::Typedef*)

```
void visit(PTree::Kwd::Typedef*);
```

visit(Kwd::Typename*)

```
void visit(PTree::Kwd::Typename*);
```

visit(Kwd::Union*)

```
void visit(PTree::Kwd::Union*);
```

visit(Kwd::Unsigned*)

```
void visit(PTree::Kwd::Unsigned*);
```

visit(Kwd::Using*)

```
void visit(PTree::Kwd::Using*);
```

visit(Kwd::Virtual*)

```
void visit(PTree::Kwd::Virtual*);
```

visit(Kwd::Void*)

```
void visit(PTree::Kwd::Void*);
```

visit(Kwd::Volatile*)

```
void visit(PTree::Kwd::Volatile*);
```

visit(Kwd::WChar*)

```
void visit(PTree::Kwd::WChar*);
```

visit(Kwd::While*)

```
void visit(PTree::Kwd::While*);
```

visit(Brace*)

```
void visit(PTree::Brace*);
```

[{ [<statement>*] }]

visit(Block*)

```
void visit(PTree::Block*);
```

[{ [<statement>*] }]

visit(ClassBody*)

```
void visit(PTree::ClassBody*);
```

visit(Typedef*)

```
void visit(PTree::Typedef*);
```

visit(TemplateDecl*)

```
void visit(PTree::TemplateDecl*);
```

[template < [types] > [decl]]

visit(TemplateInstantiation*)

```
void visit(PTree::TemplateInstantiation*);
```

visit(ExternTemplate*)

```
void visit(PTree::ExternTemplate*);
```

visit(MetaclassDecl*)

```
void visit(PTree::MetaclassDecl*);
```

visit(LinkageSpec*)

```
void visit(PTree::LinkageSpec*);
```

[extern ["C++"] [{ body }]]

visit(NamespaceSpec*)

```
void visit(PTree::NamespaceSpec*);
```

[namespace <identifier> [{ body }]]

visit(UsingDirective*)

```
void visit(PTree::UsingDirective*);
```

[using namespace Foo ;]

visit(Declaration*)

```
void visit(PTree::Declaration*);
```

One of:

- Variables: [[modifiers] name [declarators] ;]
- Function: prototype: [[modifiers] name [declarators] ;]
- Typedef: ?
- Class definition: [[modifiers] [class foo ...] [declarators]? ;]

visit(NamespaceAlias*)

```
void visit(PTree::NamespaceAlias*);
```

[namespace Foo = Bar ;]

visit(FunctionDefinition*)

```
void visit(PTree::FunctionDefinition*);
```

Function definition: [[modifiers] name declarator [{ ... }]]

visit(ParameterDeclaration*)

```
void visit(PTree::ParameterDeclaration*);
```

One of:

- [decl-specifier-seq]
- [decl-specifier-seq declarator]
- [decl-specifier-seq declarator = assignment-expression]

- [decl-specifier-seq abstract-declarator]
- [decl-specifier-seq abstract-declarator = assignment-expression]
- [decl-specifier-seq = assignment-expression]

visit(UsingDeclaration*)

```
void visit(PTree::UsingDeclaration*);
```

[using Foo *::* x ;]

visit(Declarator*)

```
void visit(PTree::Declarator*);
```

[[declarator { = <expr> }] , ...]

visit(Name*)

```
void visit(PTree::Name*);
```

visit(FstyleCastExpr*)

```
void visit(PTree::FstyleCastExpr*);
```

[[type] ([expr])]

visit(ClassSpec*)

```
void visit(PTree::ClassSpec*);
```

visit(EnumSpec*)

```
void visit(PTree::EnumSpec*);
```

[enum [name] [{ [name [= value]]* }]]

visit(TypeParameter*)

```
void visit(PTree::TypeParameter*);
```

One of:

- [typename]
- [typename identifier]
- [typename identifier = type-id]

visit(AccessSpec*)

```
void visit(PTree::AccessSpec*);
```

visit(AccessDecl*)

```
void visit(PTree::AccessDecl*);
```

visit(UserAccessSpec*)

```
void visit(PTree::UserAccessSpec*);
```

visit(IfStatement*)

```
void visit(PTree::IfStatement*);
```

[if (expr) statement (else statement)?]

visit(SwitchStatement*)

```
void visit(PTree::SwitchStatement*);
```

[switch (expr) statement]

visit(WhileStatement*)

```
void visit(PTree::WhileStatement*);
```

[while (expr) statement]

visit(DoStatement*)

```
void visit(PTree::DoStatement*);
```

[do [{ ... }] while ([...]) ;]

visit(ForStatement*)

```
void visit(PTree::ForStatement*);
```

[for (stmt expr ; expr) statement]

visit(TryStatement*)

```
void visit(PTree::TryStatement*);
```

[try [{}] [catch (arg) [{}]]*]

visit(BreakStatement*)

```
void visit(PTree::BreakStatement*);
```

[break ;]

visit(ContinueStatement*)

```
void visit(PTree::ContinueStatement*);
```

visit(ReturnStatement*)

```
void visit(PTree::ReturnStatement*);
```

visit(GotoStatement*)

```
void visit(PTree::GotoStatement*);
```

visit(CaseStatement*)

```
void visit(PTree::CaseStatement*);
```

[case expr : [expr]]

visit(DefaultStatement*)

```
void visit(PTree::DefaultStatement*);
```

[default : [expr]]

visit(LabelStatement*)

```
void visit(PTree::LabelStatement*);
```

visit(ExprStatement*)

```
void visit(PTree::ExprStatement*);
```

visit(Expression*)

```
void visit(PTree::Expression*);
```

[expr (, expr)*]

visit(AssignExpr*)

```
void visit(PTree::AssignExpr*);
```

[left = right]

visit(CondExpr*)

```
void visit(PTree::CondExpr*);
```

visit(InfixExpr*)

```
void visit(PTree::InfixExpr*);
```

[left op right]

visit(PmExpr*)

```
void visit(PTree::PmExpr*);
```

visit(CastExpr*)

```
void visit(PTree::CastExpr*);
```

(type-expr) expr ..type-expr is type encoded

visit(UnaryExpr*)

```
void visit(PTree::UnaryExpr*);
```

[op expr]

visit(ThrowExpr*)

```
void visit(PTree::ThrowExpr*);
```

[throw [expr]]

visit(SizeofExpr*)

```
void visit(PTree::SizeofExpr*);
```

[sizeof ([type [???]])]

visit(OffsetofExpr*)

```
void visit(PTree::OffsetofExpr*);
```

visit(TypeidExpr*)

```
void visit(PTree::TypeidExpr*);
```

visit(TypeofExpr*)

```
void visit(PTree::TypeofExpr*);
```

visit(NewExpr*)

```
void visit(PTree::NewExpr*);
```

visit(DeleteExpr*)

```
void visit(PTree::DeleteExpr*);
```

[delete [expr]]

visit(ArrayExpr*)

```
void visit(PTree::ArrayExpr*);
```

<postfix> [<expr>]

visit(FuncallExpr*)

```
void visit(PTree::FuncallExpr*);
```

[postfix (args)]

visit(PostfixExpr*)

```
void visit(PTree::PostfixExpr*);
```

[expr ++]

visit(DotMemberExpr*)

```
void visit(PTree::DotMemberExpr*);
```

[postfix . name]

visit(ArrowMemberExpr*)

```
void visit(PTree::ArrowMemberExpr*);
```

[postfix -> name]

visit(ParenExpr*)

```
void visit(PTree::ParenExpr*);
```

[(expr)]

class Writer

Writer(std::ostream&)

```
 Writer(std::ostream& os);
```

write(const Node*)

```
unsigned long  write(const PTree::Node*);
```

visit(Atom*)

```
void visit(PTree::Atom*);
```

visit(List*)

```
void visit(PTree::List*);
```

visit(Brace*)

```
void visit(PTree::Brace*);
```

newline()

```
void newline();
```

my_os

```
 std::ostream & my_os;
```

my_indent

```
size_t my_indent;
```

my_lines

```
unsigned long  my_lines;
```

nconc(N*,Node*)

```
 N * nconc(Synopsis::PTree::N* p, PTree::Node* q);
```

snoc(N*,Node*)

```
 N * snoc(PTree::N* p, PTree::Node* q);
```

display(const Node*,std::ostream&,bool,bool)

```
void display(const PTree::Node* node, std::ostream& os, bool encoded \
= false, bool typeinfo = false);
```

Display the given parse tree segment on the given output stream. If 'encoded' is set to 'true', print encoded names / types on appropriate nodes. If 'typeinfo' is set to 'true', print the class names of the nodes.

generate_dot_file(const Node*,std::ostream&)

```
void generate_dot_file(const PTree::Node* node, std::ostream& os);
```

Generate a dot file for the given parse tree segment.

operator<(const Encoding&,const Encoding&)

```
bool operator<(const PTree::Encoding& e1, const PTree::Encoding& e2);
```

operator<<(std::ostream&,const Encoding&)

```
 std::ostream & operator<<(std::ostream& os, const PTree::Encoding& e);
```

operator==(const Node&,char)

```
bool operator==(const PTree::Node& p, char c);
```

operator!=(const Node&,char)

```
bool operator!=(const PTree::Node& p, char c);
```

operator==(const Node&,const char*)

```
bool operator==(const PTree::Node& p, const char* str);
```

operator!=(const Node&,const char*)

```
bool operator!=(const PTree::Node& p, const char* str);
```

operator==(const Node&,const Node&)

```
bool operator==(const PTree::Node& p, const PTree::Node& q);
```

operator!=(const Node&,const Node&)

```
bool operator!=(const PTree::Node& p, const PTree::Node& q);
```

equal(const Node&,const char*,size_t)

```
bool equal(const PTree::Node& p, const char* str, size_t len);
```

equal(const Node*,const Node*)

```
bool equal(const PTree::Node* p, const PTree::Node* q);
```

equiv(const Node*,const Node*)

```
bool equiv(const PTree::Node* p, const PTree::Node* q);
```

last(const Node*)

```
const PTree::Node * last(const PTree::Node*);
```

Return the last cons cell.

last(Node*)

```
 PTree::Node * last(PTree::Node*);
```

Return the last cons cell.

first(const Node*)

```
const PTree::Node * first(const PTree::Node* p);
```

first(Node*)

```
 PTree::Node * first(PTree::Node* p);
```

rest(const Node*)

```
const PTree::Node * rest(const PTree::Node* p);
```

rest(Node*)

```
 PTree::Node * rest(PTree::Node* p);
```

nth(const Node*,size_t)

```
const PTree::Node * nth(const PTree::Node* p, size_t n);
```

nth(Node*,size_t)

```
 PTree::Node * nth(PTree::Node* p, size_t n);
```

tail(const Node*,size_t)

```
const PTree::Node * tail(const PTree::Node* p, size_t k);
```

tail(Node*,size_t)

```
 PTree::Node * tail(PTree::Node* p, size_t k);
```

second(const Node*)

```
const PTree::Node * second(const PTree::Node*);
```

second(Node*)

```
 PTree::Node * second(PTree::Node*);
```

third(const Node*)

```
const PTree::Node * third(const PTree::Node*);
```

third(Node*)

```
 PTree::Node * third(PTree::Node*);
```

length(const Node*)

```
int length(const PTree::Node*);
```

cadr(const Node*)

```
const PTree::Node * cadr(const PTree::Node* p);
```

cadr(Node*)

```
 PTree::Node * cadr(PTree::Node* p);
```

cddr(const Node*)

```
const PTree::Node * cddr(const PTree::Node* p);
```

cddr(Node*)

```
 PTree::Node * cddr(PTree::Node* p);
```

ca_ar(const Node*)

```
const PTree::Node * ca_ar(const PTree::Node*);
```

compute Caa..ar

ca_ar(Node*)

```
 PTree::Node * ca_ar(PTree::Node*);
```

cons(Node*,Node*)

```
 PTree::Node * cons(PTree::Node*, PTree::Node*);
```

list()

```
 PTree::List * list();
```

list(Node*)

```
 PTree::List * list(PTree::Node*);
```

list(Node*,Node*)

```
 PTree::List * list(PTree::Node*, PTree::Node*);
```

list(Node*,Node*,Node*)

```
 PTree::List * list(PTree::Node*, PTree::Node*, PTree::Node*);
```

list(Node*,Node*,Node*,Node*)

```
 PTree::List * list(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*);
```

list(Node*,Node*,Node*,Node*,Node*)

```
 PTree::List * list(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*, PTree::Node*);
```

list(Node*,Node*,Node*,Node*,Node*,Node*)

```
 PTree::List * list(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*, PTree::Node*, PTree::Node*);
```

list(Node*,Node*,Node*,Node*,Node*,Node*,Node*)

```
 PTree::List * list(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*, PTree::Node*, PTree::Node*, PTree::Node*);
```

list(Node*,Node*,Node*,Node*,Node*,Node*,Node*,Node*)

```
 PTree::List * list(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*, PTree::Node*, PTree::Node*, PTree::Node*, PTree::Node*);
```

copy(Node*)

```
 PTree::Node * copy(PTree::Node*);
```

append(Node*,Node*)

```
 PTree::Node * append(PTree::Node*, PTree::Node*);
```

replace_all(Node*,Node*,Node*)

```
 PTree::Node * replace_all(PTree::Node*, PTree::Node*, PTree::Node*);
```

subst(Node*,Node*,Node*)

```
 PTree::Node * subst(PTree::Node*, PTree::Node*, PTree::Node*);
```

subst(Node*,Node*,Node*,Node*,Node*)

```
 PTree::Node * subst(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*, PTree::Node*);
```

subst(Node*,Node*,Node*,Node*,Node*,Node*,Node*)

```
 PTree::Node * subst(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*, PTree::Node*, PTree::Node*, PTree::Node*);
```

shallow_subst(Node*,Node*,Node*)

```
 PTree::Node * shallow_subst(PTree::Node*, PTree::Node*, PTree::Node*);
```

shallow_subst(Node*,Node*,Node*,Node*,Node*)

```
 PTree::Node * shallow_subst(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*, PTree::Node*);
```

shallow_subst(Node*,Node*,Node*,Node*,Node*,Node*,Node*)

```
 PTree::Node * shallow_subst(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*, PTree::Node*, PTree::Node*, PTree::Node*);
```

shallow_subst(Node*,Node*,Node*,Node*,Node*,Node*,Node*,Node*,Node*)

```
 PTree::Node * shallow_subst(PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*, PTree::Node*, PTree::Node*, PTree::Node*, PTree::Node*, \
PTree::Node*);
```

subst_sublist(Node*,Node*,Node*)

```
 PTree::Node * subst_sublist(PTree::Node*, PTree::Node*, PTree::Node*);
```

nconc(Node*,Node*)

```
 PTree::Node * nconc(PTree::Node*, PTree::Node*);
```

nconc(Node*,Node*,Node*)

```
 PTree::Node * nconc(PTree::Node*, PTree::Node*, PTree::Node*);
```

snoc(Node*,Node*)

```
 PTree::Node * snoc(PTree::Node*, PTree::Node*);
```

type_of(const Node*)

```
Token::Type type_of(const PTree::Node* node);
```

is_a(const Node*,Token::Type)

```
bool is_a(const PTree::Node* node, Token::Type t);
```

is_a(const Node*,Token::Type,Token::Type)

```
bool is_a(const PTree::Node* node, Token::Type t1, Token::Type t2);
```

is_a(const Node*,Token::Type,Token::Type,Token::Type)

```
bool is_a(const PTree::Node* node, Token::Type t1, Token::Type t2, \
Token::Type t3);
```

reify(const Node*)

```
std::string reify(const PTree::Node* p);
```

Namespace PTree::Kwd

Auto

```
typedef PTree::KeywordT<Token::AUTO> Auto;
```

Bool

```
typedef PTree::KeywordT<Token::BOOLEAN> Bool;
```

Break

```
typedef PTree::KeywordT<Token::BREAK> Break;
```

Case

```
typedef PTree::KeywordT<Token::CASE> Case;
```

Catch

```
typedef PTree::KeywordT<Token::CATCH> Catch;
```

Char

```
typedef PTree::KeywordT<Token::CHAR> Char;
```

Class

```
typedef PTree::KeywordT<Token::CLASS> Class;
```

Continue

```
typedef PTree::KeywordT<Token::CONTINUE> Continue;
```

Const

```
typedef PTree::KeywordT<Token::CONST> Const;
```

Default

```
typedef PTree::KeywordT<Token::DEFAULT> Default;
```

Delete

```
typedef PTree::KeywordT<Token::DELETE> Delete;
```

Double

```
typedef PTree::KeywordT<Token::DOUBLE> Double;
```

Do

```
typedef PTree::KeywordT<Token::DO> Do;
```

Else

```
typedef PTree::KeywordT<Token::ELSE> Else;
```

Enum

```
typedef PTree::KeywordT<Token::ENUM> Enum;
```

Extern

```
typedef PTree::KeywordT<Token::EXTERN> Extern;
```

Float

```
typedef PTree::KeywordT<Token::FLOAT> Float;
```

For

```
typedef PTree::KeywordT<Token::FOR> For;
```

Friend

```
typedef PTree::KeywordT<Token::FRIEND> Friend;
```

Goto

```
typedef PTree::KeywordT<Token::GOTO> Goto;
```

Inline

```
typedef PTree::KeywordT<Token::INLINE> Inline;
```

If

```
typedef PTree::KeywordT<Token::IF> If;
```

Int

```
typedef PTree::KeywordT<Token::INT> Int;
```

Long

```
typedef PTree::KeywordT<Token::LONG> Long;
```

Mutable

```
typedef PTree::KeywordT<Token::MUTABLE> Mutable;
```

Namespace

```
typedef PTree::KeywordT<Token::NAMESPACE> Namespace;
```

New

```
typedef PTree::KeywordT<Token::NEW> New;
```

Operator

```
typedef PTree::KeywordT<Token::OPERATOR> Operator;
```

Private

```
typedef PTree::KeywordT<Token::PRIVATE> Private;
```

Protected

```
typedef PTree::KeywordT<Token::PROTECTED> Protected;
```

Public

```
typedef PTree::KeywordT<Token::PUBLIC> Public;
```

Register

```
typedef PTree::KeywordT<Token::REGISTER> Register;
```

Return

```
typedef PTree::KeywordT<Token::RETURN> Return;
```

Short

```
typedef PTree::KeywordT<Token::SHORT> Short;
```

Signed

```
typedef PTree::KeywordT<Token::SIGNED> Signed;
```

Static

```
typedef PTree::KeywordT<Token::STATIC> Static;
```

Struct

```
typedef PTree::KeywordT<Token::STRUCT> Struct;
```

Switch

```
typedef PTree::KeywordT<Token::SWITCH> Switch;
```

Template

```
typedef PTree::KeywordT<Token::TEMPLATE> Template;
```

This

```
typedef PTree::KeywordT<Token::THIS> This;
```

Throw

```
typedef PTree::KeywordT<Token::THROW> Throw;
```

Try

```
typedef PTree::KeywordT<Token::TRY> Try;
```

Typedef

```
typedef PTree::KeywordT<Token::TYPEDEF> Typedef;
```

Typename

```
typedef PTree::KeywordT<Token::TYPENAME> Typename;
```

Typeof

```
typedef PTree::KeywordT<Token::TYPEOF> Typeof;
```

Union

```
typedef PTree::KeywordT<Token::UNION> Union;
```

Unsigned

```
typedef PTree::KeywordT<Token::UNSIGNED> Unsigned;
```

Using

```
typedef PTree::KeywordT<Token::USING> Using;
```

Virtual

```
typedef PTree::KeywordT<Token::VIRTUAL> Virtual;
```

Void

```
typedef PTree::KeywordT<Token::VOID> Void;
```

Volatile

```
typedef PTree::KeywordT<Token::VOLATILE> Volatile;
```

WChar

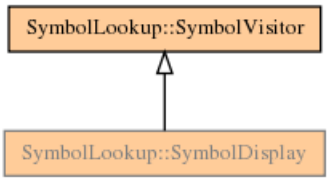
```
typedef PTree::KeywordT<Token::WCHAR> WChar;
```

While

```
typedef PTree::KeywordT<Token::WHILE> While;
```

Namespace SymbolLookup

class SymbolDisplay



SymbolDisplay(std::ostream&,size_t)

```
 SymbolDisplay(std::ostream& os, size_t indent);
```

display(const PTree::Encoding&,const Symbol*)

```
void display(const PTree::Encoding&, const SymbolLookup::Symbol*);
```

prefix(const std::string&)

```
 std::ostream & prefix(const std::string& type);
```

visit(const Symbol*)

```
void visit(const SymbolLookup::Symbol*);
```

visit(const VariableName*)

```
void visit(const SymbolLookup::VariableName*);
```

visit(const ConstName*)

```
void visit(const SymbolLookup::ConstName*);
```

visit(const TypeName*)

```
void visit(const SymbolLookup::TypeName*);
```

visit(const TypedefName*)

```
void visit(const SymbolLookup::TypedefName*);
```

visit(const ClassName*)

```
void visit(const SymbolLookup::ClassName*);
```

visit(const EnumName*)

```
void visit(const SymbolLookup::EnumName*);
```

visit(const ClassTemplateName*)

```
void visit(const SymbolLookup::ClassTemplateName*);
```

visit(const FunctionName*)

```
void visit(const SymbolLookup::FunctionName*);
```

visit(const FunctionTemplateName*)

```
void visit(const SymbolLookup::FunctionTemplateName*);
```

visit(const NamespaceName*)

```
void visit(const SymbolLookup::NamespaceName*);
```

my_os

```
 std::ostream & my_os;
```

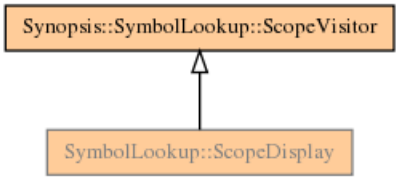
my_indent

```
std::string my_indent;
```

my_name

```
std::string my_name;
```

class ScopeDisplay



The ScopeDisplay class provides an annotated view of the symbol table, for debugging purposes.

ScopeDisplay(std::ostream&)

```
 ScopeDisplay(std::ostream& os);
```

~ScopeDisplay()

```
 ~ScopeDisplay();
```

display(const Scope*)

```
void display(const SymbolLookup::Scope* s);
```

visit(TemplateParameterScope*)

```
void visit(SymbolLookup::TemplateParameterScope*);
```

visit(LocalScope*)

```
void visit(SymbolLookup::LocalScope*);
```

visit(PrototypeScope*)

```
void visit(SymbolLookup::PrototypeScope*);
```

visit(FunctionScope*)

```
void visit(SymbolLookup::FunctionScope*);
```

visit(Class*)

```
void visit(SymbolLookup::Class*);
```

visit(Namespace*)

```
void visit(SymbolLookup::Namespace*);
```

dump(const Scope*)

```
void dump(const SymbolLookup::Scope*);
```

indent()

```
 std::ostream & indent();
```

my_os

```
 std::ostream & my_os;
```

my_indent

```
size_t my_indent;
```

class InternalError

InternalError(const std::string&)

```
 InternalError(const std::string& what);
```

~InternalError()

```
 ~InternalError();
```

what()const

```
const char * what();
```

my_what

```
std::string my_what;
```

class Scope

A Scope contains symbol definitions.

symbol_iterator

```
typedef SymbolTable::const_iterator symbol_iterator;
```

scope_iterator

```
typedef ScopeTable::const_iterator scope_iterator;
```

LookupContext

```
typedef unsigned int  LookupContext;
```

Scope()

```
 Scope();
```

ref()

```
 SymbolLookup::Scope * ref();
```

ref()const

```
const SymbolLookup::Scope * ref();
```

unref()const

```
void unref();
```

outer_scope()const

```
const SymbolLookup::Scope * outer_scope();
```

global_scope()const

```
const SymbolLookup::Scope * global_scope();
```

accept(ScopeVisitor*)

```
void accept(SymbolLookup::ScopeVisitor* v);
```

symbols_begin()const

```
SymbolLookup::Scope::symbol_iterator symbols_begin();
```

symbols_end()const

```
SymbolLookup::Scope::symbol_iterator symbols_end();
```

scopes_begin()const

```
SymbolLookup::Scope::scope_iterator scopes_begin();
```

scopes_end()const

```
SymbolLookup::Scope::scope_iterator scopes_end();
```

declare(const PTree::Encoding&,const Symbol*)

```
void declare(const PTree::Encoding& name, const SymbolLookup::Symbol* \
symbol);
```

declare the given symbol in the local scope using the given encoded name.

declare_scope(const PTree::Node*,Scope*)

```
void declare_scope(const PTree::Node* node, SymbolLookup::Scope* scope);
```

use(const PTree::UsingDirective*)

```
void use(const PTree::UsingDirective*);
```

declare a 'using' directive. The default implementation raises an exception, as it is only well-formed when the current scope is a function scope or a namespace.

find_scope(const PTree::Node*)const

```
 SymbolLookup::Scope * find_scope(const PTree::Node*);
```

find a nested scope by declaration

find_scope(const PTree::Encoding&,const Symbol*)const

```
 SymbolLookup::Scope * find_scope(const PTree::Encoding&, const \
SymbolLookup::Symbol*);
```

find a nested scope by symbol. The encoded name is provided for diagnostic purposes only.

remove_scope(const PTree::Node*)

```
void remove_scope(const PTree::Node*);
```

Remove the given nested scope from the scope.

find(const PTree::Encoding&,LookupContext)const

```
SymbolLookup::SymbolSet find(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext);
```

find the encoded name declared in this scope and return a set of matching symbols.

remove(const Symbol*)

```
void remove(const SymbolLookup::Symbol* s);
```

Remove the given symbol from the scope. s shall not be used after its removal.

lookup(const PTree::Encoding&,LookupContext)const

```
SymbolLookup::SymbolSet lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext = DEFAULT);
```

look up the encoded name and return a set of matching symbols.

unqualified_lookup(const PTree::Encoding&,LookupContext)const

```
SymbolLookup::SymbolSet unqualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext = DEFAULT);
```

qualified_lookup(const PTree::Encoding&,LookupContext)const

```
SymbolLookup::SymbolSet qualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext = DEFAULT);
```

DEFAULT

```
const SymbolLookup::Scope::LookupContext  DEFAULT;
```

SCOPE

```
const SymbolLookup::Scope::LookupContext  SCOPE;
```

USING

```
const SymbolLookup::Scope::LookupContext  USING;
```

ELABORATE

```
const SymbolLookup::Scope::LookupContext  ELABORATE;
```

DECLARATION

```
const SymbolLookup::Scope::LookupContext  DECLARATION;
```

SymbolTable

```
typedef std::multimap<PTree::Encoding, const SymbolLookup::Symbol *> \
SymbolTable;
```

ScopeTable

```
typedef std::map<const PTree::Node *,  SymbolLookup::Scope *> ScopeTable;
```

~Scope()

```
 ~Scope();
```

Scopes are ref counted, and thus are deleted only by 'unref()'

my_symbols

```
SymbolLookup::Scope::SymbolTable my_symbols;
```

my_scopes

```
SymbolLookup::Scope::ScopeTable my_scopes;
```

my_refcount

```
size_t my_refcount;
```

class ScopeVisitor

A Visitor for Scopes. The default implementation does nothing, so users only need to implement the ones they need.

~ScopeVisitor()

```
 ~ScopeVisitor();
```

visit(TemplateParameterScope*)

```
void visit(SymbolLookup::TemplateParameterScope*);
```

visit(LocalScope*)

```
void visit(SymbolLookup::LocalScope*);
```

visit(PrototypeScope*)

```
void visit(SymbolLookup::PrototypeScope*);
```

visit(FunctionScope*)

```
void visit(SymbolLookup::FunctionScope*);
```

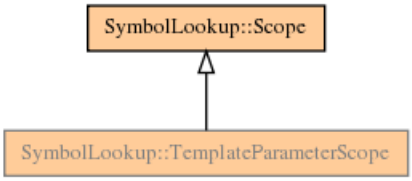
visit(Class*)

```
void visit(SymbolLookup::Class*);
```

visit(Namespace*)

```
void visit(SymbolLookup::Namespace*);
```

class TemplateParameterScope



TemplateParameterScope(const PTree::List*,const Scope*)

```
 TemplateParameterScope(const PTree::List* node, const \
SymbolLookup::Scope* outer);
```

unqualified_lookup(const PTree::Encoding&,Scope::LookupContext)const

```
SymbolLookup::SymbolSet unqualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext);
```

outer_scope()const

```
const SymbolLookup::Scope * outer_scope();
```

accept(ScopeVisitor*)

```
void accept(SymbolLookup::ScopeVisitor* v);
```

~TemplateParameterScope()

```
 ~TemplateParameterScope();
```

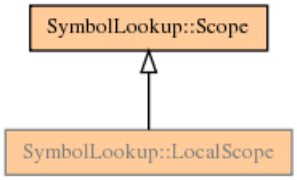
my_node

```
const PTree::List * my_node;
```

my_outer

```
const SymbolLookup::Scope * my_outer;
```

class LocalScope



LocalScope(const PTree::List*,const Scope*)

```
 LocalScope(const PTree::List* node, const SymbolLookup::Scope* outer);
```

outer_scope()const

```
const SymbolLookup::Scope * outer_scope();
```

unqualified_lookup(const PTree::Encoding&,Scope::LookupContext)const

```
SymbolLookup::SymbolSet unqualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext);
```

accept(ScopeVisitor*)

```
void accept(SymbolLookup::ScopeVisitor* v);
```

~LocalScope()

```
 ~LocalScope();
```

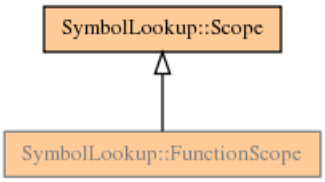
my_node

```
const PTree::List * my_node;
```

my_outer

```
const SymbolLookup::Scope * my_outer;
```

class FunctionScope



FunctionScope(const PTree::Declaration*,PrototypeScope*,const Scope*)

```
 FunctionScope(const PTree::Declaration*, SymbolLookup::PrototypeScope*, \
const SymbolLookup::Scope*);
```

use(const PTree::UsingDirective*)

```
void use(const PTree::UsingDirective*);
```

outer_scope()const

```
const SymbolLookup::Scope * outer_scope();
```

unqualified_lookup(const PTree::Encoding&,Scope::LookupContext)const

```
SymbolLookup::SymbolSet unqualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext);
```

qualified_lookup(const PTree::Encoding&,Scope::LookupContext)const

```
SymbolLookup::SymbolSet qualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext);
```

name()const

```
std::string name();
```

accept(ScopeVisitor*)

```
void accept(SymbolLookup::ScopeVisitor* v);
```

~FunctionScope()

```
 ~FunctionScope();
```

Using

```
typedef std::set<const SymbolLookup::Namespace *> Using;
```

my_decl

```
const PTree::Declaration * my_decl;
```

my_outer

```
const SymbolLookup::Scope * my_outer;
```

my_class

```
const SymbolLookup::Class * my_class;
```

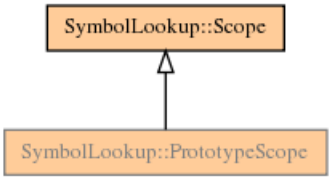
my_parameters

```
const SymbolLookup::TemplateParameterScope * my_parameters;
```

my_using

```
SymbolLookup::FunctionScope::Using my_using;
```

class PrototypeScope



PrototypeScope(const PTree::Node*,const Scope*,const TemplateParameterScope*)

```
 PrototypeScope(const PTree::Node* decl, const SymbolLookup::Scope* \
outer, const SymbolLookup::TemplateParameterScope* params);
```

outer_scope()const

```
const SymbolLookup::Scope * outer_scope();
```

unqualified_lookup(const PTree::Encoding&,Scope::LookupContext)const

```
SymbolLookup::SymbolSet unqualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext);
```

declaration()const

```
const PTree::Node * declaration();
```

parameters()const

```
const SymbolLookup::TemplateParameterScope * parameters();
```

name()const

```
std::string name();
```

accept(ScopeVisitor*)

```
void accept(SymbolLookup::ScopeVisitor* v);
```

~PrototypeScope()

```
 ~PrototypeScope();
```

FunctionScope

---

my_decl

```
const PTree::Node * my_decl;
```

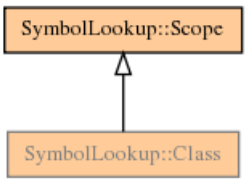
my_outer

```
const SymbolLookup::Scope * my_outer;
```

my_parameters

```
const SymbolLookup::TemplateParameterScope * my_parameters;
```

class Class



Bases

```
typedef std::vector<const SymbolLookup::Class *> Bases;
```

Class(const PTree::ClassSpec*,const Scope*,const Bases&,const TemplateParameterScope*)

```
 Class(const PTree::ClassSpec* spec, const SymbolLookup::Scope* outer, \
const SymbolLookup::Class::Bases& bases, const \
SymbolLookup::TemplateParameterScope* params);
```

outer_scope()const

```
const SymbolLookup::Scope * outer_scope();
```

unqualified_lookup(const PTree::Encoding&,Scope::LookupContext)const

```
SymbolLookup::SymbolSet unqualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext);
```

name()const

```
std::string name();
```

accept(ScopeVisitor*)

```
void accept(SymbolLookup::ScopeVisitor* v);
```

~Class()

```
 ~Class();
```

my_spec

```
const PTree::ClassSpec * my_spec;
```

my_outer

```
const SymbolLookup::Scope * my_outer;
```

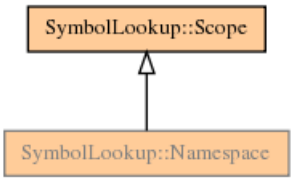
my_bases

```
SymbolLookup::Class::Bases my_bases;
```

my_parameters

```
const SymbolLookup::TemplateParameterScope * my_parameters;
```

class Namespace



Namespace(const PTree::NamespaceSpec*,const Namespace*)

```
 Namespace(const PTree::NamespaceSpec* spec, const \
SymbolLookup::Namespace* outer);
```

find_namespace(const PTree::NamespaceSpec*)const

```
 SymbolLookup::Namespace * find_namespace(const PTree::NamespaceSpec* \
name);
```

Find a nested namespace.

use(const PTree::UsingDirective*)

```
void use(const PTree::UsingDirective*);
```

outer_scope()const

```
const SymbolLookup::Scope * outer_scope();
```

unqualified_lookup(const PTree::Encoding&,Scope::LookupContext)const

```
SymbolLookup::SymbolSet unqualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext);
```

qualified_lookup(const PTree::Encoding&,Scope::LookupContext)const

```
SymbolLookup::SymbolSet qualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext);
```

name()const

```
std::string name();
```

accept(ScopeVisitor*)

```
void accept(SymbolLookup::ScopeVisitor* v);
```

~Namespace()

```
 ~Namespace();
```

Using

```
typedef std::set<const SymbolLookup::Namespace *> Using;
```

unqualified_lookup(const PTree::Encoding&,Scope::LookupContext,Using&)const

```
SymbolLookup::SymbolSet unqualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext, SymbolLookup::Namespace::Using&);
```

qualified_lookup(const PTree::Encoding&,Scope::LookupContext,Using&)const

```
SymbolLookup::SymbolSet qualified_lookup(const PTree::Encoding&, \
SymbolLookup::Scope::LookupContext, SymbolLookup::Namespace::Using&);
```

my_spec

```
const PTree::NamespaceSpec * my_spec;
```

my_outer

```
const SymbolLookup::Namespace * my_outer;
```

my_using

```
SymbolLookup::Namespace::Using my_using;
```

class SymbolVisitor

~SymbolVisitor()

```
 ~SymbolVisitor();
```

visit(const Symbol*)

```
void visit(const SymbolLookup::Symbol*);
```

visit(const VariableName*)

```
void visit(const SymbolLookup::VariableName*);
```

visit(const ConstName*)

```
void visit(const SymbolLookup::ConstName*);
```

visit(const TypeName*)

```
void visit(const SymbolLookup::TypeName*);
```

visit(const TypedefName*)

```
void visit(const SymbolLookup::TypedefName*);
```

visit(const ClassName*)

```
void visit(const SymbolLookup::ClassName*);
```

visit(const EnumName*)

```
void visit(const SymbolLookup::EnumName*);
```

visit(const ClassTemplateName*)

```
void visit(const SymbolLookup::ClassTemplateName*);
```

visit(const FunctionName*)

```
void visit(const SymbolLookup::FunctionName*);
```

visit(const FunctionTemplateName*)

```
void visit(const SymbolLookup::FunctionTemplateName*);
```

visit(const NamespaceName*)

```
void visit(const SymbolLookup::NamespaceName*);
```

class Symbol

Symbol(const PTree::Encoding&,const PTree::Node*,bool,Scope*)

```
 Symbol(const PTree::Encoding& t, const PTree::Node* p, bool def, \
SymbolLookup::Scope* s);
```

~Symbol()

```
 ~Symbol();
```

accept(SymbolVisitor*)const

```
void accept(SymbolLookup::SymbolVisitor* v);
```

type()const

```
const PTree::Encoding & type();
```

ptree()const

```
const PTree::Node * ptree();
```

is_definition()const

```
bool is_definition();
```

scope()const

```
 SymbolLookup::Scope * scope();
```

my_type

```
PTree::Encoding my_type;
```

my_ptree

```
const PTree::Node * my_ptree;
```

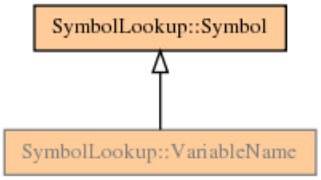
my_definition

```
bool my_definition;
```

my_scope

```
 SymbolLookup::Scope * my_scope;
```

class VariableName



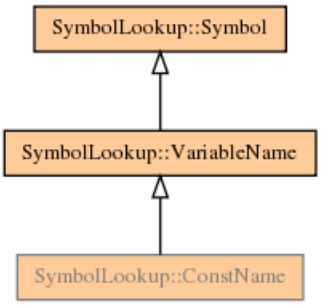
VariableName(const PTree::Encoding&,const PTree::Node*,bool,Scope*)

```
 VariableName(const PTree::Encoding& type, const PTree::Node* ptree, \
bool def, SymbolLookup::Scope* s);
```

accept(SymbolVisitor*)const

```
void accept(SymbolLookup::SymbolVisitor* v);
```

class ConstName



ConstName(const PTree::Encoding&,long,const PTree::Node*,bool,Scope*)

```
 ConstName(const PTree::Encoding& type, long v, const PTree::Node* \
ptree, bool def, SymbolLookup::Scope* s);
```

ConstName(const PTree::Encoding&,const PTree::Node*,bool,Scope*)

```
 ConstName(const PTree::Encoding& type, const PTree::Node* ptree, bool \
def, SymbolLookup::Scope* s);
```

accept(SymbolVisitor*)const

```
void accept(SymbolLookup::SymbolVisitor* v);
```

defined()const

```
bool defined();
```

value()const

```
long value();
```

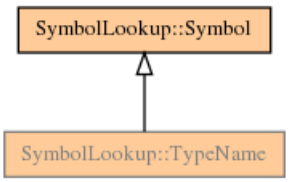
my_defined

```
bool my_defined;
```

my_value

```
long my_value;
```

class TypeName



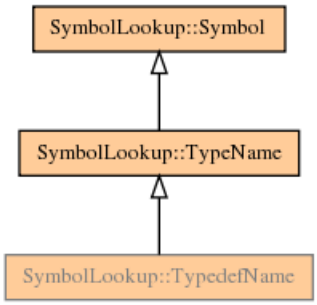
TypeName(const PTree::Encoding&,const PTree::Node*,bool,Scope*)

```
 TypeName(const PTree::Encoding& type, const PTree::Node* ptree, bool \
def, SymbolLookup::Scope* s);
```

accept(SymbolVisitor*)const

```
void accept(SymbolLookup::SymbolVisitor* v);
```

class TypedefName



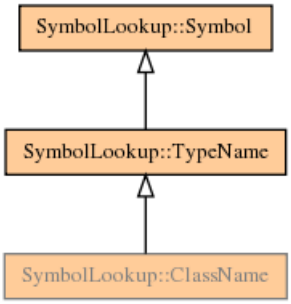
TypedefName(const PTree::Encoding&,const PTree::Node*,Scope*)

```
 TypedefName(const PTree::Encoding& type, const PTree::Node* ptree, \
SymbolLookup::Scope* scope);
```

accept(SymbolVisitor*)const

```
void accept(SymbolLookup::SymbolVisitor* v);
```

class ClassName



ClassName(const PTree::Encoding&,const PTree::Node*,bool,Scope*)

```
 ClassName(const PTree::Encoding& type, const PTree::Node* ptree, bool \
def, SymbolLookup::Scope* s);
```

accept(SymbolVisitor*)const

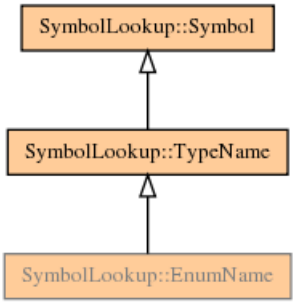
```
void accept(SymbolLookup::SymbolVisitor* v);
```

as_scope()const

```
 SymbolLookup::Class * as_scope();
```

Return the class scope associated with this symbol. This will return 0 if the class definition hasn't been seen yet.

class EnumName



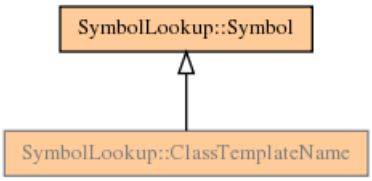
EnumName(const PTree::Encoding&,const PTree::Node*,Scope*)

```
 EnumName(const PTree::Encoding& type, const PTree::Node* ptree, \
SymbolLookup::Scope* scope);
```

accept(SymbolVisitor*)const

```
void accept(SymbolLookup::SymbolVisitor* v);
```

class ClassTemplateName



ClassTemplateName(const PTree::Encoding&,const PTree::Node*,bool,Scope*)

```
 ClassTemplateName(const PTree::Encoding& type, const PTree::Node* \
ptree, bool def, SymbolLookup::Scope* s);
```

accept(SymbolVisitor*)const

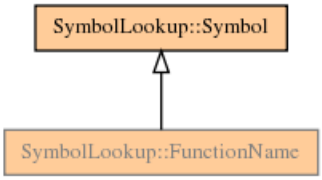
```
void accept(SymbolLookup::SymbolVisitor* v);
```

as_scope()const

```
 SymbolLookup::Class * as_scope();
```

Return the class scope associated with this symbol. This will return 0 if the class definition hasn't been seen yet.

class FunctionName



FunctionName(const PTree::Encoding&,const PTree::Node*,bool,Scope*)

```
 FunctionName(const PTree::Encoding& type, const PTree::Node* ptree, \
bool def, SymbolLookup::Scope* s);
```

accept(SymbolVisitor*)const

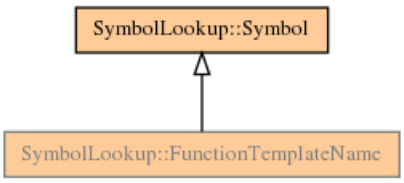
```
void accept(SymbolLookup::SymbolVisitor* v);
```

as_scope()const

```
 SymbolLookup::FunctionScope * as_scope();
```

Return the function scope associated with this symbol. This will return 0 if the function definition hasn't been seen yet.

class FunctionTemplateName



FunctionTemplateName(const PTree::Encoding&,const PTree::Node*,Scope*)

```
 FunctionTemplateName(const PTree::Encoding& type, const PTree::Node* \
ptree, SymbolLookup::Scope* s);
```

accept(SymbolVisitor*)const

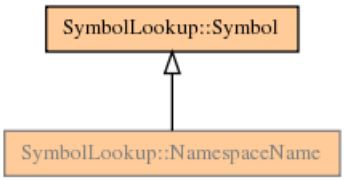
```
void accept(SymbolLookup::SymbolVisitor* v);
```

as_scope()const

```
 SymbolLookup::FunctionScope * as_scope();
```

Return the function scope associated with this symbol. This will return 0 if the function definition hasn't been seen yet.

class NamespaceName



NamespaceName(const PTree::Encoding&,const PTree::Node*,bool,Scope*)

```
 NamespaceName(const PTree::Encoding& type, const PTree::Node* ptree, \
bool def, SymbolLookup::Scope* s);
```

accept(SymbolVisitor*)const

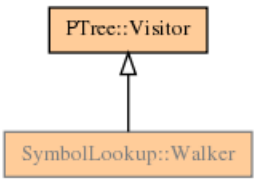
```
void accept(SymbolLookup::SymbolVisitor* v);
```

as_scope()const

```
 SymbolLookup::Namespace * as_scope();
```

Return the namespace scope associated with this symbol. This will return 0 if the namespace definition hasn't been seen yet.

class Walker



This Walker adjusts the symbol lookup table while the parse tree is being traversed such that symbols in the parse tree can be looked up correctly in the right context.

Walker(Scope*)

```
Walker(SymbolLookup::Scope*);
```

~Walker()

```
~Walker();
```

visit(PTree::List*)

```
void visit(PTree::List*);
```

visit(PTree::Block*)

```
void visit(PTree::Block*);
```

visit(PTree::TemplateDecl*)

```
void visit(PTree::TemplateDecl*);
```

visit(PTree::NamespaceSpec*)

```
void visit(PTree::NamespaceSpec*);
```

visit(PTree::FunctionDefinition*)

```
void visit(PTree::FunctionDefinition*);
```

visit(PTree::ClassSpec*)

```
void visit(PTree::ClassSpec*);
```

visit(PTree::DotMemberExpr*)

```
void visit(PTree::DotMemberExpr*);
```

visit(PTree::ArrowMemberExpr*)

```
void visit(PTree::ArrowMemberExpr*);
```

traverse_body(PTree::NamespaceSpec*)

```
void traverse_body(PTree::NamespaceSpec*);
```

Traverse the body of a namespace definition.

traverse_body(PTree::ClassSpec*)

```
void traverse_body(PTree::ClassSpec*);
```

Traverse the body of the class definition.

traverse_parameters(PTree::TemplateDecl*)

```
void traverse_parameters(PTree::TemplateDecl*);
```

Traverse the template parameter list of a template declaration.

traverse_body(PTree::FunctionDefinition*)

```
void traverse_body(PTree::FunctionDefinition*);
```

Traverse the body of the function definition.

current_scope()

```
const SymbolLookup::Scope * current_scope();
```

leave_scope()

```
void leave_scope();
```

Scopes

```
typedef std::stack< SymbolLookup::Scope *> Scopes;
```

visit_block(PTree::Block*)

```
void visit_block(PTree::Block*);
```

the virtual visit(Block) version above does scoping, which isn't what we want if traversing a function (FIXME: or is it ?) so the following factors out the common code.

my_scopes

```
SymbolLookup::Walker::Scopes my_scopes;
```

The symbol lookup table.

SymbolSet

```
typedef std::set<const SymbolLookup::Symbol *> SymbolSet;
```

struct TypeError

TypeError(const PTree::Encoding&,const PTree::Encoding&)

```
 TypeError(const PTree::Encoding& n, const PTree::Encoding& t);
```

~TypeError()

```
 ~TypeError();
```

what()const

```
const char * what();
```

name

```
PTree::Encoding name;
```

type

```
PTree::Encoding type;
```

struct Undefined

Undefined(const PTree::Encoding&,const PTree::Node*)

```
 Undefined(const PTree::Encoding& n, const PTree::Node* ref = 0);
```

~Undefined()

```
 ~Undefined();
```

what()const

```
const char * what();
```

name

```
PTree::Encoding name;
```

ptree

```
const PTree::Node * ptree;
```

struct MultiplyDefined

MultiplyDefined(const PTree::Encoding&,const PTree::Node*,const PTree::Node*)

```
 MultiplyDefined(const PTree::Encoding& n, const PTree::Node* decl, \
const PTree::Node* orig);
```

~MultiplyDefined()

```
 ~MultiplyDefined();
```

what()const

```
const char * what();
```

name

```
PTree::Encoding name;
```

declaration

```
const PTree::Node * declaration;
```

original

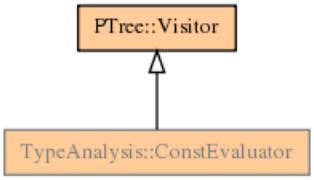
```
const PTree::Node * original;
```

display(const Scope*,std::ostream&)

```
void display(const SymbolLookup::Scope* s, std::ostream& os);
```

Namespace TypeAnalysis

class ConstEvaluator



Evaluate the value of a constant expression.

ConstEvaluator(const SymbolLookup::Scope*)

```
 ConstEvaluator(const SymbolLookup::Scope* s);
```

evaluate(const PTree::Node*,long&)

```
bool evaluate(const PTree::Node* node, long& value);
```

visit(PTree::Literal*)

```
void visit(PTree::Literal*);
```

visit(PTree::Identifier*)

```
void visit(PTree::Identifier*);
```

visit(PTree::FstyleCastExpr*)

```
void visit(PTree::FstyleCastExpr*);
```

visit(PTree::InfixExpr*)

```
void visit(PTree::InfixExpr*);
```

visit(PTree::SizeofExpr*)

```
void visit(PTree::SizeofExpr*);
```

visit(PTree::UnaryExpr*)

```
void visit(PTree::UnaryExpr*);
```

visit(PTree::CondExpr*)

```
void visit(PTree::CondExpr*);
```

visit(PTree::ParenExpr*)

```
void visit(PTree::ParenExpr*);
```

my_valid

```
bool my_valid;
```

my_value

```
long my_value;
```

my_scope

```
const SymbolLookup::Scope * my_scope;
```

class Kit

creates and remembers declared types.

Kit()

```
 Kit();
```

builtin(const std::string&)

```
const TypeAnalysis::Type * builtin(const std::string& name);
```

enum_(const std::string&)

```
const TypeAnalysis::Type * enum_(const std::string& name);
```

class_(const std::string&)

```
const TypeAnalysis::Type * class_(const std::string& name);
```

union_(const std::string&)

```
const TypeAnalysis::Type * union_(const std::string& name);
```

pointer(const Type*)

```
const TypeAnalysis::Type * pointer(const TypeAnalysis::Type* type);
```

reference(const Type*)

```
const TypeAnalysis::Type * reference(const TypeAnalysis::Type* type);
```

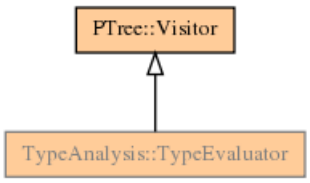
array(const Type*)

```
const TypeAnalysis::Type * array(const TypeAnalysis::Type* type);
```

pointer_to_member(const Type*,const Type*)

```
const TypeAnalysis::Type * pointer_to_member(const TypeAnalysis::Type* \
container, const TypeAnalysis::Type* member);
```

class TypeEvaluator



evaluate the type of an expression

TypeEvaluator(const SymbolLookup::Scope*)

```
 TypeEvaluator(const SymbolLookup::Scope* s);
```

evaluate(const PTree::Node*)

```
const TypeAnalysis::Type * evaluate(const PTree::Node* node);
```

visit(PTree::Literal*)

```
void visit(PTree::Literal*);
```

visit(PTree::Identifier*)

```
void visit(PTree::Identifier*);
```

visit(PTree::Kwd::This*)

```
void visit(PTree::Kwd::This*);
```

visit(PTree::Name*)

```
void visit(PTree::Name*);
```

visit(PTree::FstyleCastExpr*)

```
void visit(PTree::FstyleCastExpr*);
```

visit(PTree::AssignExpr*)

```
void visit(PTree::AssignExpr*);
```

visit(PTree::CondExpr*)

```
void visit(PTree::CondExpr*);
```

visit(PTree::InfixExpr*)

```
void visit(PTree::InfixExpr*);
```

visit(PTree::PmExpr*)

```
void visit(PTree::PmExpr*);
```

visit(PTree::CastExpr*)

```
void visit(PTree::CastExpr*);
```

visit(PTree::UnaryExpr*)

```
void visit(PTree::UnaryExpr*);
```

visit(PTree::ThrowExpr*)

```
void visit(PTree::ThrowExpr*);
```

visit(PTree::SizeofExpr*)

```
void visit(PTree::SizeofExpr*);
```

visit(PTree::TypeidExpr*)

```
void visit(PTree::TypeidExpr*);
```

visit(PTree::TypeofExpr*)

```
void visit(PTree::TypeofExpr*);
```

visit(PTree::NewExpr*)

```
void visit(PTree::NewExpr*);
```

visit(PTree::DeleteExpr*)

```
void visit(PTree::DeleteExpr*);
```

visit(PTree::ArrayExpr*)

```
void visit(PTree::ArrayExpr*);
```

visit(PTree::FuncallExpr*)

```
void visit(PTree::FuncallExpr*);
```

visit(PTree::PostfixExpr*)

```
void visit(PTree::PostfixExpr*);
```

visit(PTree::DotMemberExpr*)

```
void visit(PTree::DotMemberExpr*);
```

visit(PTree::ArrowMemberExpr*)

```
void visit(PTree::ArrowMemberExpr*);
```

visit(PTree::ParenExpr*)

```
void visit(PTree::ParenExpr*);
```

my_scope

```
const SymbolLookup::Scope * my_scope;
```

my_type

```
const TypeAnalysis::Type * my_type;
```

class Type

Type(const std::string&)

```
 Type(const std::string& name);
```

~Type()

```
 ~Type();
```

name()const

```
const std::string & name();
```

accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

ref()const

```
void ref();
```

deref()const

```
void deref();
```

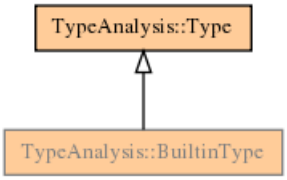
my_name

```
const std::string  my_name;
```

my_refcounter

```
size_t my_refcounter;
```

class BuiltinType



BuiltinType(const std::string&)

```
 BuiltinType(const std::string& name);
```

accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

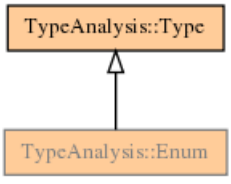
ref()const

```
void ref();
```

deref()const

```
void deref();
```

class Enum



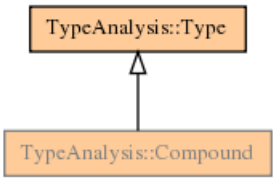
Enum(const std::string&)

```
 Enum(const std::string& name);
```

accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

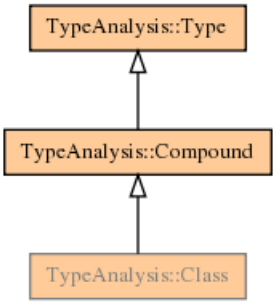
class Compound



Compound(const std::string&)

```
Compound(const std::string& name);
```

class Class



Kind

```
enum Kind { STRUCT, CLASS};
```

Class(Kind,const std::string&)

```
Class(TypeAnalysis::Class::Kind kind, const std::string& name);
```

accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

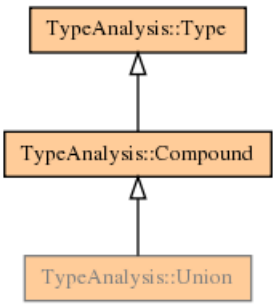
my_kind

```
TypeAnalysis::Class::Kind my_kind;
```

STRUCT

CLASS

class Union



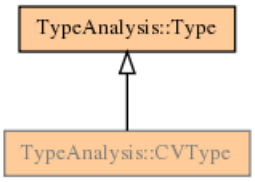
Union(const std::string&)

```
 Union(const std::string& name);
```

accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

class CVType



CVQualifier

```
enum CVQualifier { NONE=0x0, CONST=0x1, VOLATILE=0x2};
```

CVType(const Type*,CVQualifier)

```
 CVType(const TypeAnalysis::Type* type, \
TypeAnalysis::CVType::CVQualifier q);
```

accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

names

```
[4] const std::string  names;
```

my_type

```
const TypeAnalysis::Type * my_type;
```

my_qual

```
TypeAnalysis::CVType::CVQualifier my_qual;
```

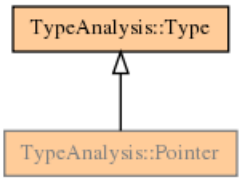
NONE

CONST

VOLATILE

---

class Pointer



Pointer(const Type*)

```
 Pointer(const TypeAnalysis::Type* type);
```

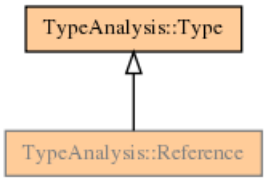
accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

my_type

```
const TypeAnalysis::Type * my_type;
```

class Reference



Reference(const Type*)

```
 Reference(const TypeAnalysis::Type* type);
```

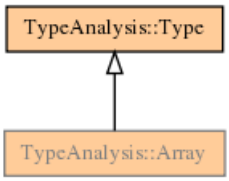
accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

my_type

```
const TypeAnalysis::Type * my_type;
```

class Array



Array(const Type*)

```
 Array(const TypeAnalysis::Type* type);
```

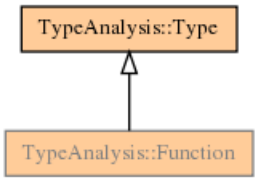
accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

my_type

```
const TypeAnalysis::Type * my_type;
```

class Function



Function()

```
 Function();
```

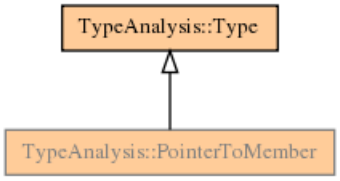
accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

my_type

```
const TypeAnalysis::Type * my_type;
```

class PointerToMember



PointerToMember()

```
 PointerToMember();
```

accept(Visitor*)

```
void accept(TypeAnalysis::Visitor* visitor);
```

my_container

```
const TypeAnalysis::Type * my_container;
```

my_member

```
const TypeAnalysis::Type * my_member;
```

class Visitor

~Visitor()

```
 ~Visitor();
```

visit(Type*)

```
void visit(TypeAnalysis::Type*);
```

visit(BuiltinType*)

```
void visit(TypeAnalysis::BuiltinType*);
```

visit(Enum*)

```
void visit(TypeAnalysis::Enum*);
```

visit(Class*)

```
void visit(TypeAnalysis::Class*);
```

visit(Union*)

```
void visit(TypeAnalysis::Union*);
```

visit(CVType*)

```
void visit(TypeAnalysis::CVType*);
```

visit(Pointer*)

```
void visit(TypeAnalysis::Pointer*);
```

visit(Reference*)

```
void visit(TypeAnalysis::Reference*);
```

visit(Array*)

```
void visit(TypeAnalysis::Array*);
```

visit(Function*)

```
void visit(TypeAnalysis::Function*);
```

visit(PointerToMember*)

```
void visit(TypeAnalysis::PointerToMember*);
```

evaluate_const(const SymbolLookup::Scope*,const PTree::Node*,long&)

```
bool evaluate_const(const SymbolLookup::Scope* scope, const PTree::Node* \
node, long& value);
```

Evaluate the value of a constant expression. TODO: This may also return the type of the expression...

resolve_funcall(const PTree::FuncallExpr*,const SymbolLookup::Scope*)

```
const SymbolLookup::Symbol * resolve_funcall(const PTree::FuncallExpr* \
funcall, const SymbolLookup::Scope*);
```

Resolve a function call in the context of the given scope.

type_of(const PTree::Node*,const SymbolLookup::Scope*)

```
const TypeAnalysis::Type * type_of(const PTree::Node* node, const \
SymbolLookup::Scope* s);
```

BOOL

```
TypeAnalysis::BuiltinType BOOL;
```

CHAR

```
TypeAnalysis::BuiltinType CHAR;
```

WCHAR

```
TypeAnalysis::BuiltinType WCHAR;
```

SHORT

```
TypeAnalysis::BuiltinType SHORT;
```

INT

```
TypeAnalysis::BuiltinType INT;
```

LONG

```
TypeAnalysis::BuiltinType LONG;
```

FLOAT

```
TypeAnalysis::BuiltinType FLOAT;
```

DOUBLE

```
TypeAnalysis::BuiltinType DOUBLE;
```

UCHAR

```
TypeAnalysis::BuiltinType UCHAR;
```

USHORT

```
TypeAnalysis::BuiltinType USHORT;
```

UINT

```
TypeAnalysis::BuiltinType UINT;
```

ULONG

```
TypeAnalysis::BuiltinType ULONG;
```

SCHAR

```
TypeAnalysis::BuiltinType SCHAR;
```

SSHORT

```
TypeAnalysis::BuiltinType SSHORT;
```

SINT

```
TypeAnalysis::BuiltinType SINT;
```

SLONG

```
TypeAnalysis::BuiltinType SLONG;
```

class Buffer

Buffer holds the memory on top of which a parse tree / syntax tree is constructed. Besides giving access to individual characters, it provides the means to register replacements for buffer chunks, such that when the Buffer's write method is executed the new file will contain the modified source.

Buffer(std::streambuf*,const std::string&)

```
 Buffer(std::streambuf*, const std::string& = std :: string ( "unknown" \
));
```

size()const

```
unsigned long  size();
```

return the size of the buffer

get()

```
char get();
```

report the character at the current position and advance one character

unget()

```
void unget();
```

undo the last get

reset(unsigned long)

```
void reset(unsigned long c = 0);
```

reset the current position to position c

position()const

```
unsigned long  position();
```

report the current position

at(unsigned long)const

```
char at(unsigned long p);
```

report the character at position p

ptr(unsigned long)const

```
const char * ptr(unsigned long p = 0);
```

report the pointer at position p

replace(const char*,const char*,const char*,unsigned long)

```
void replace(const char* from, const char* to, const char* begin, \
unsigned long length);
```

replace the text between from and to by the text between begin and begin + length

origin(const char*,std::string&)const

```
unsigned long  origin(const char*, std::string&);
```

Return the origin of the given pointer (filename and line number)

write(std::ostream&,const std::string&)const

```
void write(std::ostream&, const std::string&);
```

Write the buffer into the given output stream The first line contains a line directive issuing the input file name; if filename is non-empty, use this to fake another one.

Replacements

```
typedef std::vector<Buffer::Replacement> Replacements;
```

struct Replacement

Replacement(const char*,const char*,const char*,unsigned long)

```
 Replacement(const char* from, const char* to, const char* begin, \
unsigned long length);
```

smaller(const Replacement&,const Replacement&)

```
bool smaller(const Buffer::Replacement& r1, const Buffer::Replacement& \
r2);
```

from

```
const char * from;
```

to

```
const char * to;
```

begin

```
const char * begin;
```

length

```
unsigned long  length;
```

read_line_directive(unsigned long,long,unsigned long&,unsigned long&)const

```
long read_line_directive(unsigned long cursor, long line, unsigned \
long& begin, unsigned long& end);
```

read a line directive starting at position pos, and return the line number found. Also report the begin and end of the filename (with respect to the internal buffer). line is the default line number that gets reported on error (in which case begin and end remain unchanged)

my_filename

```
std::string my_filename;
```

my_buffer

```
std::string my_buffer;
```

my_cursor

```
unsigned long  my_cursor;
```

my_replacements

```
Buffer::Replacements my_replacements;
```

class Lexer

a Lexer reads tokens from a stream.

Comments

```
typedef std::vector<Token> Comments;
```

struct InvalidChar

InvalidChar(const std::string&)

```
 InvalidChar(const std::string& msg);
```

TokenSet

```
enum TokenSet { C=0x0, CXX=0x01, GCC=0x02, MSVC=0x04};
```

Define sets of token that are to be recognized as special keywords (as opposed to identifiers). They can be or'ed. If CXX is not specified, the Lexer will operate in 'C mode'.

Lexer(Buffer*,int)

```
 Lexer(Buffer*, int tokenset = CXX | GCC);
```

Construct a Lexer on the given Buffer using the given token set. The default token set is CXX with GCC extensions.

get_token(Token&)

```
Token::Type get_token(Token&);
```

look_ahead(size_t)

```
Token::Type look_ahead(size_t);
```

look_ahead(size_t,Token&)

```
Token::Type look_ahead(size_t, Token&);
```

save()

```
const char * save();
```

restore(const char*)

```
void restore(const char*);
```

get_comments()

```
Lexer::Comments get_comments();
```

origin(const char*,std::string&)const

```
unsigned long  origin(const char*, std::string&);
```

Return the origin of the given pointer (filename and line number)

class Queue

a Queue is used to read in tokens from a stream without consuming them

Container

```
typedef std::deque<Token> Container;
```

size_type

```
typedef Container::size_type size_type;
```

empty()const

```
bool empty();
```

size()const

```
Lexer::Queue::size_type size();
```

front()const

```
const Token & front();
```

back()const

```
const Token & back();
```

at(size_type)const

```
const Token & at(Lexer::Queue::size_type i);
```

push(const Token&)

```
void push(const Token& t);
```

pop()

```
void pop();
```

clear()

```
void clear();
```

my_container

```
Lexer::Queue::Container my_container;
```

Dictionary

```
typedef std::map<std::string, Token::Type> Dictionary;
```

rewind(const char*)

```
void rewind(const char*);
```

read_token(const char*&,size_t&)

```
Token::Type read_token(const char*&, size_t&);
```

fill(size_t)

```
bool fill(size_t o);
```

try to fill the token cache to contain at least o tokens. Returns false if there are not enough tokens.

skip_paren()

```
void skip_paren();
```

skip till end of paren

skip_line()

```
void skip_line();
```

skip till end of line

skip_attribute()

```
void skip_attribute();
```

skip __attribute__(...), ___asm__(...), ...

skip_extension(const char*&,size_t&)

```
Token::Type skip_extension(const char*&, size_t&);
```

skip __extension__(...).

skip_asm()

```
void skip_asm();
```

skip __asm ...

skip_declspec()

```
void skip_declspec();
```

skip __declspec(...).

skip_pragma()

```
void skip_pragma();
```

skip __pragma(...);.

get_next_non_white_char()

```
char get_next_non_white_char();
```

read_line()

```
Token::Type read_line();
```

read_char_const(unsigned long)

```
bool read_char_const(unsigned long top);
```

read_str_const(unsigned long)

```
bool read_str_const(unsigned long top);
```

read_number(char,unsigned long)

```
Token::Type read_number(char c, unsigned long top);
```

read_float(unsigned long)

```
Token::Type read_float(unsigned long top);
```

read_identifier(unsigned long)

```
Token::Type read_identifier(unsigned long top);
```

screen(const char*,size_t)

```
Token::Type screen(const char* identifier, size_t len);
```

read_separator(char,unsigned long)

```
Token::Type read_separator(char c, unsigned long top);
```

single_char_op(unsigned char)

```
Token::Type single_char_op(unsigned char c);
```

read_comment(char,unsigned long)

```
Token::Type read_comment(char c, unsigned long top);
```

my_buffer

```
 Buffer * my_buffer;
```

my_tokens

```
Lexer::Queue my_tokens;
```

my_keywords

```
Lexer::Dictionary my_keywords;
```

my_token

```
Token my_token;
```

my_comments

```
Lexer::Comments my_comments;
```

C

CXX

GCC

MSVC

class Parser

C++ Parser

This parser is a LL(k) parser with ad hoc rules such as backtracking.

<name>() is the grammer rule for a non-terminal <name>. opt_<name>() is the grammer rule for an optional non-terminal <name>. is_<name>() looks ahead and returns true if the next symbol is <name>.

class Error

Error is used to cache parse errors encountered during the execution of the parse method.

~Error()

```
~Error();
```

write(std::ostream&)const

```
void write(std::ostream&);
```

ErrorList

```
typedef std::vector< Parser::Error *> ErrorList;
```

RuleSet

```
enum RuleSet { CXX=0x01, GCC=0x02, MSVC=0x04};
```

RuleSet defines non-standard optional rules that can be chosen at runtime.

Parser(Lexer&,SymbolFactory&,int)

```
Parser(Lexer& lexer, SymbolFactory& symbols, int ruleset = CXX | GCC);
```

~Parser()

```
~Parser();
```

errors()const

```
const Parser::ErrorList & errors();
```

origin(const char*,std::string&)const

```
unsigned long  origin(const char*, std::string&);
```

Return the origin of the given pointer (filename and line number)

parse()

```
 PTree::Node * parse();
```

class StatusGuard

A StatusGuard manages a tentative parse. All actions invoked after its instantiation will be rolled back in the destructor unless 'commit' has been called before.

StatusGuard(Parser&)

```
 StatusGuard(Parser&);
```

~StatusGuard()

```
 ~StatusGuard();
```

commit()

```
void commit();
```

my_lexer

```
 Lexer & my_lexer;
```

my_token_mark

```
const char * my_token_mark;
```

my_errors

```
Parser::ErrorList my_errors;
```

my_error_mark

```
Parser::ErrorList::size_type my_error_mark;
```

my_committed

```
bool my_committed;
```

ScopeGuard

DeclKind

```
enum DeclKind { kDeclarator, kArgDeclarator, kCastDeclarator};
```

TemplateDeclKind

```
enum TemplateDeclKind { tdk_unknown, tdk_decl, tdk_instantiation, \
tdk_specialization, num_tdks};
```

declare(T*)

```
bool declare(Parser::T*);
```

mark_error()

```
bool mark_error();
```

show_message_head(const char*)

```
void show_message_head(const char*);
```

definition(PTree::Node*&)

```
bool definition(PTree::Node*&);
```

null_declaration(PTree::Node*&)

```
bool null_declaration(PTree::Node*&);
```

typedef_(PTree::Typedef*&)

```
bool typedef_(PTree::Typedef*&);
```

type_specifier(PTree::Node*&,bool,PTree::Encoding&)

```
bool type_specifier(PTree::Node*&, bool, PTree::Encoding&);
```

is_type_specifier()

```
bool is_type_specifier();
```

metaclass_decl(PTree::Node*&)

```
bool metaclass_decl(PTree::Node*&);
```

meta_arguments(PTree::Node*&)

```
bool meta_arguments(PTree::Node*&);
```

linkage_spec(PTree::Node*&)

```
bool linkage_spec(PTree::Node*&);
```

namespace_spec(PTree::NamespaceSpec*&)

```
bool namespace_spec(PTree::NamespaceSpec*&);
```

namespace_alias(PTree::NamespaceAlias*&)

```
bool namespace_alias(PTree::NamespaceAlias*&);
```

using_directive(PTree::UsingDirective*&)

```
bool using_directive(PTree::UsingDirective*&);
```

using_declaration(PTree::UsingDeclaration*&)

```
bool using_declaration(PTree::UsingDeclaration*&);
```

linkage_body(PTree::Node*&)

```
bool linkage_body(PTree::Node*&);
```

template_decl(PTree::Node*&)

```
bool template_decl(PTree::Node*&);
```

template_decl2(PTree::TemplateDecl*&,TemplateDeclKind&)

```
bool template_decl2(PTree::TemplateDecl*&, Parser::TemplateDeclKind& \
kind);
```

template_parameter_list(PTree::List*&)

```
bool template_parameter_list(PTree::List*&);
```

template-parameter-list:

- template-parameter
- template-parameter-list , template-parameter

template_parameter(PTree::Node*&)

```
bool template_parameter(PTree::Node*&);
```

template-parameter:

- type-parameter
- parameter-declaration

type_parameter(PTree::Node*&)

```
bool type_parameter(PTree::Node*&);
```

type-parameter:

- class identifier [opt]
- class identifier [opt] = type-id
- typename identifier [opt]
- typename identifier [opt] = type-id
- template < template-parameter-list > class identifier [opt]
- template < template-parameter-list > class identifier [opt] = id-expression

extern_template_decl(PTree::Node*&)

```
bool extern_template_decl(PTree::Node*&);
```

GNU extension: extern-template-decl:

- extern template declaration

declaration(PTree::Declaration*&)

```
bool declaration(PTree::Declaration*&);
```

integral_declaration(PTree::Declaration*&,PTree::Encoding&,PTree::Node*,PTree::Node*,PTree::Node*)

```
bool integral_declaration(PTree::Declaration*&, PTree::Encoding&, \
PTree::Node*, PTree::Node*, PTree::Node*);
```

const_declaration(PTree::Declaration*&,PTree::Encoding&,PTree::Node*,PTree::Node*)

```
bool const_declaration(PTree::Declaration*&, PTree::Encoding&, \
PTree::Node*, PTree::Node*);
```

other_declaration(PTree::Declaration*&,PTree::Encoding&,PTree::Node*,PTree::Node*,PTree::Node*)

```
bool other_declaration(PTree::Declaration*&, PTree::Encoding&, \
PTree::Node*, PTree::Node*, PTree::Node*);
```

condition(PTree::Node*&)

```
bool condition(PTree::Node*&);
```

condition:

- expression
- type-specifier-seq declarator = assignment-expression

is_constructor_decl()

```
bool is_constructor_decl();
```

is_ptr_to_member(int)

```
bool is_ptr_to_member(int);
```

opt_member_spec(PTree::Node*&)

```
bool opt_member_spec(PTree::Node*&);
```

opt_storage_spec(PTree::Node*&)

```
bool opt_storage_spec(PTree::Node*&);
```

storage-spec:

- empty
- static
- extern
- auto
- register
- mutable

opt_cv_qualifier(PTree::Node*&)

```
bool opt_cv_qualifier(PTree::Node*&);
```

cv-qualifier:

- empty
- const
- volatile

opt_integral_type_or_class_spec(PTree::Node*&,PTree::Encoding&)

```
bool opt_integral_type_or_class_spec(PTree::Node*&, PTree::Encoding&);
```

constructor_decl(PTree::Node*&,PTree::Encoding&)

```
bool constructor_decl(PTree::Node*&, PTree::Encoding&);
```

opt_throw_decl(PTree::Node*&)

```
bool opt_throw_decl(PTree::Node*&);
```

init_declarator_list(PTree::Node*&,PTree::Encoding&,bool,bool)

```
bool init_declarator_list(PTree::Node*&, PTree::Encoding&, bool, bool \
= false);
```

[gram.dcl.decl]

init_declarator(PTree::Node*&,PTree::Encoding&,bool,bool)

```
bool init_declarator(PTree::Node*&, PTree::Encoding&, bool, bool);
```

declarator(PTree::Node*&,DeclKind,bool,PTree::Encoding&,PTree::Encoding&,bool,bool)

```
bool declarator(PTree::Node*&, Parser::DeclKind, bool, PTree::Encoding&, \
PTree::Encoding&, bool, bool = false);
```

declarator2(PTree::Node*&,DeclKind,bool,PTree::Encoding&,PTree::Encoding&,bool,bool,PTree::Node**)

```
bool declarator2(PTree::Node*&, Parser::DeclKind, bool, \
PTree::Encoding&, PTree::Encoding&, bool, bool, PTree::Node**);
```

opt_ptr_operator(PTree::Node*&,PTree::Encoding&)

```
bool opt_ptr_operator(PTree::Node*&, PTree::Encoding&);
```

member_initializers(PTree::Node*&)

```
bool member_initializers(PTree::Node*&);
```

member_init(PTree::Node*&)

```
bool member_init(PTree::Node*&);
```

name(PTree::Node*&,PTree::Encoding&)

```
bool name(PTree::Node*&, PTree::Encoding&);
```

operator_name(PTree::Node*&,PTree::Encoding&)

```
bool operator_name(PTree::Node*&, PTree::Encoding&);
```

cast_operator_name(PTree::Node*&,PTree::Encoding&)

```
bool cast_operator_name(PTree::Node*&, PTree::Encoding&);
```

ptr_to_member(PTree::Node*&,PTree::Encoding&)

```
bool ptr_to_member(PTree::Node*&, PTree::Encoding&);
```

template_args(PTree::Node*&,PTree::Encoding&)

```
bool template_args(PTree::Node*&, PTree::Encoding&);
```

parameter_declaration_list_or_init(PTree::Node*&,bool&,PTree::Encoding&,bool)

```
bool parameter_declaration_list_or_init(PTree::Node*&, bool&, \
PTree::Encoding&, bool);
```

parameter_declaration_list(PTree::Node*&,PTree::Encoding&)

```
bool parameter_declaration_list(PTree::Node*&, PTree::Encoding&);
```

parameter_declaration(PTree::ParameterDeclaration*&,PTree::Encoding&)

```
bool parameter_declaration(PTree::ParameterDeclaration*&, \
PTree::Encoding&);
```

parameter-declaration:

- decl-specifier-seq declarator
- decl-specifier-seq declarator = assignment-expression
- decl-specifier-seq abstract-declarator [opt]
- decl-specifier-seq abstract-declarator [opt] = assignment-expression

function_arguments(PTree::Node*&)

```
bool function_arguments(PTree::Node*&);
```

designation(PTree::Node*&)

```
bool designation(PTree::Node*&);
```

initialize_expr(PTree::Node*&)

```
bool initialize_expr(PTree::Node*&);
```

enum_spec(PTree::EnumSpec*&,PTree::Encoding&)

```
bool enum_spec(PTree::EnumSpec*&, PTree::Encoding&);
```

enum_body(PTree::Node*&)

```
bool enum_body(PTree::Node*&);
```

class_spec(PTree::ClassSpec*&,PTree::Encoding&)

```
bool class_spec(PTree::ClassSpec*&, PTree::Encoding&);
```

base_clause(PTree::Node*&)

```
bool base_clause(PTree::Node*&);
```

base-clause:

- *:* base-specifier-list

base-specifier-list:

- base-specifier
- base-specifier-list , base-specifier

base-specifier:

- virtual access-specifier [opt] *::* [opt] nested-name-specifier [opt] class-name
- access-specifier virtual [opt] *::* [opt] nested-name-specifier [opt] class-name

class_body(PTree::ClassBody*&)

```
bool class_body(PTree::ClassBody*&);
```

class_member(PTree::Node*&)

```
bool class_member(PTree::Node*&);
```

access_decl(PTree::Node*&)

```
bool access_decl(PTree::Node*&);
```

user_access_spec(PTree::Node*&)

```
bool user_access_spec(PTree::Node*&);
```

expression(PTree::Node*&)

```
bool expression(PTree::Node*&);
```

expression:

- assignment-expression
- expression , assignment-expression

assign_expr(PTree::Node*&)

```
bool assign_expr(PTree::Node*&);
```

assignment-expression:

- conditional-expression
- logical-or-expression assignment-operator assignment-expression
- throw-expression

conditional_expr(PTree::Node*&)

```
bool conditional_expr(PTree::Node*&);
```

conditional-expression:

- logical-or-expression
- logical-or-expression *?* expression *:* assignment-expression

logical_or_expr(PTree::Node*&)

```
bool logical_or_expr(PTree::Node*&);
```

logical-or-expression:

- logical-and-expression
- logical-or-expression || logical-and-expression

logical_and_expr(PTree::Node*&)

```
bool logical_and_expr(PTree::Node*&);
```

logical-and-expression:

- inclusive-or-expression
- logical-and-expr *&&* inclusive-or-expression

inclusive_or_expr(PTree::Node*&)

```
bool inclusive_or_expr(PTree::Node*&);
```

inclusive-or-expression:

- exclusive-or-expression
- inclusive-or-expression | exclusive-or-expression

exclusive_or_expr(PTree::Node*&)

```
bool exclusive_or_expr(PTree::Node*&);
```

exclusive-or-expression:

- and-expression
- exclusive-or-expression ^ and-expression

and_expr(PTree::Node*&)

```
bool and_expr(PTree::Node*&);
```

and-expression:

- equality-expression
- and-expression & equality-expression

equality_expr(PTree::Node*&)

```
bool equality_expr(PTree::Node*&);
```

equality-expression:

- relational-expression
- equality-expression == relational-expression
- equality-expression *!*= relational-expression

relational_expr(PTree::Node*&)

```
bool relational_expr(PTree::Node*&);
```

relational-expression:

- shift-expression
- relational-expression < shift-expression
- relational-expression > shift-expression
- relational-expression <= shift-expression
- relational-expression >= shift-expression

shift_expr(PTree::Node*&)

```
bool shift_expr(PTree::Node*&);
```

shift-expression:

- additive-expression

- shift-expression << additive-expression
- shift-expression >> additive-expression

additive_expr(PTree::Node*&)

```
bool additive_expr(PTree::Node*&);
```

additive-expression:

- multiplicative-expression
- additive-expression + multiplicative-expression
- additive-expression - multiplicative-expression

multiplicative_expr(PTree::Node*&)

```
bool multiplicative_expr(PTree::Node*&);
```

multiplicative-expression:

- pm-expression
- multiplicative-expression * pm-expression
- multiplicative-expression / pm-expression
- multiplicative-expression % pm-expression

pm_expr(PTree::Node*&)

```
bool pm_expr(PTree::Node*&);
```

pm-expression:

- cast-expression
- pm-expression .* cast-expression
- pm-expression ->* cast-expression

cast_expr(PTree::Node*&)

```
bool cast_expr(PTree::Node*&);
```

cast-expression:

- unary-expression
- *(* type-id *)* cast-expression

type_id(PTree::Node*&)

```
bool type_id(PTree::Node*&);
```

type-id:

- type-specifier-seq abstract-declarator [opt]

type_id(PTree::Node*&,PTree::Encoding&)

```
bool type_id(PTree::Node*&, PTree::Encoding&);
```

unary_expr(PTree::Node*&)

```
bool unary_expr(PTree::Node*&);
```

unary-expression:

- postfix-expression
- ++ cast-expression
- -- cast-expression
- unary-operator cast-expression
- *sizeof* unary-expression
- *sizeof (* unary-expression *)*
- new-expression
- delete-expression

unary-operator:

- *
- &
- +
- -
- *!*
- ~

throw_expr(PTree::Node*&)

```
bool throw_expr(PTree::Node*&);
```

throw-expression:

- *throw* assignment-expression

sizeof_expr(PTree::Node*&)

```
bool sizeof_expr(PTree::Node*&);
```

sizeof-expression:

- *sizeof* unary-expression
- *sizeof (* type-id *)*

offsetof_expr(PTree::Node*&)

```
bool offsetof_expr(PTree::Node*&);
```

typeid_expr(PTree::Node*&)

```
bool typeid_expr(PTree::Node*&);
```

typeid-expression:

- typeid *(* type-id *)*
- typeid *(* expression *)*

is_allocate_expr(Token::Type)

```
bool is_allocate_expr(Token::Type);
```

allocate_expr(PTree::Node*&)

```
bool allocate_expr(PTree::Node*&);
```

userdef_keyword(PTree::Node*&)

```
bool userdef_keyword(PTree::Node*&);
```

allocate_type(PTree::Node*&)

```
bool allocate_type(PTree::Node*&);
```

new_declarator(PTree::Declarator*&,PTree::Encoding&)

```
bool new_declarator(PTree::Declarator*&, PTree::Encoding&);
```

allocate_initializer(PTree::Node*&)

```
bool allocate_initializer(PTree::Node*&);
```

postfix_expr(PTree::Node*&)

```
bool postfix_expr(PTree::Node*&);
```

primary_expr(PTree::Node*&)

```
bool primary_expr(PTree::Node*&);
```

typeof_expr(PTree::Node*&)

```
bool typeof_expr(PTree::Node*&);
```

userdef_statement(PTree::Node*&)

```
bool userdef_statement(PTree::Node*&);
```

var_name(PTree::Node*&)

```
bool var_name(PTree::Node*&);
```

var_name_core(PTree::Node*&,PTree::Encoding&)

```
bool var_name_core(PTree::Node*&, PTree::Encoding&);
```

is_template_args()

```
bool is_template_args();
```

function_body(PTree::Block*&)

```
bool function_body(PTree::Block*&);
```

function-body:

- compound-statement

compound_statement(PTree::Block*&,bool)

```
bool compound_statement(PTree::Block*&, bool create_scope = false);
```

compound-statement:

- *{* statement [opt] *}*

statement(PTree::Node*&)

```
bool statement(PTree::Node*&);
```

if_statement(PTree::Node*&)

```
bool if_statement(PTree::Node*&);
```

if-statement:

- *if (* condition *)* statement
- *if (* condition *)* statement else statement

switch_statement(PTree::Node*&)

```
bool switch_statement(PTree::Node*&);
```

switch-statement:

- *switch (* condition *)* statement

while_statement(PTree::Node*&)

```
bool while_statement(PTree::Node*&);
```

while-statement:

- *while (* condition *)* statement

do_statement(PTree::Node*&)

```
bool do_statement(PTree::Node*&);
```

do-statement:

- *do* statement *while (* condition *) ;*

for_statement(PTree::Node*&)

```
bool for_statement(PTree::Node*&);
```

try_block(PTree::Node*&)

```
bool try_block(PTree::Node*&);
```

try-block:

- *try* compound-statement handler-seq

handler-seq:

- handler handler-seq [opt]

handler:

- *catch (* exception-declaration *)* compound-statement

exception-declaration:

- type-specifier-seq declarator
- type-specifier-seq abstract-declarator
- type-specifier-seq
- ...

expr_statement(PTree::Node*&)

```
bool expr_statement(PTree::Node*&);
```

declaration_statement(PTree::Declaration*&)

```
bool declaration_statement(PTree::Declaration*&);
```

integral_decl_statement(PTree::Declaration*&,PTree::Encoding&,PTree::Node*,PTree::Node*,PTree::Node*)

```
bool integral_decl_statement(PTree::Declaration*&, PTree::Encoding&, \
PTree::Node*, PTree::Node*, PTree::Node*);
```

other_decl_statement(PTree::Declaration*&,PTree::Encoding&,PTree::Node*,PTree::Node*)

```
bool other_decl_statement(PTree::Declaration*&, PTree::Encoding&, \
PTree::Node*, PTree::Node*);
```

maybe_typename_or_class_template(Token&)

```
bool maybe_typename_or_class_template(Token&);
```

skip_to(Token::Type)

```
void skip_to(Token::Type token);
```

more_var_name()

```
bool more_var_name();
```

my_lexer

```
 Lexer & my_lexer;
```

my_ruleset

```
int my_ruleset;
```

my_symbols

```
 SymbolFactory & my_symbols;
```

my_scope_is_valid

```
bool my_scope_is_valid;
```

Record whether the current scope is valid. This allows the parser to continue parsing even after it was unable to enter a scope (such as in a function definition with a qualified name that wasn't declared before).

my_errors

```
Parser::ErrorList my_errors;
```

my_comments

```
PTree::Node * my_comments;
```

my_gt_is_operator

```
bool my_gt_is_operator;
```

If true, > is interpreted as ther greater-than operator. If false, it marks the end of a template-id or template-parameter-list.

my_in_template_decl

```
bool my_in_template_decl;
```

CXX

GCC

MSVC

kDeclarator

kArgDeclarator

kCastDeclarator

tdk_unknown

tdk_decl

tdk_instantiation

tdk_specialization

num_tdks

class SymbolFactory

SymbolFactory populates a symbol table.

Language

```
enum Language { NONE=0x00, C99=0x01, CXX=0x02};
```

SymbolFactory(Language)

```
SymbolFactory(SymbolFactory::Language = CXX);
```

Create a symbol lookup table for the given language. Right now only CXX is supported.

current_scope()

```
SymbolLookup::Scope * current_scope();
```

enter_scope(const PTree::NamespaceSpec*)

```
void enter_scope(const PTree::NamespaceSpec*);
```

enter_scope(const PTree::ClassSpec*)

```
void enter_scope(const PTree::ClassSpec*);
```

enter_scope(const PTree::Node*)

```
void enter_scope(const PTree::Node*);
```

enter_scope(const PTree::FunctionDefinition*)

```
void enter_scope(const PTree::FunctionDefinition*);
```

enter_scope(const PTree::TemplateDecl*)

```
void enter_scope(const PTree::TemplateDecl*);
```

enter_scope(const PTree::Block*)

```
void enter_scope(const PTree::Block*);
```

leave_scope()

```
void leave_scope();
```

declare(const PTree::Declaration*)

```
void declare(const PTree::Declaration*);
```

declare(const PTree::Typedef*)

```
void declare(const PTree::Typedef*);
```

declare(const PTree::EnumSpec*)

```
void declare(const PTree::EnumSpec*);
```

declare the enumeration as a new TYPE as well as all the enumerators as CONST

declare(const PTree::NamespaceSpec*)

```
void declare(const PTree::NamespaceSpec*);
```

declare the namespace as a new NAMESPACE

declare(const PTree::ClassSpec*)

```
void declare(const PTree::ClassSpec*);
```

declare the class as a new TYPE

declare(const PTree::TemplateDecl*)

```
void declare(const PTree::TemplateDecl*);
```

declare(const PTree::TypeParameter*)

```
void declare(const PTree::TypeParameter*);
```

declare(const PTree::UsingDirective*)

```
void declare(const PTree::UsingDirective*);
```

declare(const PTree::ParameterDeclaration*)

```
void declare(const PTree::ParameterDeclaration*);
```

declare(const PTree::UsingDeclaration*)

```
void declare(const PTree::UsingDeclaration*);
```

Scopes

```
typedef std::stack< SymbolLookup::Scope *> Scopes;
```

lookup_scope_of_qname(PTree::Encoding&,const PTree::Node*)

```
 SymbolLookup::Scope * lookup_scope_of_qname(PTree::Encoding&, const \
PTree::Node*);
```

Lookup the scope of a qualified name. The encoded name is modified in place to refer to the unqualified name.

my_language

```
SymbolFactory::Language my_language;
```

my_scopes

```
SymbolFactory::Scopes my_scopes;
```

my_prototype

```
 SymbolLookup::PrototypeScope * my_prototype;
```

When parsing a function definition the declarator is seen first, and thus a prototype is created to hold the parameters. Later, when the function definition proper is seen, the symbols are transfered and the prototype is deleted.

my_template_parameters

```
SymbolLookup::TemplateParameterScope * my_template_parameters;
```

When parsing a class or function template the template-parameter-list is seen first. Since ClassSpec and Declarator don't know they are part of a template declaration, we cache it here so it gets consumed when the Class or PrototypeScope are created.

NONE

C99

CXX

---

class Timer

Timer()

```
Timer();
```

elapsed()const

```
double elapsed();
```

my_start

```
std::clock_t my_start;
```

struct Token

A Token is what the Lexer splits an input stream into. It refers to a region in the underlaying buffer and it has a type.

- line directive: *^"#"{blank}*{digit}+({blank}+.*)?n*
- pragma directive: *^"#"{blank}*"pragma".*n*
- Constant:
  - *{digit}+{int_suffix}**
  - *"0"{xletter}{hexdigit}+{int_suffix}**
  - *{digit}*.{digit}+{float_suffix}**
  - *{digit}+.{float_suffix}**
  - *{digit}*.{digit}+"e"("+"|"-")*{digit}+{float_suffix}**
  - *{digit}+."e"("+"|"-")*{digit}+{float_suffix}**

- *{digit}+"e"("+"|"-")*{digit}+{float_suffix}**

- CharConst: *'([^'n]|\[^n])'*
- WideCharConst: *L'([^'n]|\[^n])'*
- StringL: *"([^"n]|\["n])*"*
- WideStringL: *L"([^"n]|\["n])*"*
- Identifier: *{letter}+({letter}|{digit})**
- AssignOp: *= /= %= += -= &= ^= <<= >>=
- EqualOp: == *!=*
- RelOp: <= >=
- ShiftOp: << >>
- LogOrOp: ||
- LogAndOp: *&&*
- IncOp: ++ --
- Scope: *::*
- Ellipsis: ...
- PmOp: .* ->*
- ArrowOp: ->
- others: *!%^&*()-+={}|~[];:<>?,./*
- BadToken: *others*

Type

```
typedef int Type;
```

`0000

```
enum `0000 { Identifier=258, Constant, CharConst, StringL, AssignOp, \
EqualOp, RelOp, ShiftOp, LogOrOp, LogAndOp, IncOp, Scope, Ellipsis, \
PmOp, ArrowOp, BadToken, AUTO, CHAR, CLASS, CONST, DELETE, DOUBLE, \
ENUM, EXTERN, FLOAT, FRIEND, INLINE, INT, LONG, NEW, OPERATOR, PRIVATE, \
PROTECTED, PUBLIC, REGISTER, SHORT, SIGNED, STATIC, STRUCT, TYPEDEF, \
TYPENAME, UNION, UNSIGNED, VIRTUAL, VOID, VOLATILE, TEMPLATE, MUTABLE, \
BREAK, CASE, CONTINUE, DEFAULT, DO, ELSE, FOR, GOTO, IF, OFFSETOF, \
RETURN, SIZEOF, SWITCH, THIS, WHILE, ATTRIBUTE, METACLASS, UserKeyword, \
UserKeyword2, UserKeyword3, UserKeyword4, BOOLEAN, EXTENSION, TRY, \
CATCH, THROW, UserKeyword5, NAMESPACE, USING, TYPEID, TYPEOF, \
WideStringL, WideCharConst, WCHAR, ntDeclarator=400, ntName, \
ntFstyleCast, ntClassSpec, ntEnumSpec, ntDeclaration, ntTypedef, \
```

```
ntTemplateDecl, ntMetaclassDecl, ntParameterDecl, ntLinkageSpec, \
ntAccessSpec, ntUserAccessSpec, ntUserdefKeyword, ntExternTemplate, \
ntAccessDecl, ntNamespaceSpec, ntUsing, ntTemplateInstantiation, \
ntNamespaceAlias, ntIfStatement, ntSwitchStatement, ntWhileStatement, \
ntDoStatement, ntForStatement, ntBreakStatement, ntContinueStatement, \
ntReturnStatement, ntGotoStatement, ntCaseStatement, ntDefaultStatement, \
ntLabelStatement, ntExprStatement, ntTryStatement, ntCommaExpr, \
ntAssignExpr, ntCondExpr, ntInfixExpr, ntPmExpr, ntCastExpr, \
ntUnaryExpr, ntSizeofExpr, ntNewExpr, ntDeleteExpr, ntArrayExpr, \
ntFuncallExpr, ntPostfixExpr, ntUserStatementExpr, ntDotMemberExpr, \
ntArrowMemberExpr, ntParenExpr, ntStaticUserStatementExpr, ntThrowExpr, \
ntTypeidExpr, ntTypeofExpr, Ignore=500, ASM, DECLSPEC, PRAGMA, INT64, \
Comment};
```

Token()

```
 Token();
```

Token(const char*,size_t,Type)

```
 Token(const char* s, size_t l, Token::Type t);
```

operator==(char)const

```
bool operator==(char c);
```

ptr

```
const char * ptr;
```

length

```
size_t length;
```

type

```
Token::Type type;
```

Identifier

Constant

< The first 256 are representing character literals.

CharConst

StringL

AssignOp

EqualOp

RelOp

ShiftOp

LogOrOp

LogAndOp

IncOp

Scope

Ellipsis

PmOp

ArrowOp

BadToken

AUTO

CHAR

CLASS

CONST

DELETE

DOUBLE

ENUM

EXTERN

FLOAT

FRIEND

INLINE

INT

LONG

NEW

OPERATOR

PRIVATE

PROTECTED

PUBLIC

REGISTER

SHORT

SIGNED

STATIC

STRUCT

TYPEDEF

TYPENAME

UNION

UNSIGNED

VIRTUAL

VOID

VOLATILE

TEMPLATE

MUTABLE

BREAK

CASE

CONTINUE

DEFAULT

DO

ELSE

FOR

GOTO

IF

OFFSETOF

RETURN

SIZEOF

SWITCH

THIS

WHILE

ATTRIBUTE

METACLASS

UserKeyword

UserKeyword2

UserKeyword3

UserKeyword4

BOOLEAN

EXTENSION

TRY

CATCH

THROW

UserKeyword5

NAMESPACE

USING

TYPEID

TYPEOF

WideStringL

WideCharConst

WCHAR

ntDeclarator

ntName

ntFstyleCast

ntClassSpec

ntEnumSpec

ntDeclaration

ntTypedef

ntTemplateDecl

ntMetaclassDecl

ntParameterDecl

ntLinkageSpec

ntAccessSpec

ntUserAccessSpec

ntUserdefKeyword

ntExternTemplate

ntAccessDecl

ntNamespaceSpec

ntUsing

ntTemplateInstantiation

ntNamespaceAlias

ntIfStatement

ntSwitchStatement

ntWhileStatement

ntDoStatement

ntForStatement

ntBreakStatement

ntContinueStatement

ntReturnStatement

ntGotoStatement

ntCaseStatement

ntDefaultStatement

ntLabelStatement

ntExprStatement

ntTryStatement

ntCommaExpr

ntAssignExpr

ntCondExpr

ntInfixExpr

ntPmExpr

ntCastExpr

ntUnaryExpr

ntSizeofExpr

ntNewExpr

ntDeleteExpr

ntArrayExpr

ntFuncallExpr

ntPostfixExpr

ntUserStatementExpr

ntDotMemberExpr

ntArrowMemberExpr

ntParenExpr

ntStaticUserStatementExpr

ntThrowExpr

ntTypeidExpr

ntTypeofExpr

Ignore

ASM

DECLSPEC

PRAGMA

INT64

Comment

class Trace

struct Entry

operator<<(const T&)const

```
const Trace::Entry & operator<<(const Trace::Entry::T& t);
```

Entry(bool)

```
 Entry(bool e);
```

Entry(const Entry&)

```
 Entry(const Trace::Entry& e);
```

~Entry()

```
 ~Entry();
```

enabled

```
bool enabled;
```

Category

```
enum Category { NONE=0x0, PTREE=0x01, SYMBOLLOOKUP=0x02, PARSING=0x04, \
TRANSLATION=0x08, ALL=0xff};
```

Trace(const std::string&,unsigned int,const T&)

```
 Trace(const std::string& s, unsigned int c, const Synopsis::Trace::T& \
t);
```

operator<<(const T&)const

```
Trace::Entry operator<<(const Trace::T& t);
```

Trace(const std::string&,unsigned int)

```
 Trace(const std::string& s, unsigned int c);
```

~Trace()

```
 ~Trace();
```

enable(unsigned int)

```
void enable(unsigned int mask = ALL);
```

indent()

```
std::string indent();
```

my_mask

```
unsigned int  my_mask;
```

my_level

```
size_t my_level;
```

my_scope

```
std::string my_scope;
```

my_visibility

```
bool my_visibility;
```

NONE

PTREE

SYMBOLLOOKUP

PARSING

TRANSLATION

ALL

is_blank(char)

```
bool is_blank(char c);
```

is_digit(char)

```
bool is_digit(char c);
```

is_eletter(char)

```
bool is_eletter(char c);
```

is_float_suffix(char)

```
bool is_float_suffix(char c);
```

is_hexdigit(char)

```
bool is_hexdigit(char c);
```

is_int_suffix(char)

```
bool is_int_suffix(char c);
```

is_letter(char)

```
bool is_letter(char c);
```

is_xletter(char)

```
bool is_xletter(char c);
```

Types

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Functions

Symbols

IDL.idlast.DeclRepoId.__init__,
IDL.idlast.Enum.__init__,
IDL.idlast.Enumerator.__init__,
IDL.idlast.Exception.__init__,
IDL.idlast.Factory.__init__,
IDL.idlast.Forward.__init__,
IDL.idlast.Interface.__init__,
IDL.idlast.Member.__init__,
IDL.idlast.Module.__init__,
IDL.idlast.Native.__init__,
IDL.idlast.Operation.__init__,
IDL.idlast.Parameter.__init__,
IDL.idlast.Pragma.__init__,
IDL.idlast.StateMember.__init__,
IDL.idlast.Struct.__init__,
IDL.idlast.StructForward.__init__,
IDL.idlast.Typedef.__init__,
IDL.idlast.Union.__init__,
IDL.idlast.UnionCase.__init__,
IDL.idlast.UnionForward.__init__,
IDL.idlast.Value.__init__,
IDL.idlast.ValueAbs.__init__,
IDL.idlast.ValueBox.__init__,
IDL.idlast.ValueForward.__init__,
IDL.idltype.Base.__init__,
IDL.idltype.Declared.__init__,
IDL.idltype.Error.__init__,
IDL.idltype.Fixed.__init__,
IDL.idltype.Sequence.__init__,
IDL.idltype.String.__init__,
IDL.idltype.Type.__init__,
IDL.idltype.WString.__init__,
IDL.omni.ASGTranslator.__init__,
IDL.omni.TypeTranslator.__init__,
Inheritance.__init__,
IR.IR.__init__,
Macro.__init__,
Markup.Javadoc.Javadoc.Block.__init__,
Markup.Javadoc.Javadoc.__init__,
Markup.Markup.Struct.__init__,
Markup.RST.DocBookTranslator.__init__,
Markup.RST.SummaryExtractor.__init__,
MetaModule.__init__,
ModifierTypeId.__init__,
Module.__init__,
NamedTypeId.__init__,
Operation.__init__,
OperationTemplate.__init__,
Parameter.__init__,
ParametrizedTypeId.__init__,
Processor.Composite.__init__,
Processor.Error.__init__,
Processor.Parameter.__init__,
Processor.Parametrized.__init__,

A

Builtin.accept,
BuiltinTypeId.accept,
Class.accept,
ClassTemplate.accept,
Const.accept,
Declaration.accept,
DeclaredTypeId.accept,
DependentTypeId.accept,
Enum.accept,
Enumerator.accept,
Forward.accept,
Function.accept,
FunctionTemplate.accept,
FunctionTypeId.accept,
Group.accept,
IDL.idlast.AST.accept,
IDL.idlast.Attribute.accept,
IDL.idlast.CaseLabel.accept,
IDL.idlast.Const.accept,
IDL.idlast.Decl.accept,
IDL.idlast.Declarator.accept,
IDL.idlast.Enum.accept,
IDL.idlast.Enumerator.accept,
IDL.idlast.Exception.accept,
IDL.idlast.Factory.accept,
IDL.idlast.Forward.accept,
IDL.idlast.Interface.accept,
IDL.idlast.Member.accept,
IDL.idlast.Module.accept,
IDL.idlast.Native.accept,
IDL.idlast.Operation.accept,
IDL.idlast.Parameter.accept,
IDL.idlast.StateMember.accept,
IDL.idlast.Struct.accept,
IDL.idlast.StructForward.accept,
IDL.idlast.Typedef.accept,
IDL.idlast.Union.accept,
IDL.idlast.UnionCase.accept,
IDL.idlast.UnionForward.accept,
IDL.idlast.Value.accept,
IDL.idlast.ValueAbs.accept,
IDL.idlast.ValueBox.accept,
IDL.idlast.ValueForward.accept,
IDL.idltype.Base.accept,
IDL.idltype.Declared.accept,
IDL.idltype.Fixed.accept,
IDL.idltype.Sequence.accept,
IDL.idltype.String.accept,
IDL.idltype.Type.accept,
IDL.idltype.WString.accept,
Inheritance.accept,
Macro.accept,
MetaModule.accept,
ModifierTypeId.accept,

B

C

D

depart_document
    Markup.RST.DocBookTranslator.depart_document,
depart_emphasis
    Markup.RST.DocBookTranslator.depart_emphasis,
depart_entry
    Markup.RST.DocBookTranslator.depart_entry,
depart_enumerated_list
    Markup.RST.DocBookTranslator.depart_enumerated_list,
depart_error
    Markup.RST.DocBookTranslator.depart_error,
depart_field
    Markup.RST.DocBookTranslator.depart_field,
depart_field_argument
    Markup.RST.DocBookTranslator.depart_field_argument,
depart_field_body
    Markup.RST.DocBookTranslator.depart_field_body,
depart_field_list
    Markup.RST.DocBookTranslator.depart_field_list,
depart_field_name
    Markup.RST.DocBookTranslator.depart_field_name,
depart_figure
    Markup.RST.DocBookTranslator.depart_figure,
depart_footer
    Markup.RST.DocBookTranslator.depart_footer,
depart_footnote
    Markup.RST.DocBookTranslator.depart_footnote,
depart_generated
    Markup.RST.DocBookTranslator.depart_generated,
depart_header
    Markup.RST.DocBookTranslator.depart_header,
depart_hint
    Markup.RST.DocBookTranslator.depart_hint,
depart_image
    Markup.RST.DocBookTranslator.depart_image,
depart_important
    Markup.RST.DocBookTranslator.depart_important,
depart_interpreted
    Markup.RST.DocBookTranslator.depart_interpreted,
depart_label
    Markup.RST.DocBookTranslator.depart_label,
depart_legend
    Markup.RST.DocBookTranslator.depart_legend,
depart_line_block
    Markup.RST.DocBookTranslator.depart_line_block,
depart_list_item
    Markup.RST.DocBookTranslator.depart_list_item,
depart_literal
    Markup.RST.DocBookTranslator.depart_literal,
depart_literal_block
    Markup.RST.DocBookTranslator.depart_literal_block,
depart_note
    Markup.RST.DocBookTranslator.depart_note,
depart_option
    Markup.RST.DocBookTranslator.depart_option,

depart_option_argument
    Markup.RST.DocBookTranslator.depart_option_argument,
depart_option_group
    Markup.RST.DocBookTranslator.depart_option_group,
depart_option_list
    Markup.RST.DocBookTranslator.depart_option_list,
depart_option_list_item
    Markup.RST.DocBookTranslator.depart_option_list_item,
depart_option_string
    Markup.RST.DocBookTranslator.depart_option_string,
depart_paragraph
    Markup.RST.DocBookTranslator.depart_paragraph,
depart_reference
    Markup.RST.DocBookTranslator.depart_reference,
depart_row
    Markup.RST.DocBookTranslator.depart_row,
depart_rubric
    Markup.RST.DocBookTranslator.depart_rubric,
depart_section
    Markup.RST.DocBookTranslator.depart_section,
depart_sidebar
    Markup.RST.DocBookTranslator.depart_sidebar,
depart_strong
    Markup.RST.DocBookTranslator.depart_strong,
depart_subscript
    Markup.RST.DocBookTranslator.depart_subscript,
depart_subtitle
    Markup.RST.DocBookTranslator.depart_subtitle,
depart_superscript
    Markup.RST.DocBookTranslator.depart_superscript,
depart_table
    Markup.RST.DocBookTranslator.depart_table,
depart_target
    Markup.RST.DocBookTranslator.depart_target,
depart_tbody
    Markup.RST.DocBookTranslator.depart_tbody,
depart_term
    Markup.RST.DocBookTranslator.depart_term,
depart_Text
    Markup.RST.DocBookTranslator.depart_Text,
depart_tgroup
    Markup.RST.DocBookTranslator.depart_tgroup,
depart_thead
    Markup.RST.DocBookTranslator.depart_thead,
depart_tip
    Markup.RST.DocBookTranslator.depart_tip,
depart_title
    Markup.RST.DocBookTranslator.depart_title,
depart_title_reference
    Markup.RST.DocBookTranslator.depart_title_reference,
depart_topic
    Markup.RST.DocBookTranslator.depart_topic,
depart_transition
    Markup.RST.DocBookTranslator.depart_transition,

E

F

format_inheritance
    Fragments.ClassHierarchySimple.ClassHierarchySimple.format_inheritance,
format_inlines
    Markup.Javadoc.Javadoc.format_inlines,
format_inline_tag
    Markup.Javadoc.Javadoc.format_inline_tag,
format_macro
    Fragment.Fragment.format_macro,
    Fragments.DeclarationFormatter.DeclarationFormatter.format_macro,
    Fragments.Default.Default.format_macro,
format_meta_module
    Fragment.Fragment.format_meta_module,
    Fragments.DeclarationFormatter.DeclarationFormatter.format_meta_module,
    Fragments.Default.Default.format_meta_module,
    Fragments.HeadingFormatter.HeadingFormatter.format_meta_module,
format_modifiers
    Fragment.Fragment.format_modifiers,
format_module
    Fragment.Fragment.format_module,
    Fragments.DeclarationFormatter.DeclarationFormatter.format_module,
    Fragments.Default.Default.format_module,
    Fragments.HeadingFormatter.HeadingFormatter.format_module,
format_module_of_name
    Fragments.HeadingFormatter.HeadingFormatter.format_module_of_name,
format_module_or_group
    DocBook.DetailFormatter.format_module_or_group,
format_name
    Fragments.HeadingFormatter.HeadingFormatter.format_name,
format_name_in_module
    Fragments.HeadingFormatter.HeadingFormatter.format_name_in_module,
format_operation
    Fragment.Fragment.format_operation,
    Fragments.DeclarationFormatter.DeclarationFormatter.format_operation,
    Fragments.Default.Default.format_operation,
    Fragments.InheritanceFormatter.InheritanceFormatter.format_operation,
format_operation_template
    Fragment.Fragment.format_operation_template,
    Fragments.DeclarationFormatter.DeclarationFormatter.format_operation_template,
    Fragments.Default.Default.format_operation_template,
format_parameter
    Fragments.DeclarationFormatter.DeclarationFormatter.format_parameter,
    Fragments.HeadingFormatter.HeadingFormatter.format_parameter,
format_parameters
    Fragments.DeclarationFormatter.DeclarationFormatter.format_parameters,
format_params
    Markup.Javadoc.Javadoc.format_params,
format_scope
    Fragment.Fragment.format_scope,
    Fragments.DeclarationFormatter.DeclarationFormatter.format_scope,
    Fragments.Default.Default.format_scope,
format_tag
    Markup.Javadoc.Javadoc.format_tag,
format_throws
    Markup.Javadoc.Javadoc.format_throws,

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

IDL.omni.ASGTranslator.visitForward,
visitFrom
Python.ASGTranslator.ASGTranslator.visitFrom,
visitFunction
Python.ASGTranslator.ASGTranslator.visitFunction,
visitGetattr
Python.ASGTranslator.ASGTranslator.visitGetattr,
visitImport
Python.ASGTranslator.ASGTranslator.visitImport,
visitInterface
IDL.idlvisitor.AstVisitor.visitInterface,
IDL.omni.ASGTranslator.visitInterface,
visitMember
IDL.idlvisitor.AstVisitor.visitMember,
IDL.omni.ASGTranslator.visitMember,
visitModule
IDL.idlvisitor.AstVisitor.visitModule,
IDL.omni.ASGTranslator.visitModule,
Python.ASGTranslator.ASGTranslator.visitModule,
visitName
Python.ASGTranslator.ASGTranslator.visitName,
visitNative
IDL.idlvisitor.AstVisitor.visitNative,
visitOperation
IDL.idlvisitor.AstVisitor.visitOperation,
IDL.omni.ASGTranslator.visitOperation,
visitParameter
IDL.idlvisitor.AstVisitor.visitParameter,
IDL.omni.ASGTranslator.visitParameter,
visitSequenceType
IDL.idlvisitor.TypeVisitor.visitSequenceType,
IDL.omni.TypeTranslator.visitSequenceType,
visitStateMember
IDL.idlvisitor.AstVisitor.visitStateMember,
visitStmt
Python.ASGTranslator.ASGTranslator.visitStmt,
visitStringType
IDL.idlvisitor.TypeVisitor.visitStringType,
IDL.omni.TypeTranslator.visitStringType,
visitStruct
IDL.idlvisitor.AstVisitor.visitStruct,
IDL.omni.ASGTranslator.visitStruct,
visitStructForward
IDL.idlvisitor.AstVisitor.visitStructForward,
visitTypedef
IDL.idlvisitor.AstVisitor.visitTypedef,
IDL.omni.ASGTranslator.visitTypedef,
visitUnion
IDL.idlvisitor.AstVisitor.visitUnion,
IDL.omni.ASGTranslator.visitUnion,
visitUnionCase
IDL.idlvisitor.AstVisitor.visitUnionCase,
IDL.omni.ASGTranslator.visitUnionCase,
visitUnionForward

visit_citation_reference
    Markup.RST.DocBookTranslator.visit_citation_reference,
visit_class
    DocBook.DetailFormatter.visit_class,
    DocBook._BaseClasses.visit_class,
    Linker.Linker.visit_class,
    Part.Part.visit_class,
    ScopeStripper.ScopeStripper.visit_class,
    Syntax.CxxDetailSyntax.visit_class,
    Syntax.CxxSummarySyntax.visit_class,
    Syntax.PythonDetailSyntax.visit_class,
    Syntax.PythonSummarySyntax.visit_class,
    Visitor.visit_class,
visit_classifier
    Markup.RST.DocBookTranslator.visit_classifier,
visit_class_template
    Part.Part.visit_class_template,
    ScopeStripper.ScopeStripper.visit_class_template,
    Syntax.CxxDetailSyntax.visit_class_template,
    Syntax.CxxSummarySyntax.visit_class_template,
    Visitor.visit_class_template,
visit_colspec
    Markup.RST.DocBookTranslator.visit_colspec,
visit_comment
    Markup.RST.DocBookTranslator.visit_comment,
visit_const
    Linker.Linker.visit_const,
    Part.Part.visit_const,
    Syntax.CxxSyntax.visit_const,
    Syntax.PythonSyntax.visit_const,
    Visitor.visit_const,
visit_contact
    Markup.RST.DocBookTranslator.visit_contact,
visit_copyright
    Markup.RST.DocBookTranslator.visit_copyright,
visit_danger
    Markup.RST.DocBookTranslator.visit_danger,
visit_date
    Markup.RST.DocBookTranslator.visit_date,
visit_declaration
    AccessRestrictor.AccessRestrictor.visit_declaration,
    Comments.Filter.Filter.visit_declaration,
    Comments.Grouper.Grouper.visit_declaration,
    Comments.Previous.Previous.visit_declaration,
    Comments.Translator.Translator.visit_declaration,
    DocBook.DetailFormatter.visit_declaration,
    DocBook.SummaryFormatter.visit_declaration,
    ModuleFilter.ModuleFilter.visit_declaration,
    NameMapper.NamePrefixer.visit_declaration,
    Part.Part.visit_declaration,
    ScopeStripper.ScopeStripper.visit_declaration,
    Visitor.visit_declaration,
visit_declared_type_id
    DocBook.DetailFormatter.visit_declared_type_id,

visit_image
    Markup.RST.DocBookTranslator.visit_image,
visit_important
    Markup.RST.DocBookTranslator.visit_important,
visit_inheritance
    DocBook.DetailFormatter.visit_inheritance,
    DocBook._BaseClasses.visit_inheritance,
    Linker.Linker.visit_inheritance,
    Syntax.CxxDetailSyntax.visit_inheritance,
    Syntax.CxxSummarySyntax.visit_inheritance,
    Syntax.PythonDetailSyntax.visit_inheritance,
    Syntax.PythonSummarySyntax.visit_inheritance,
    Visitor.visit_inheritance,
visit_interpreted
    Markup.RST.DocBookTranslator.visit_interpreted,
visit_label
    Markup.RST.DocBookTranslator.visit_label,
visit_legend
    Markup.RST.DocBookTranslator.visit_legend,
visit_line_block
    Markup.RST.DocBookTranslator.visit_line_block,
visit_list_item
    Markup.RST.DocBookTranslator.visit_list_item,
visit_literal
    Markup.RST.DocBookTranslator.visit_literal,
visit_literal_block
    Markup.RST.DocBookTranslator.visit_literal_block,
visit_macro
    MacroFilter.MacroFilter.visit_macro,
    Part.Part.visit_macro,
    Syntax.CxxDetailSyntax.visit_macro,
    Syntax.CxxSummarySyntax.visit_macro,
    Visitor.visit_macro,
visit_meta_module
    DocBook.SummaryFormatter.visit_meta_module,
    Linker.Linker.visit_meta_module,
    ModuleSorter.ModuleSorter.visit_meta_module,
    Part.Part.visit_meta_module,
    ScopeStripper.ScopeStripper.visit_meta_module,
    Visitor.visit_meta_module,
visit_modifier_type_id
    DocBook.DetailFormatter.visit_modifier_type_id,
    Linker.Linker.visit_modifier_type_id,
    Part.Part.visit_modifier_type_id,
    Syntax.CxxSyntax.visit_modifier_type_id,
    Views.InheritanceGraph.DeclarationFinder.visit_modifier_type_id,
    Visitor.visit_modifier_type_id,
visit_module
    DocBook.DetailFormatter.visit_module,
    DocBook.ModuleLister.visit_module,
    Linker.Linker.visit_module,
    ModuleFilter.ModuleFilter.visit_module,
    Part.Part.visit_module,
    Syntax.CxxDetailSyntax.visit_module,

W

X

Chapter 3. Python API Reference

Abstract explaining the Python API reference.

The Internal Representation

Modules

- DocString
- IR
- QualifiedName
- SXR
- SourceFile

Module DocString

class DocString

A doc-string for ASG nodes.

text

```
text
```

markup

```
markup
```

__init__

```
__init__(self, text, markup)
```

Module IR

class IR

Top-level Internal Representation. This is essentially a dictionary of different representations such as Parse Tree, Abstract Semantic Graph, etc.

files

```
files
```

A dictionary mapping filenames to `SourceFile.SourceFile` instances.

asg

```
asg
```

The Abstract Semantic Graph.

sxr

```
sxr
```

The Source Cross-Reference SymbolTable.

__init__

```
__init__(self, files = None, asg = None, sxr = None)
```

Constructor

copy

```
copy(self)
```

Make a shallow copy of this IR.

save

```
save(self, filename)
```

Saves an IR object to the given filename

merge

```
merge(self, other)
```

Merges another IR. Files and declarations are appended to those in this IR, and types are merged by overwriting existing types - Unduplicator is responsible for sorting out the mess this may cause :)

load

```
load(filename)
```

Loads an IR object from the given filename

Module QualifiedName

class QualifiedName

QualifiedName.QualifiedName

__getslice__

```
__getslice__(self, begin, end)
```

This method exists because python < 3.0 still uses __getslice__ for builtin types. (See http://bugs.python.org/issue2041)

__add__

```
__add__(self, other)
```

Overload self + other to preserve the type.

sep

```
sep
```

__getitem__

```
__getitem__(self, i)
```

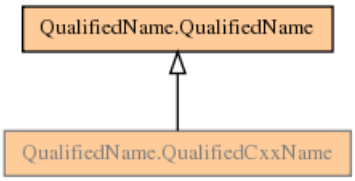
If i is a slice, make sure a QualifiedName is returned.

prune

```
prune(self, other)
```

Return a copy of other with any prefix it shares with self removed. e.g. ('A', 'B', 'C', 'D').prune(('A', 'B', 'D')) -> ('C', 'D')

class QualifiedCxxName



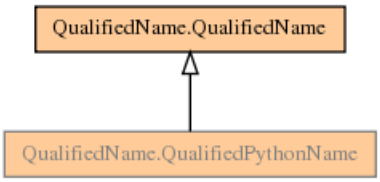
sep

```
sep
```

__str__

```
__str__(self)
```

class QualifiedPythonName



sep

```
sep
```

__str__

```
__str__(self)
```

Module SXR

class Entry

SXR.Entry

definitions

```
definitions
```

List of (file, line, scope) tuples.

calls

```
calls
```

List of (file, line, scope) tuples.

references

```
references
```

List of (file, line, scope) tuples.

__init__

```
__init__(self)
```

Represents a set of references found for a given symbol.

class SXR

SXR.SXR

Symboltable containing source code locations of symbol definitions, as well as different types of references.

_index

```
_index
```

__init__

```
__init__(self)
```

index

```
index(self)
```

generate_index

```
generate_index(self)
```

(Re-)generate an index after entries have been added.

merge

```
merge(self, other)
```

Module SourceFile

class Include

Information about an include directive in a SourceFile. If the include directive required a macro expansion to get the filename, the is_macro will return true. If the include directive was actually an include_next, then is_next will return true.

target

```
target
```

The target SourceFile object being referenced.

name

```
name
```

The name by which the target is referenced.

is_macro

```
is_macro
```

True if the directive uses a macro.

is_next

```
is_next
```

True if this is using #include_next (GNU extension).

__init__

```
__init__(self, target, name, is_macro, is_next)
```

class MacroCall

A class to support mapping from positions in a preprocessed file back to positions in the original file.

name

```
name
```

The name of the macro being called.

start

```
start
```

(line, column) pair indicating the start of the call.

end

```
end
```

(line, column) pair indicating the end of the call.

expanded_start

```
expanded_start
```

(line, column) pair indicating the start of the expansion in the preprocessed file.

expanded_end

```
expanded_end
```

(line, column) pair indicating the end of the expansion in the preprocessed file.

__init__

```
__init__(self, name, start, end, expanded_start, expanded_end)
```

class SourceFile

The information about a file that the ASG was generated from. Contains filename, all declarations from this file (even nested ones) and includes (aka imports) from this file.

name

```
name
```

The filename.

abs_name

```
abs_name
```

The absolute filename.

annotations

```
annotations
```

Dictionary with file annotations.

includes

```
includes
```

List of includes this file contains.

declarations

```
declarations
```

List of declarations this file contains.

macro_calls

```
macro_calls
```

List of macro calls this file contains.

__init__

```
__init__(self, name, abs_name, language, primary = False)
```

Constructor

The Abstract Semantic Graph

class ASG

declarations

```
declarations
```

types

```
types
```

__init__

```
__init__(self, declarations = None, types = None)
```

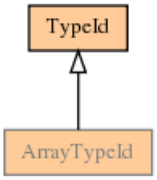
copy

```
copy(self)
```

merge

```
merge(self, other)
```

class ArrayTypeId



A modifier that adds array dimensions to a type-id.

__cmp__

```
__cmp__(self, other)
```

Comparison operator

alias

```
alias
```

sizes

```
sizes
```

__init__

```
__init__(self, language, alias, sizes)
```

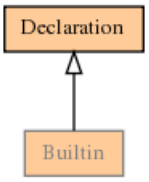
accept

```
accept(self, visitor)
```

__str__

```
__str__(self)
```

class Builtin

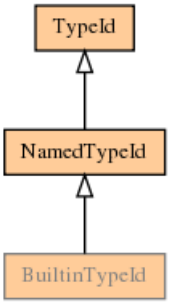


A node for internal use only.

accept

```
accept(self, visitor)
```

class BuiltinTypeId



Class for builtin type-ids

__cmp__

```
__cmp__(self, other)
```

Comparison operator

__init__

```
__init__(self, language, name)
```

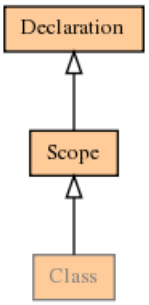
accept

```
accept(self, visitor)
```

__str__

```
__str__(self)
```

class Class



parents

```
parents
```

is_template_specialization

```
is_template_specialization
```

primary_template

```
primary_template
```

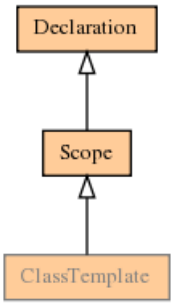
__init__

```
__init__(self, file, line, type, name, is_template_specialization = \
False)
```

accept

```
accept(self, visitor)
```

class ClassTemplate



parents

```
parents
```

template

```
template
```

is_template_specialization

```
is_template_specialization
```

primary_template

```
primary_template
```

specializations

```
specializations
```

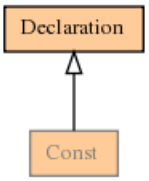
__init__

```
__init__(self, file, line, type, name, template = None, \
is_template_specialization = False)
```

accept

```
accept(self, visitor)
```

class Const



Constant declaration. A constant is a name with a type and value.

ctype

```
ctype
```

value

```
value
```

__init__

```
__init__(self, file, line, type, ctype, name, value)
```

accept

```
accept(self, visitor)
```

DEFAULT

```
DEFAULT
```

class Debugger

Wrap the object's 'accept' method, printing out the visitor's type. Useful for tracing visitors visiting declarations.

__init__

```
__init__(cls, name, bases, dict)
```

class Declaration

Declaration base class. Every declaration has a name, type, accessibility and annotations. The default accessibility is DEFAULT except for C++ where the Parser always sets it to one of the other three.

file

```
file
```

SourceFile instance this declaration is part of.

line

```
line
```

The line number of this declaration.

name

```
name
```

The (fully qualified) name of the declared object.

type

```
type
```

A string describing the (language-specific) type of the declared object.

accessibility

```
accessibility
```

Accessibility descriptor for the declared object.

annotations

```
annotations
```

A dictionary holding any annotations of this object.

__init__

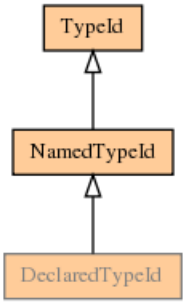
```
__init__(self, file, line, type, name)
```

accept

```
accept(self, visitor)
```

Visit the given visitor

class DeclaredTypeId



Class for declared types

__cmp__

```
__cmp__(self, other)
```

Comparison operator

declaration

```
declaration
```

__init__

```
__init__(self, language, name, declaration)
```

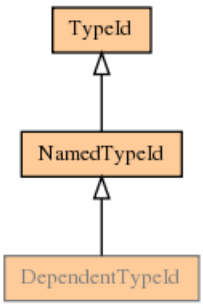
accept

```
accept(self, visitor)
```

__str__

```
__str__(self)
```

class DependentTypeId



Class for template dependent type-ids

__cmp__

```
__cmp__(self, other)
```

Comparison operator

__init__

```
__init__(self, language, name)
```

accept

```
accept(self, visitor)
```

__str__

```
__str__(self)
```

class Dictionary

Dictionary extends the builtin 'dict' by adding a lookup method to it.

lookup

```
lookup(self, name, scopes)
```

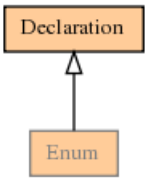
locate 'name' in one of the scopes

merge

```
merge(self, dict)
```

merge in a foreign dictionary, overriding already defined types only if they are of type 'Unknown'.

class Enum



Enum declaration. The actual names and values are encapsulated by Enumerator objects.

enumerators

```
enumerators
```

eos

```
eos
```

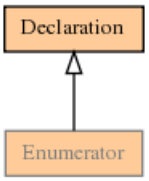
__init__

```
__init__(self, file, line, name, enumerators)
```

accept

```
accept(self, visitor)
```

class Enumerator



Enumerator of an Enum. Enumerators represent the individual names and values in an enum.

value

```
value
```

__init__

```
__init__(self, file, line, name, value)
```

accept

```
accept(self, visitor)
```

class Error

Exception class used by ASG internals.

err

```
err
```

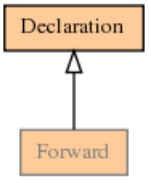
__init__

```
__init__(self, err)
```

__repr__

```
__repr__(self)
```

class Forward



Forward declaration

template

```
template
```

is_template_specialization

```
is_template_specialization
```

primary_template

```
primary_template
```

specializations

```
specializations
```

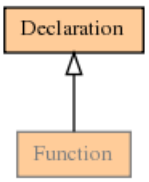
__init__

```
__init__(self, file, line, type, name, is_template_specialization = \
False)
```

accept

```
accept(self, visitor)
```

class Function



Function declaration. Note that function names are stored in mangled form to allow overriding. Formatters should use the real_name to extract the unmangled name.

__cmp__

```
__cmp__(self, other)
```

Recursively compares the typespec of the function

_real_name

```
_real_name
```

premodifier

```
premodifier
```

return_type

```
return_type
```

parameters

```
parameters
```

postmodifier

```
postmodifier
```

exceptions

```
exceptions
```

real_name

```
real_name
```

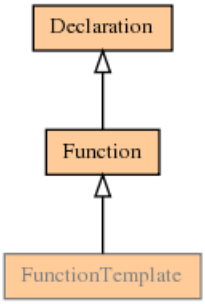
__init__

```
__init__(self, file, line, type, premod, return_type, postmod, name, \
real_name)
```

accept

```
accept(self, visitor)
```

class FunctionTemplate



template

```
template
```

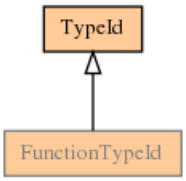
__init__

```
__init__(self, file, line, type, premod, return_type, postmod, name, \
real_name, template = None)
```

accept

```
accept(self, visitor)
```

class FunctionTypeId



Class for function (pointer) types.

return_type

```
return_type
```

premod

```
premod
```

parameters

```
parameters
```

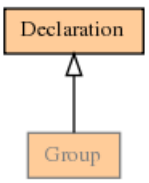
__init__

```
__init__(self, language, return_type, premod, parameters)
```

accept

```
accept(self, visitor)
```

class Group



Base class for groups which contain declarations. This class doesn't correspond to any language construct. Rather, it may be used with comment-embedded grouping tags to regroup declarations that are to appear together in the manual.

declarations

```
declarations
```

__init__

```
__init__(self, file, line, type, name)
```

accept

```
accept(self, visitor)
```

class Inheritance

Inheritance class. This class encapsulates the information about an inheritance, such as attributes like 'virtual' and 'public'

type

```
type
```

parent

```
parent
```

attributes

```
attributes
```

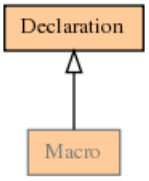
__init__

```
__init__(self, type, parent, attributes)
```

accept

```
accept(self, visitor)
```

class Macro



A preprocessor macro. Note that macros are not strictly part of the ASG, and as such are always in the global scope. A macro is "temporary" if it was #undefined in the same file it was #defined in.

parameters

```
parameters
```

text

```
text
```

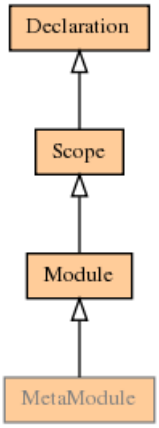
__init__

```
__init__(self, file, line, type, name, parameters, text)
```

accept

```
accept(self, visitor)
```

class MetaModule



Module Class that references all places where this Module occurs

module_declarations

```
module_declarations
```

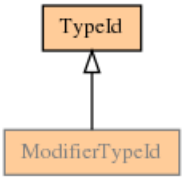
__init__

```
__init__(self, type, name)
```

accept

```
accept(self, visitor)
```

class ModifierTypeId



Class for alias types with modifiers (such as 'const', '&', etc.)

__cmp__

```
__cmp__(self, other)
```

Comparison operator

alias

```
alias
```

premod

```
premod
```

postmod

```
postmod
```

__init__

```
__init__(self, language, alias, premod, postmod)
```

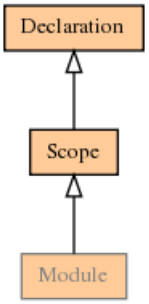
accept

```
accept(self, visitor)
```

__str__

```
__str__(self)
```

class Module



Module class

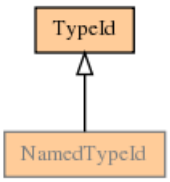
__init__

```
__init__(self, file, line, type, name)
```

accept

```
accept(self, visitor)
```

class NamedTypeId



Named type abstract class

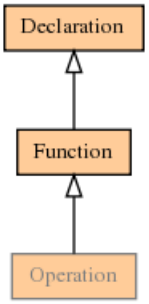
name

```
name
```

__init__

```
__init__(self, language, name)
```

class Operation



Operation class. An operation is related to a Function and is currently identical.

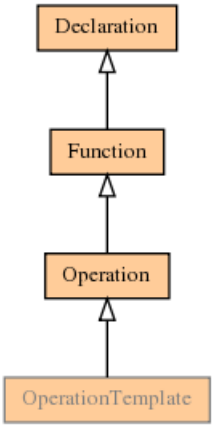
__init__

```
__init__(self, file, line, type, premod, return_type, postmod, name, \
real_name)
```

accept

```
accept(self, visitor)
```

class OperationTemplate



template

```
template
```

__init__

```
__init__(self, file, line, type, premod, return_type, postmod, name, \
real_name, template = None)
```

accept

```
accept(self, visitor)
```

PRIVATE

```
PRIVATE
```

PROTECTED

```
PROTECTED
```

PUBLIC

```
PUBLIC
```

class Parameter

Function Parameter

__cmp__

```
__cmp__(self, other)
```

Comparison operator

premodifier

```
premodifier
```

type

```
type
```

postmodifier

```
postmodifier
```

name

```
name
```

value

```
value
```

__init__

```
__init__(self, premod, type, postmod, name = '', value = '')
```

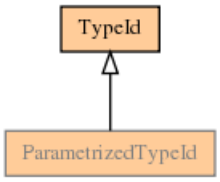
accept

```
accept(self, visitor)
```

__str__

```
__str__(self)
```

class ParametrizedTypeId



Class for parametrized type-id instances.

__cmp__

```
__cmp__(self, other)
```

Comparison operator

template

```
template
```

parameters

```
parameters
```

__init__

```
__init__(self, language, template, parameters)
```

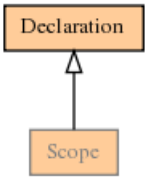
accept

```
accept(self, visitor)
```

__str__

```
__str__(self)
```

class Scope



Base class for scopes (named groups).

declarations

```
declarations
```

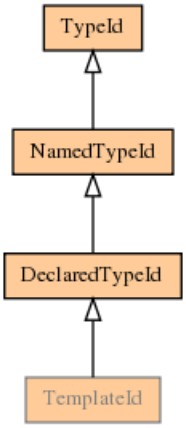
__init__

```
__init__(self, file, line, type, name)
```

accept

```
accept(self, visitor)
```

class TemplateId



Class for template-ids.

__cmp__

```
__cmp__(self, other)
```

Comparison operator

parameters

```
parameters
```

__init__

```
__init__(self, language, name, declaration, parameters)
```

accept

```
accept(self, visitor)
```

__str__

```
__str__(self)
```

class TypeId

Type-id abstract class.

__cmp__

```
__cmp__(self, other)
```

Comparison operator

language

```
language
```

__init__

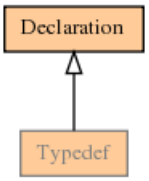
```
__init__(self, language)
```

accept

```
accept(self, visitor)
```

visitor pattern accept. @see Visitor

class Typedef



alias

```
alias
```

constr

```
constr
```

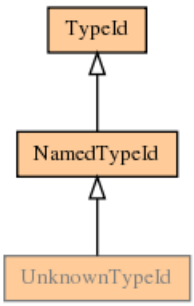
__init__

```
__init__(self, file, line, type, name, alias, constr)
```

accept

```
accept(self, visitor)
```

class UnknownTypeId



Class for not (yet) known type-ids.

__cmp__

```
__cmp__(self, other)
```

Comparison operator

link

```
link
```

base

```
base
```

__init__

```
__init__(self, language, name)
```

resolve

```
resolve(self, language, name, link)
```

Associate this type-id with an external reference, instead of a declaration.

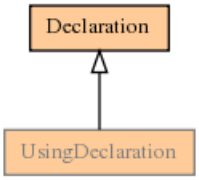
accept

```
accept(self, visitor)
```

__str__

```
__str__(self)
```

class UsingDeclaration



Import a declaration into this module.

alias

```
alias
```

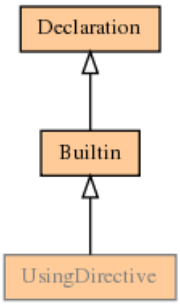
__init__

```
__init__(self, file, line, type, name, alias)
```

accept

```
accept(self, visitor)
```

class UsingDirective

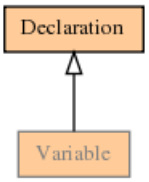


Import one module's content into another.

accept

```
accept(self, visitor)
```

class Variable



Variable definition

vtype

```
vtype
```

constr

```
constr
```

__init__

```
__init__(self, file, line, type, name, vtype, constr)
```

accept

```
accept(self, visitor)
```

class Visitor

Visitor for ASG nodes

visit_builtin_type_id

```
visit_builtin_type_id(self, type)
```

visit_unknown_type_id

```
visit_unknown_type_id(self, type)
```

visit_declared_type_id

```
visit_declared_type_id(self, type)
```

visit_modifier_type_id

```
visit_modifier_type_id(self, type)
```

visit_array_type_id

```
visit_array_type_id(self, type)
```

visit_template_id

```
visit_template_id(self, type)
```

visit_parametrized_type_id

```
visit_parametrized_type_id(self, type)
```

visit_function_type_id

```
visit_function_type_id(self, type)
```

visit_dependent_type_id

```
visit_dependent_type_id(self, type)
```

visit_declaration

```
visit_declaration(self, node)
```

visit_builtin

```
visit_builtin(self, node)
```

Visit a Builtin instance. By default do nothing. Processors who operate on Builtin nodes have to provide an appropriate implementation.

visit_using_directive

```
visit_using_directive(self, node)
```

visit_using_declaration

```
visit_using_declaration(self, node)
```

visit_macro

```
visit_macro(self, node)
```

visit_forward

```
visit_forward(self, node)
```

visit_group

```
visit_group(self, node)
```

visit_scope

```
visit_scope(self, node)
```

visit_module

```
visit_module(self, node)
```

visit_meta_module

```
visit_meta_module(self, node)
```

visit_class

```
visit_class(self, node)
```

visit_class_template

```
visit_class_template(self, node)
```

visit_typedef

```
visit_typedef(self, node)
```

visit_enumerator

```
visit_enumerator(self, node)
```

visit_enum

```
visit_enum(self, node)
```

visit_variable

```
visit_variable(self, node)
```

visit_const

```
visit_const(self, node)
```

visit_function

```
visit_function(self, node)
```

visit_function_template

```
visit_function_template(self, node)
```

visit_operation

```
visit_operation(self, node)
```

visit_operation_template

```
visit_operation_template(self, node)
```

visit_parameter

```
visit_parameter(self, node)
```

visit_inheritance

```
visit_inheritance(self, node)
```

ccmp

```
ccmp(a, b)
```

Compares classes of two objects

The Processor Framework

Modules

- Processor
- process

Module Processor

class Error

An exception a processor may raise during processing.

what

```
what
```

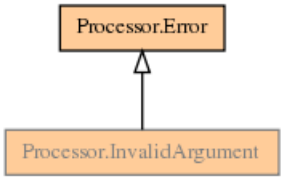
__init__

```
__init__(self, what)
```

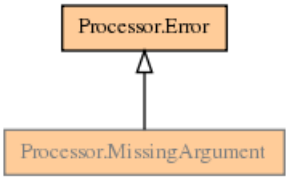
__str__

```
__str__(self)
```

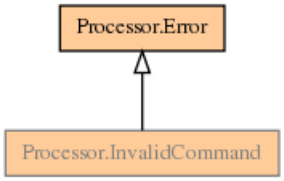
class InvalidArgument



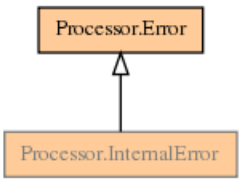
class MissingArgument



class InvalidCommand



class InternalError



class Parameter

A Parameter is a documented value, kept inside a Processor.

value

```
value
```

doc

```
doc
```

__init__

```
__init__(self, value, doc)
```

class Type

Type is the Processor's __metaclass__.

__init__

```
__init__(cls, name, bases, dict)
```

Generate a '_parameters' dictionary holding all the 'Parameter' objects. Then replace 'Parameter' objects by their values for convenient use inside the code.

class Parametrized

Parametrized implements handling of Parameter attributes.

__metaclass__

```
__metaclass__
```

__new__

```
__new__(cls, args, kwds)
```

merge all parameter catalogs for easy access to documentation, then use keyword arguments to override default values.

__init__

```
__init__(self, kwds)
```

The constructor uses the keywords to update the parameter list.

clone

```
clone(self, args, kwds)
```

Create a copy of this Parametrized. The only copied attributes are the ones corresponding to parameters.

get_parameters

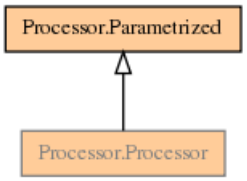
```
get_parameters(self)
```

set_parameters

```
set_parameters(self, kwds)
```

Sets the given parameters to override the default values.

class Processor



Processor documentation...

verbose

```
verbose
```

debug

```
debug
```

profile

```
profile
```

input

```
input
```

output

```
output
```

merge_input

```
merge_input(self, ir)
```

Join the given IR with a set of IRs to be read from 'input' parameter

output_and_return_ir

```
output_and_return_ir(self)
```

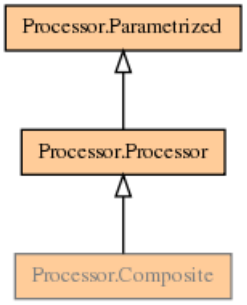
writes output if the 'output' attribute is set, then returns

process

```
process(self, ir, kwds)
```

The process method provides the interface to be implemented by subclasses. Commonly used arguments are 'input' and 'output'. If 'input' is defined, it is interpreted as one or more input file names. If 'output' is defined, it is interpreted as an output file (or directory) name. This implementation may serve as a template for real processors.

class Composite



A Composite processor.

processors

```
processors
```

__init__

```
__init__(self, processors, kwds)
```

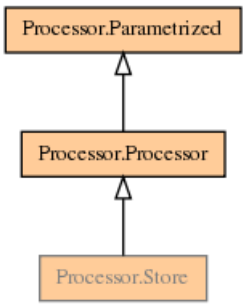
This __init__ is a convenience constructor that takes a var list to list the desired processors. If the named values contain 'processors', they override the var list.

process

```
process(self, ir, kwds)
```

apply a list of processors. The 'input' value is passed to the first processor only, the 'output' to the last. 'verbose' and 'debug' are passed down if explicitely given as named values. All other keywords are ignored.

class Store



Store is a convenience class useful to write out the intermediate state of the IR within a pipeline such as represented by the 'Composite'

process

```
process(self, ir, kwds)
```

Simply store the current IR in the 'output' file.

Module process

error

```
error(msg)
```

Write an error message and exit.

process

```
process(argv = sys.argv, commands)
```

Accept a set of commands and process according to command line options. The typical call will start with the name of the processor to be executed, followed by a set of parameters, followed by non-parameter arguments. All parameters are either of the form 'name=value', or '--name=value'. The first form expects 'value' to be valid python, the second a string. The remaining non-parameter arguments are associated with the 'input' parameter. Once this initialization is done, the named command's 'process' method is executed.

ASG Processors

Modules

- AccessRestrictor
- Comments
- Linker
- MacroFilter
- ModuleFilter

- ModuleSorter
- NameMapper
- SXRCompiler
- ScopeStripper
- TemplateLinker
- Transformer
- TypeMapper
- TypedefFolder

Module AccessRestrictor

class AccessRestrictor

This class processes declarations, and removes those that need greated access than the maximum passed to the constructor

__scopestack

```
__scopestack
```

__currscope

```
__currscope
```

access

```
access
```

__init__

```
__init__(self, kwds)
```

process

```
process(self, ir, kwds)
```

push

```
push(self)
```

pop

```
pop(self, decl)
```

add

```
add(self, decl)
```

visit_declaration

```
visit_declaration(self, decl)
```

visit_scope

```
visit_scope(self, scope)
```

Package Comments

Modules

- Comments.Filter
- Comments.Grouper
- Comments.Previous
- Comments.Translator

Module Comments.Filter

class Filter

Base class for comment filters.

visit_builtin

```
visit_builtin
```

visit_sourcefile

```
visit_sourcefile
```

process

```
process(self, ir, kwds)
```

visit_declaration

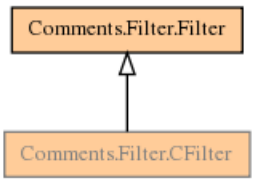
```
visit_declaration(self, decl)
```

filter_comment

```
filter_comment(self, comment)
```

Filter comment.

class CFilter



A class that filters C-style comments.

comment

```
comment
```

line

```
line
```

__init__

```
__init__(self, kwds)
```

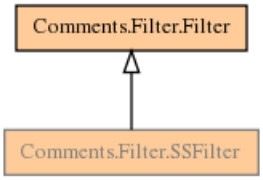
Compiles the regular expressions

filter_comment

```
filter_comment(self, comment)
```

Finds comments in the C format. The format is /* ... */. It has to cater for all five line forms: "/* ...", " * ...", " ...", " */" and the one-line "/* ... */".

class SSFilter



A class that selects only // comments.

ss

```
ss
```

__init__

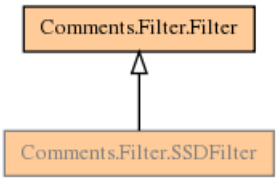
```
__init__(self, kwds)
```

Compiles the regular expressions

filter_comment

```
filter_comment(self, comment)
```

class SSDFilter



A class that selects only //. comments.

ssd

```
ssd
```

__init__

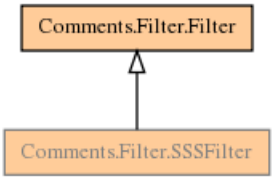
```
__init__(self, kwds)
```

Compiles the regular expressions

filter_comment

```
filter_comment(self, comment)
```

class SSSFilter



A class that selects only /// comments.

sss

```
sss
```

__init__

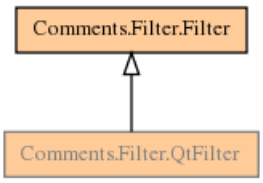
```
__init__(self, kwds)
```

Compiles the regular expressions

filter_comment

```
filter_comment(self, comment)
```

class QtFilter



A class that finds Qt style comments. These have two styles: //! ... and /*! ... */. The first means "brief comment" and there must only be one. The second type is the detailed comment.

brief

```
brief
```

detail

```
detail
```

__init__

```
__init__(self, kwds)
```

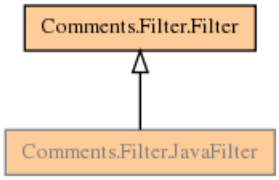
Compiles the regular expressions

filter_comment

```
filter_comment(self, comment)
```

Matches either brief or detailed comments.

class JavaFilter



A class that selects java /** style comments

java

```
java
```

line

```
line
```

__init__

```
__init__(self)
```

Compiles the regular expressions

filter_comment

```
filter_comment(self, comment)
```

Finds comments in the java format. The format is /** ... */, and it has to cater for all four line forms: "/** ...", " * ...", " */" and the one-line "/** ... */".

Module Comments.Grouper

class Grouper

A class that detects grouping tags and moves the enclosed nodes into a subnode (a 'Group')

__group_stack

```
__group_stack
```

tags

```
tags
```

__init__

```
__init__(self, kwds)
```

strip_dangling_groups

```
strip_dangling_groups(self)
```

As groups must not overlap with 'real' scopes, make sure all groups created in the current scope are closed when leaving the scope.

finalize

```
finalize(self)
```

replace the ASG with the newly created one

push

```
push(self)
```

starts a new group stack to be able to validate group scopes

pop

```
pop(self, decl)
```

Make sure the current group stack is empty.

push_group

```
push_group(self, group)
```

Push new group scope to the stack.

pop_group

```
pop_group(self, decl = None)
```

Pop a group scope from the stack. decl -- an optional declaration from which to extract the context, used for the error message if needed.

process_comments

```
process_comments(self, decl)
```

Checks for grouping tags. If an opening tag is found in the middle of a comment, a new Group is generated, the preceeding comments are associated with it, and is pushed onto the scope stack as well as the groups stack.

visit_declaration

```
visit_declaration(self, decl)
```

visit_scope

```
visit_scope(self, scope)
```

Visits all children of the scope in a new scope. The value of current_scope() at the end of the list is used to replace scope's list of declarations - hence you can remove (or insert) declarations from the list.

visit_enum

```
visit_enum(self, enum)
```

Does the same as visit_scope, but for the enum's list of enumerators

visit_enumerator

```
visit_enumerator(self, enumor)
```

Removes dummy enumerators

Module Comments.Previous

class Previous

A class that maps comments that begin with '<' to the previous declaration

process

```
process(self, ir, kwds)
```

decorates process() to initialise last and laststack

push

```
push(self)
```

decorates push() to also push 'last' onto 'laststack'

pop

```
pop(self)
```

decorates pop() to also pop 'last' from 'laststack'

visit_scope

```
visit_scope(self, scope)
```

overrides visit_scope() to set 'last' after each declaration

visit_declaration

```
visit_declaration(self, decl)
```

visit_builtin

```
visit_builtin(self, decl)
```

visit_enum

```
visit_enum(self, enum)
```

Does the same as visit_scope but for enum and enumerators

visit_enumerator

```
visit_enumerator(self, enumor)
```

Checks previous comment and removes dummies

process_comments

```
process_comments(self, decl)
```

Checks a decl to see if the comment should be moved. If the comment begins with a less-than sign, then it is moved to the 'last' declaration

Module Comments.Translator

class Translator

A Translator translates comments into documentation.

filter

```
filter
```

processor

```
processor
```

markup

```
markup
```

concatenate

```
concatenate
```

primary_only

```
primary_only
```

process

```
process(self, ir, kwds)
```

visit_declaration

```
visit_declaration(self, decl)
```

Map comments to a doc string.

visit_sourcefile

```
visit_sourcefile(self, sf)
```

Map comments to a doc string.

Module Linker

class Linker

Visitor that removes duplicate declarations

remove_empty_modules

```
remove_empty_modules
```

sort_modules

```
sort_modules
```

sxr_prefix

```
sxr_prefix
```

visit_declaration

```
visit_declaration
```

visit_using_declaration

```
visit_using_declaration
```

visit_forward

```
visit_forward
```

visit_enum

```
visit_enum
```

visit_operation

```
visit_operation
```

process

```
process(self, ir, kwds)
```

lookup

```
lookup(self, name)
```

look whether the current scope already contains a declaration with the given name

append

```
append(self, declaration)
```

append declaration to the current scope

push

```
push(self, scope)
```

push new scope on the stack

pop

```
pop(self)
```

restore the previous scope

top

```
top(self)
```

top_dict

```
top_dict(self)
```

link_type

```
link_type(self, type)
```

Returns the same or new proxy type

visit_builtin_type_id

```
visit_builtin_type_id(self, type)
```

visit_unknown_type_id

```
visit_unknown_type_id(self, type)
```

visit_declared_type_id

```
visit_declared_type_id(self, type)
```

visit_template_id

```
visit_template_id(self, type)
```

visit_modifier_type_id

```
visit_modifier_type_id(self, type)
```

visit_array_type_id

```
visit_array_type_id(self, type)
```

visit_parametrized_type_id

```
visit_parametrized_type_id(self, type)
```

visit_function_type_id

```
visit_function_type_id(self, type)
```

visit_source_file

```
visit_source_file(self, file)
```

Resolves any duplicates in the list of declarations from this file

visit_module

```
visit_module(self, module)
```

visit_group

```
visit_group(self, group)
```

merge_comments

```
merge_comments(self, metamodule, module)
```

Append the module comments into the metamodule.

visit_meta_module

```
visit_meta_module(self, module)
```

add_declaration

```
add_declaration(self, decl)
```

Adds a declaration to the current (top) scope. If there is already a Forward declaration, then this replaces it unless this is also a Forward.

visit_builtin

```
visit_builtin(self, builtin)
```

preserve builtins unconditionally

visit_named_type

```
visit_named_type(self, decl)
```

visit_function

```
visit_function(self, func)
```

visit_variable

```
visit_variable(self, var)
```

visit_typedef

```
visit_typedef(self, tdef)
```

visit_class

```
visit_class(self, class_)
```

visit_inheritance

```
visit_inheritance(self, parent)
```

visit_parameter

```
visit_parameter(self, param)
```

visit_const

```
visit_const(self, const)
```

Module MacroFilter

class MacroFilter

A MacroFilter allows macros to be filtered, based on pattern matching. Macros with matching names will be removed.

pattern

```
pattern
```

process

```
process(self, ir, kwds)
```

visit_macro

```
visit_macro(self, node)
```

Module ModuleFilter

class ModuleFilter

A processor that filters modules.

modules

```
modules
```

remove_empty

```
remove_empty
```

visit_builtin

```
visit_builtin
```

visit_group

```
visit_group
```

visit_scope

```
visit_scope
```

visit_enum

```
visit_enum
```

process

```
process(self, ir, kwds)
```

push

```
push(self)
```

Pushes the current scope onto the stack and starts a new one

pop

```
pop(self, decl)
```

Pops the current scope from the stack, and appends the given declaration to it

pop_only

```
pop_only(self)
```

Only pops, doesn't append to scope

add

```
add(self, decl)
```

Adds the given decl to the current scope

visit_declaration

```
visit_declaration(self, decl)
```

Adds declaration to scope

visit_module

```
visit_module(self, module)
```

Visits all children of the module, and if there are no declarations after that removes the module

Module ModuleSorter

class ModuleSorter

A processor that sorts declarations in a module alphabetically.

process

```
process(self, ir, kwds)
```

visit_meta_module

```
visit_meta_module(self, module)
```

Visits all children of the module, and if there are no declarations after that removes the module

Module NameMapper

class NameMapper

Abstract base class for name mapping.

visit_scope

```
visit_scope(self, node)
```

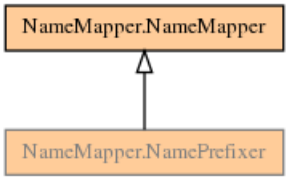
Recursively visits declarations under this scope.

visit_group

```
visit_group(self, node)
```

Recursively visits declarations under this group.

class NamePrefixer



This class adds a prefix to all declaration and type names.

prefix

```
prefix
```

type

```
type
```

process

```
process(self, ir, kwds)
```

visit_declaration

```
visit_declaration(self, decl)
```

Changes the name of this declaration and its associated type

Module SXRCompiler

class SXRCompiler

This class compiles symbol references stored in sxr files into a single symbol table.

prefix

```
prefix
```

no_locals

```
no_locals
```

process

```
process(self, ir, kwds)
```

compile

```
compile(self, filename, language)
```

Module ScopeStripper

class ScopeStripper

Strip common prefix from the declaration's name. Keep a list of root nodes, such that children whos parent scopes are not accepted but which themselfs are correct can be maintained as new root nodes.

declarations

```
declarations
```

inside

```
inside
```

_scope

```
_scope
```

scope

```
scope
```

__init__

```
__init__(self, kwds)
```

process

```
process(self, ir, kwds)
```

strip_name

```
strip_name(self, name)
```

strip_declarations

```
strip_declarations(self, declarations)
```

strip_types

```
strip_types(self, types)
```

strip

```
strip(self, declaration)
```

test whether the declaration matches one of the prefixes, strip it off, and return success. Success means that the declaration matches the prefix set and thus should not be removed from the ASG.

visit_scope

```
visit_scope(self, scope)
```

visit_class

```
visit_class(self, class_)
```

visit_class_template

```
visit_class_template(self, class_)
```

visit_declaration

```
visit_declaration(self, decl)
```

visit_enumerator

```
visit_enumerator(self, enumerator)
```

visit_enum

```
visit_enum(self, enum)
```

visit_function

```
visit_function(self, function)
```

visit_parameter

```
visit_parameter(self, parameter)
```

visit_function_template

```
visit_function_template(self, function)
```

visit_operation

```
visit_operation(self, operation)
```

visit_operation_template

```
visit_operation_template(self, operation)
```

visit_meta_module

```
visit_meta_module(self, module)
```

Module TemplateLinker

class TemplateLinker

Link template specializations to their primary templates, and vice versa.

visit_forward

```
visit_forward
```

visit_class

```
visit_class
```

visit_class_template

```
visit_class_template
```

process

```
process(self, ir, kwds)
```

link

```
link(self, d)
```

Module Transformer

class Transformer

A class that creates a new ASG from an old one. This is a helper base for more specialized classes that manipulate the ASG based on the comments in the nodes

__scopes

```
__scopes
```

__current

```
__current
```

__init__

```
__init__(self, kwds)
```

Constructor

process

```
process(self, ir, kwds)
```

finalize

```
finalize(self)
```

replace the ASG with the newly created one

push

```
push(self)
```

Pushes the current scope onto the stack and starts a new one

pop

```
pop(self, decl)
```

Pops the current scope from the stack, and appends the given declaration to it

add

```
add(self, decl)
```

Adds the given decl to the current scope

current_scope

```
current_scope(self)
```

Returns the current scope: a list of declarations

visit_builtin

```
visit_builtin(self, decl)
```

Module TypeMapper

class TypeMapper

Base class for type mapping

process

```
process(self, ir, kwds)
```

Module TypedefFolder

class TypedefFolder

Fold (anonymous) types into enclosing typedefs.

anonymous_only

```
anonymous_only
```

visit_group

```
visit_group
```

process

```
process(self, ir, kwds)
```

visit_scope

```
visit_scope(self, s)
```

visit_typedef

```
visit_typedef(self, t)
```

Parsers

Packages

- C
- Cpp
- Cxx
- IDL
- Python

Package C

Packages

- C.C

Package C.C

class Parser

preprocess

```
preprocess
```

emulate_compiler

```
emulate_compiler
```

compiler_flags

```
compiler_flags
```

cppflags

```
cppflags
```

primary_file_only

```
primary_file_only
```

base_path

```
base_path
```

sxr_prefix

```
sxr_prefix
```

process

```
process(self, ir, kwds)
```

Package Cpp

Packages

- Cpp.Cpp
- Cpp.Emulator

Package Cpp.Cpp

class Parser

emulate_compiler

```
emulate_compiler
```

compiler_flags

```
compiler_flags
```

flags

```
flags
```

primary_file_only

```
primary_file_only
```

cpp_output

```
cpp_output
```

base_path

```
base_path
```

language

```
language
```

probe

```
probe(self, kwds)
```

process

```
process(self, ir, kwds)
```

Module Cpp.Emulator

__docformat__

```
__docformat__
```

class TempFile

__del__

```
__del__(self)
```

name

```
name
```

file

```
file
```

__init__

```
__init__(self, suffix)
```

class CompilerInfo

Info about one compiler.

_write

```
_write(self, os)
```

compiler

```
compiler
```

The name of the compiler, typically the executable name, which must either be in the path or given as an absolute, pathname.

flags

```
flags
```

Compiler flags that impact its characteristics.

language

```
language
```

The programming language the compiler is used for.

kind

```
kind
```

A string indicating the type of this info: one of 'system', 'custom', ''. 'custom' compilers will never be automatically updated, and an empty string indicates a failure to look up the given compiler.

timestamp

```
timestamp
```

The timestamp of the compiler binary.

include_paths

```
include_paths
```

A list of strings indicating the include paths.

macros

```
macros
```

A list of (name,value) pairs. Values may be empty, or None. The latter ase indicates that the macro is to be undefined.

class CompilerList

_query

```
_query(self, language, compiler, flags)
```

Construct and return a CompilerInfo object for the given compiler.

compilers

```
compilers
```

no_cache

```
no_cache
```

user_emulations_file

```
user_emulations_file
```

The cache file.

__init__

```
__init__(self, filename = '')
```

list

```
list(self)
```

add_default_compilers

```
add_default_compilers(self)
```

load

```
load(self, filename = '')
```

Loads the compiler infos from a file.

save

```
save(self, filename = '')
```

refresh

```
refresh(self)
```

Refreshes the compiler list. Regenerate all non-custom compilers without destroying custom compilers.

find

```
find(self, language, compiler, flags)
```

find_ms_compiler_info

```
find_ms_compiler_info()
```

Try to find a (C++) MSVC compiler. Return tuple of include path list and macro dictionary.

find_gcc_compiler_info

```
find_gcc_compiler_info(language, compiler, flags)
```

Try to find a GCC-based C or C++ compiler. Return tuple of include path list and macro dictionary.

find_compiler_info

```
find_compiler_info(language, compiler, flags)
```

get_compiler_timestamp

```
get_compiler_timestamp(compiler)
```

Returns the timestamp for the given compiler, or 0 if not found

get_compiler_info

```
get_compiler_info(language, compiler = '', flags = None)
```

Returns the compiler info for the given compiler. If none is specified (''), return the first available one for the given language. The info is returned as a CompilerInfo object, or None if the compiler isn't found.

compiler_list

```
compiler_list
```

Package Cxx

Packages

- Cxx.Cxx

Package Cxx.Cxx

class Parser

preprocess

```
preprocess
```

emulate_compiler

```
emulate_compiler
```

compiler_flags

```
compiler_flags
```

cppflags

```
cppflags
```

primary_file_only

```
primary_file_only
```

base_path

```
base_path
```

sxr_prefix

```
sxr_prefix
```

process

```
process(self, ir, kwds)
```

Package IDL

Packages

- IDL.IDL
- IDL.idlast
- IDL.idltype
- IDL.idlutil
- IDL.idlvisitor
- IDL.omni

Package IDL.IDL

class Parser

preprocess

```
preprocess
```

cppflags

```
cppflags
```

primary_file_only

```
primary_file_only
```

base_path

```
base_path
```

process

```
process(self, ir, kwds)
```

Module IDL.idlast

class AST

Class for top-level Abstract Syntax Tree. Functions: file() -- the file name of the main IDL file. declarations() -- list of Decl objects corresponding to declarations at file scope. pragmas() -- list of Pragma objects containing #pragmas which occurred before any declarations. Later #pragmas are attached to Decl objects. comments() -- list of Comment objects containing comments which occurred before any declarations. accept(visitor) -- visitor pattern accept. See idlvisitor.py.

__file

```
__file
```

__declarations

```
__declarations
```

__pragmas

```
__pragmas
```

__comments

```
__comments
```

__init__

```
__init__(self, file, declarations, pragmas, comments)
```

file

```
file(self)
```

declarations

```
declarations(self)
```

pragmas

```
pragmas(self)
```

comments

```
comments(self)
```

accept

```
accept(self, visitor)
```

class Decl

Base class for all declarations. Functions: file() -- the IDL file this declaration came from. line() -- the line number within the file. mainFile() -- boolean: true if the file was the main IDL file; false if it was an included file. pragmas() -- list of Pragma objects containing #pragmas which immediately followed this declaration. comments() -- list of Comment objects containing comments which immediately followed this declaration. fullDecl() -- the 'full' Decl for typedefs, forwards, etc. accept(visitor) -- visitor pattern accept. See idlvisitor.py.

__file

```
__file
```

__line

```
__line
```

__mainFile

```
__mainFile
```

__builtIn

```
__builtIn
```

__pragmas

```
__pragmas
```

__comments

```
__comments
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments)
```

accept

```
accept(self, visitor)
```

file

```
file(self)
```

line

```
line(self)
```

mainFile

```
mainFile(self)
```

builtIn

```
builtIn(self)
```

pragmas

```
pragmas(self)
```

comments

```
comments(self)
```

fullDecl

```
fullDecl(self)
```

class DeclRepoId

Mixin class for Decls which have a Repository Id Functions: identifier() -- name of the declaration as a string scopedName() -- list of strings forming the fully-scoped name of the declaration. e.g. ::foo::bar::baz is represented as ['foo', 'bar', 'baz']. repoId() -- repository identifier for this declaration.

__identifier

```
__identifier
```

__scopedName

```
__scopedName
```

__repoId

```
__repoId
```

__init__

```
__init__(self, identifier, scopedName, repoId)
```

identifier

```
identifier(self)
```

scopedName

```
scopedName(self)
```

repoId

```
repoId(self)
```

class Pragma

Class containing information about an unknown pragma Functions: text() -- text of the pragma __str__() -- same as text() file() -- file containing the pragma line() -- line number in file

__text

```
__text
```

__file

```
__file
```

__line

```
__line
```

__init__

```
__init__(self, text, file, line)
```

text

```
text(self)
```

__str__

```
__str__(self)
```

file

```
file(self)
```

line

```
line(self)
```

class Comment

Class containing information about a comment Functions: text() -- text of the comment __str__() -- same as text() file() -- file containing the comment line() -- line number in file

__text

```
__text
```

__file

```
__file
```

__line

```
__line
```

__init__

```
__init__(self, text, file, line)
```

text

```
text(self)
```

__str__

```
__str__(self)
```

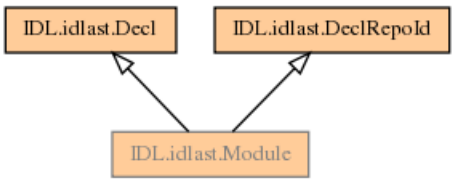
file

```
file(self)
```

line

```
line(self)
```

class Module



Module declaration (Decl, DeclRepoId) Functions: definitions() -- list of Decl objects declared within this module. continuations() -- list containing continuations of this module. When modules are re-opened, multiple Module objects with the same name appear in the enclosing Module or AST object. In case it's useful, the first Module object for a particular module has a list containing continuations of that module. You will probably not have any use for this.

__definitions

```
__definitions
```

_continuations

```
_continuations
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, definitions)
```

accept

```
accept(self, visitor)
```

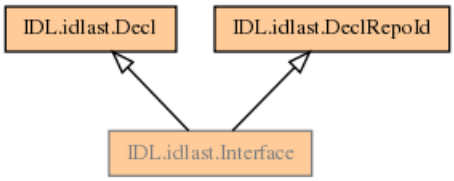
definitions

```
definitions(self)
```

continuations

```
continuations(self)
```

class Interface



Interface declaration (Decl, DeclRepoId) Functions: abstract() -- boolean: true if the interface is declared abstract. local() -- boolean: true if the interface is declared local. inherits() -- list of Interface objects from which this one inherits. contents() -- list of Decl objects for all items declared within this interface. declarations() -- subset of contents() containing types, constants and exceptions. callables() -- subset of contents() containing Operations and Attributes. all_callables()-- callables of this and inherited interfaces.

_setContents

```
_setContents(self, contents)
```

__abstract

```
__abstract
```

__local

```
__local
```

__inherits

```
__inherits
```

__contents

```
__contents
```

__declarations

```
__declarations
```

__callables

```
__callables
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, abstract, local, inherits)
```

accept

```
accept(self, visitor)
```

abstract

```
abstract(self)
```

local

```
local(self)
```

inherits

```
inherits(self)
```

contents

```
contents(self)
```

declarations

```
declarations(self)
```

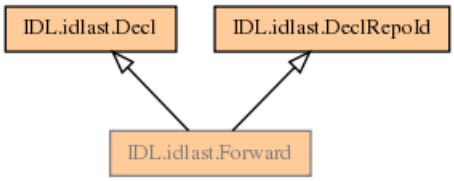
callables

```
callables(self)
```

all_callables

```
all_callables(self)
```

class Forward



Forward-declared interface (Decl, DeclRepoId) Functions: abstract() -- boolean: true if the interface is declared abstract. local() -- boolean: true if the interface is declared local. fullDecl() -- Interface object corresponding to full interface declaration or None if there is no full declaration.

__abstract

```
__abstract
```

__local

```
__local
```

_fullDecl

```
_fullDecl
```

_more

```
_more
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, abstract, local)
```

accept

```
accept(self, visitor)
```

abstract

```
abstract(self)
```

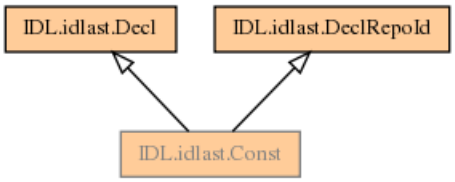
local

```
local(self)
```

fullDecl

```
fullDecl(self)
```

class Const



Constant declaration (Decl, DeclRepoId) Functions: constType() -- IdlType.Type object of this constant. Aliases not stripped. constKind() -- TypeCode kind of constant with aliases stripped. value() -- value of the constant. Either an integer or an Enumerator object.

__constType

```
__constType
```

__constKind

```
__constKind
```

__value

```
__value
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, constType, constKind, value)
```

accept

```
accept(self, visitor)
```

constType

```
constType(self)
```

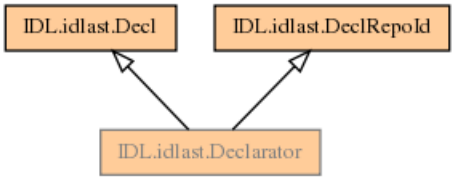
constKind

```
constKind(self)
```

value

```
value(self)
```

class Declarator



Declarator used in typedefs, struct members, etc. (Decl, DeclRepoId) Functions: sizes() -- list of array sizes, or None if this is a simple declarator. alias() -- Typedef object for this declarator if this is a typedef declarator. None otherwise.

_setAlias

```
_setAlias(self, alias)
```

__sizes

```
__sizes
```

__alias

```
__alias
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, sizes)
```

accept

```
accept(self, visitor)
```

sizes

```
sizes(self)
```

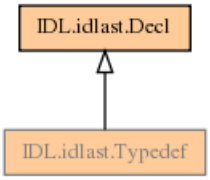
alias

```
alias(self)
```

fullDecl

```
fullDecl(self)
```

class Typedef



Typedef (Decl) Functions: aliasType() -- IdlType.Type object that this is an alias to. constrType() -- boolean: true if the alias type was constructed within this typedef declaration. declarators() -- list of Declarator objects.

__aliasType

```
__aliasType
```

__constrType

```
__constrType
```

__declarators

```
__declarators
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, aliasType, \
constrType, declarators)
```

accept

```
accept(self, visitor)
```

aliasType

```
aliasType(self)
```

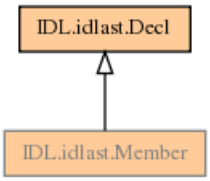
constrType

```
constrType(self)
```

declarators

```
declarators(self)
```

class Member



Member of a struct or exception (Decl) Functions: memberType() -- IdlType.Type object for the type of this member. constrType() -- boolean: true if the member type was constructed within the member declaration. declarators() -- list of Declarator objects.

__memberType

```
__memberType
```

__constrType

```
__constrType
```

__declarators

```
__declarators
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, memberType, \
constrType, declarators)
```

accept

```
accept(self, visitor)
```

memberType

```
memberType(self)
```

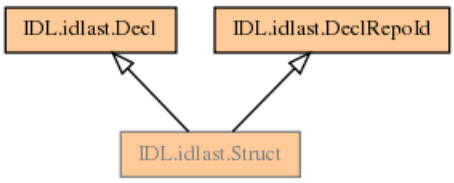
constrType

```
constrType(self)
```

declarators

```
declarators(self)
```

class Struct



Struct declaration (Decl, DeclRepoId) Functions: members() -- list of Member objects for the struct contents. recursive() -- boolean: true if the struct is recursive.

_setMembers

```
_setMembers(self, members)
```

__recursive

```
__recursive
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, recursive)
```

accept

```
accept(self, visitor)
```

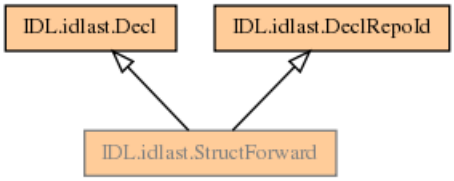
members

```
members(self)
```

recursive

```
recursive(self)
```

class StructForward



Struct forward declaration (Decl, DeclRepoId) Functions: fullDecl() -- full definition of the struct.

_fullDecl

```
_fullDecl
```

_more

```
_more
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId)
```

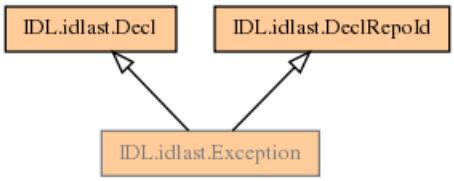
accept

```
accept(self, visitor)
```

fullDecl

```
fullDecl(self)
```

class Exception



Exception declaration (Decl, DeclRepoId) Function: members() -- list of Member objects for the exception contents.

__members

```
__members
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, members)
```

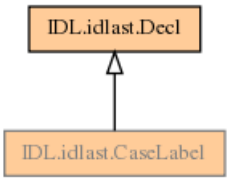
accept

```
accept(self, visitor)
```

members

```
members(self)
```

class CaseLabel



Case label within a union (Decl) Functions: default() -- boolean: true if this is the default label. value() -- label value. Either an integer or an Enumerator object. If default() is true, returns a value used by none of the other union labels. labelKind() -- TypeCode kind of label.

__default

```
__default
```

__value

```
__value
```

__labelKind

```
__labelKind
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, default, value, \
labelKind)
```

accept

```
accept(self, visitor)
```

default

```
default(self)
```

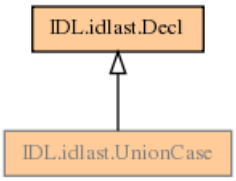
value

```
value(self)
```

labelKind

```
labelKind(self)
```

class UnionCase



One case within a union (Decl) Functions: labels() -- list of CaseLabel objects. caseType() -- IdlType.Type object for the case type. constrType() -- boolean: true if the case type was constructed within the case. declarator() -- Declarator object

__labels

```
__labels
```

__caseType

```
__caseType
```

__constrType

```
__constrType
```

__declarator

```
__declarator
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, labels, \
caseType, constrType, declarator)
```

accept

```
accept(self, visitor)
```

labels

```
labels(self)
```

caseType

```
caseType(self)
```

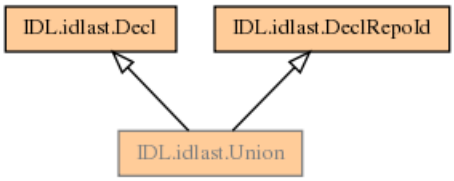
constrType

```
constrType(self)
```

declarator

```
declarator(self)
```

class Union



Union declaration (Decl, DeclRepoId) Functions: switchType() -- IdlType.Type object corresponding to the switch type. constrType() -- boolean: true if the switch type was declared within the switch statement. Only possible for Enums. cases() -- list of UnionCase objects. recursive() -- boolean: true if the union is recursive.

_setCases

```
_setCases(self, cases)
```

__switchType

```
__switchType
```

__constrType

```
__constrType
```

__recursive

```
__recursive
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, switchType, constrType, recursive)
```

accept

```
accept(self, visitor)
```

switchType

```
switchType(self)
```

constrType

```
constrType(self)
```

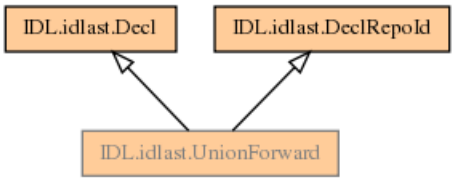
cases

```
cases(self)
```

recursive

```
recursive(self)
```

class UnionForward



Union forward declaration (Decl, DeclRepoId) Functions: fullDecl() -- full definition of the union.

_fullDecl

```
_fullDecl
```

_more

```
_more
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId)
```

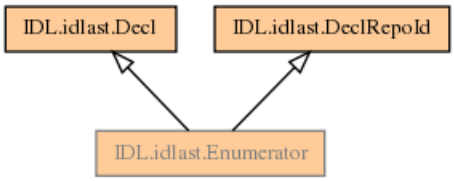
accept

```
accept(self, visitor)
```

fullDecl

```
fullDecl(self)
```

class Enumerator



Enumerator of an Enum (Decl, DeclRepoId) Function: value() -- integer value of enumerator, as marshalled.

__value

```
__value
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, value)
```

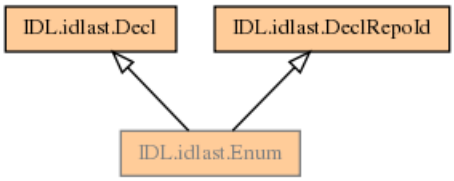
accept

```
accept(self, visitor)
```

value

```
value(self)
```

class Enum



Enum declaration (Decl, DeclRepoId) Function: enumerators() -- list of Enumerator objects.

__enumerators

```
__enumerators
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, enumerators)
```

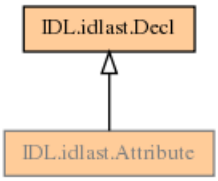
accept

```
accept(self, visitor)
```

enumerators

```
enumerators(self)
```

class Attribute



Attribute declaration (Decl) Functions: readonly() -- boolean: true if the attribute is read only. attrType() -- IdlType.Type object for the attribute's type. declarators() -- list of the attribute's declarators. identifiers() -- list of strings containing the attribute identifiers (equivalent to the identifiers inside the declarators).

__readonly

```
__readonly
```

__attrType

```
__attrType
```

__declarators

```
__declarators
```

__identifiers

```
__identifiers
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, readonly, \
attrType, declarators)
```

accept

```
accept(self, visitor)
```

readonly

```
readonly(self)
```

attrType

```
attrType(self)
```

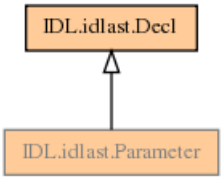
declarators

```
declarators(self)
```

identifiers

```
identifiers(self)
```

class Parameter



A Parameter of an operation or factory specifier (Decl) Functions: direction() -- integer: 0 == in, 1 == out, 2 == inout. is_in() -- boolean: true if in or inout. is_out() -- boolean: true if out or inout. paramType() -- IdlType.Type object for the parameter type. identifier() -- string of parameter identifier.

__direction

```
__direction
```

__is_in

```
__is_in
```

__is_out

```
__is_out
```

__paramType

```
__paramType
```

__identifier

```
__identifier
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, direction, \
paramType, identifier)
```

accept

```
accept(self, visitor)
```

direction

```
direction(self)
```

is_in

```
is_in(self)
```

is_out

```
is_out(self)
```

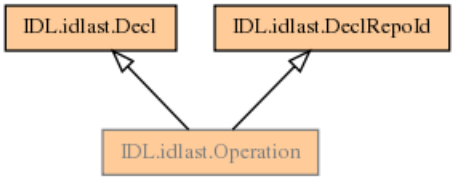
paramType

```
paramType(self)
```

identifier

```
identifier(self)
```

class Operation



Operation declaration (Decl, DeclRepoId) Functions: oneway() -- boolean: true if operation is one way. returnType() -- IdlType.Type object for return type. parameters() -- list of Parameter objects. raises() -- list of Exception objects. contexts() -- list of strings for context expressions.

__oneway

```
__oneway
```

__returnType

```
__returnType
```

__parameters

```
__parameters
```

__raises

```
__raises
```

__contexts

```
__contexts
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, oneway, \
returnType, identifier, scopedName, repoId, parameters, raises, contexts)
```

accept

```
accept(self, visitor)
```

oneway

```
oneway(self)
```

returnType

```
returnType(self)
```

parameters

```
parameters(self)
```

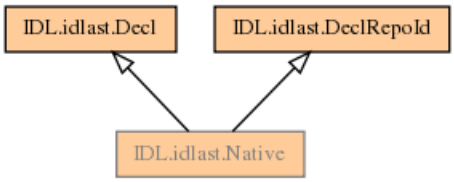
raises

```
raises(self)
```

contexts

```
contexts(self)
```

class Native



Native declaration (Decl, DeclRepoId) Native should not be used in normal IDL. No non-inherited functions.

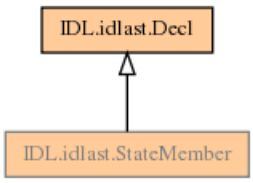
__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId)
```

accept

```
accept(self, visitor)
```

class StateMember



State member of a valuetype (Decl) Functions: memberAccess() -- integer: 0 == public, 1 == private. memberType() -- IdlType.Type object for member type. constrType() -- boolean: true if member type is declared within the StateMember. declarators() -- list of Declarator objects.

__memberAccess

```
__memberAccess
```

__memberType

```
__memberType
```

__constrType

```
__constrType
```

__declarators

```
__declarators
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, memberAccess, \
memberType, constrType, declarators)
```

accept

```
accept(self, visitor)
```

memberAccess

```
memberAccess(self)
```

memberType

```
memberType(self)
```

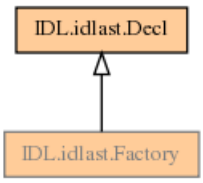
constrType

```
constrType(self)
```

declarators

```
declarators(self)
```

class Factory



Factory method of valuetype (Decl) Functions: identifier() -- string. parameters() -- list of Parameter objects. raises() -- list of Exception objects.

__identifier

```
__identifier
```

__parameters

```
__parameters
```

__raises

```
__raises
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
parameters, raises)
```

accept

```
accept(self, visitor)
```

identifier

```
identifier(self)
```

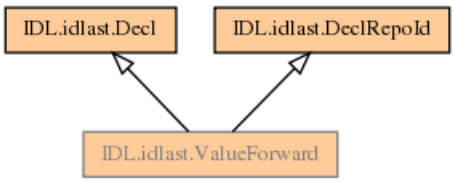
parameters

```
parameters(self)
```

raises

```
raises(self)
```

class ValueForward



Forward declared valuetype (Decl, DeclRepoId) Function: abstract() -- boolean: true if declared abstract. fullDecl() -- Value or ValueAbs object corresponding to the full valuetype declaration or None if there is no full declaration.

__abstract

```
__abstract
```

_fullDecl

```
_fullDecl
```

_more

```
_more
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, abstract)
```

accept

```
accept(self, visitor)
```

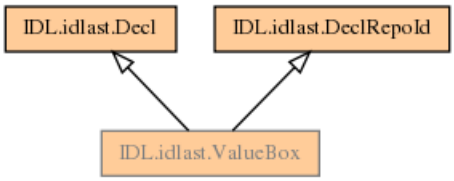
abstract

```
abstract(self)
```

fullDecl

```
fullDecl(self)
```

class ValueBox



ValueBox declaration (Decl, DeclRepoId) Functions: boxedType() -- IdlType.Type object for boxed type. constrType() -- boolean: true if boxed type is declared inside the ValueBox declaration.

__boxedType

```
__boxedType
```

__constrType

```
__constrType
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, boxedType, constrType)
```

accept

```
accept(self, visitor)
```

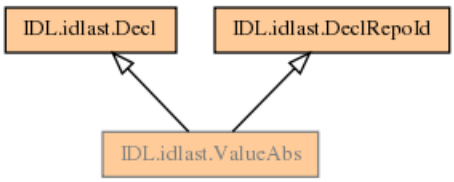
boxedType

```
boxedType(self)
```

constrType

```
constrType(self)
```

class ValueAbs



Abstract valuetype declaration (Decl, DeclRepoId) Functions: inherits() -- list of ValueAbs objects from which this inherits. supports() -- list of Interface objects which this supports. contents() -- list of Decl objects for declarations within this valuetype. declarations() -- subset of contents() containing types, constants and exceptions. callables() -- subset of contents() containing Operations and Attributes. statemembers() -- subset of contents() containing StateMembers. factories() -- subset of contents() containing Factory instances.

_setContents

```
_setContents(self, contents)
```

__inherits

```
__inherits
```

__supports

```
__supports
```

__contents

```
__contents
```

__declarations

```
__declarations
```

__callables

```
__callables
```

__statemembers

```
__statemembers
```

__factories

```
__factories
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, inherits, supports)
```

accept

```
accept(self, visitor)
```

inherits

```
inherits(self)
```

supports

```
supports(self)
```

contents

```
contents(self)
```

declarations

```
declarations(self)
```

callables

```
callables(self)
```

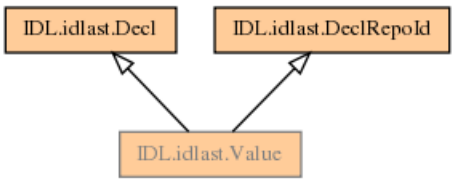
statemembers

```
statemembers(self)
```

factories

```
factories(self)
```

class Value



valuetype declaration (Decl, DeclRepoId) Functions: custom() -- boolean: true if declared custom. inherits() -- list of valuetypes from which this inherits. The first may be a Value object or a ValueAbs object; any others will be ValueAbs objects. truncatable() -- boolean: true if the inherited Value is declared truncatable. supports() -- list of Interface objects which this supports. contents() -- list of Decl objects for all items declared within this valuetype. declarations() -- subset of contents() containing types, constants and exceptions. callables() -- subset of contents() containing Operations and Attributes. statemembers() -- subset of contents() containing StateMembers. factories() -- subset of contents() containing Factory instances.

_setContents

```
_setContents(self, contents)
```

__custom

```
__custom
```

__inherits

```
__inherits
```

__truncatable

```
__truncatable
```

__supports

```
__supports
```

__contents

```
__contents
```

__declarations

```
__declarations
```

__callables

```
__callables
```

__statemembers

```
__statemembers
```

__factories

```
__factories
```

__init__

```
__init__(self, file, line, mainFile, pragmas, comments, identifier, \
scopedName, repoId, custom, inherits, truncatable, supports)
```

accept

```
accept(self, visitor)
```

custom

```
custom(self)
```

inherits

```
inherits(self)
```

truncatable

```
truncatable(self)
```

supports

```
supports(self)
```

contents

```
contents(self)
```

declarations

```
declarations(self)
```

callables

```
callables(self)
```

statemembers

```
statemembers(self)
```

factories

```
factories(self)
```

class DeclNotFound

Exception to indicate that findDecl() could not find the requested Decl object.

__scopedName

```
__scopedName
```

__init__

```
__init__(self, scopedName)
```

scopedName

```
scopedName(self)
```

registerDecl

```
registerDecl(scopedName, decl)
```

Private function

findDecl

```
findDecl(scopedName)
```

findDecl(scopedName) -> Decl Find a Decl object given a fully scoped name represented as a list of strings. Raises DeclNotFound if the name is not recognised.

clear

```
clear()
```

Clear back-end structures ready for another run

declMap

```
declMap
```

CORBAObject

```
CORBAObject
```

CORBAValueBase

```
CORBAValueBase
```

CORBAModule

```
CORBAModule
```

Module IDL.idltype

class Error

Exception class used by IdlType internals.

err

```
err
```

__init__

```
__init__(self, err)
```

__repr__

```
__repr__(self)
```

class Type

Type abstract class. Function: kind() -- TypeCode kind of type. unalias() -- Return an equivalent Type object with aliases stripped accept(visitor) -- visitor pattern accept. See idlvisitor.py.

__kind

```
__kind
```

__local

```
__local
```

__init__

```
__init__(self, kind, local)
```

kind

```
kind(self)
```

local

```
local(self)
```

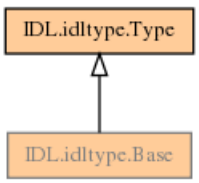
unalias

```
unalias(self)
```

accept

```
accept(self, visitor)
```

class Base



Class for CORBA base types. (Type) No non-inherited functions.

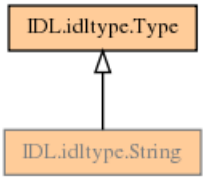
__init__

```
__init__(self, kind)
```

accept

```
accept(self, visitor)
```

class String



Class for string types (Type) Function: bound() -- bound of bounded string. 0 for unbounded.

__bound

```
__bound
```

__init__

```
__init__(self, bound)
```

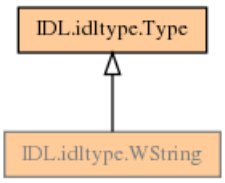
accept

```
accept(self, visitor)
```

bound

```
bound(self)
```

class WString



Class for wide string types (Type) Function: bound() -- bound of bounded wstring. 0 for unbounded.

__bound

```
__bound
```

__init__

```
__init__(self, bound)
```

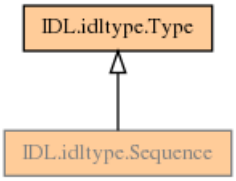
accept

```
accept(self, visitor)
```

bound

```
bound(self)
```

class Sequence



Class for sequence types (Type) Functions: seqType() -- Type this is a sequence of. bound() -- bound of bounded sequence. 0 for unbounded.

__seqType

```
__seqType
```

__bound

```
__bound
```

__init__

```
__init__(self, seqType, bound, local)
```

accept

```
accept(self, visitor)
```

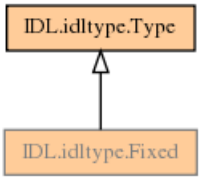
seqType

```
seqType(self)
```

bound

```
bound(self)
```

class Fixed



Class for fixed point types (Type) Functions: digits() -- digits. scale() -- scale.

__digits

```
__digits
```

__scale

```
__scale
```

__init__

```
__init__(self, digits, scale)
```

accept

```
accept(self, visitor)
```

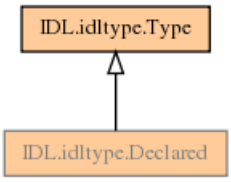
digits

```
digits(self)
```

scale

```
scale(self)
```

class Declared



Class for declared types (Type) Functions: decl() -- Decl object which corresponds to this type. scopedName() -- Fully scoped name of the type as a list of strings. name() -- Simple name of the type.

__decl

```
__decl
```

__scopedName

```
__scopedName
```

__init__

```
__init__(self, decl, scopedName, kind, local)
```

accept

```
accept(self, visitor)
```

decl

```
decl(self)
```

scopedName

```
scopedName(self)
```

name

```
name(self)
```

containsValueType

```
containsValueType(t, track = None)
```

Returns true if the type contains valuetypes

baseType

```
baseType(kind)
```

stringType

```
stringType(bound)
```

wstringType

```
wstringType(bound)
```

sequenceType

```
sequenceType(type_spec, bound, local)
```

fixedType

```
fixedType(digits, scale)
```

declaredType

```
declaredType(decl, scopedName, kind, local)
```

clear

```
clear()
```

Clear back-end structures ready for another run

tk_null

```
tk_null
```

tk_void

```
tk_void
```

tk_short

```
tk_short
```

tk_long

```
tk_long
```

tk_ushort

```
tk_ushort
```

tk_ulong

```
tk_ulong
```

tk_float

```
tk_float
```

tk_double

```
tk_double
```

tk_boolean

```
tk_boolean
```

tk_char

```
tk_char
```

tk_octet

```
tk_octet
```

tk_any

```
tk_any
```

tk_TypeCode

```
tk_TypeCode
```

tk_Principal

```
tk_Principal
```

tk_objref

```
tk_objref
```

tk_struct

```
tk_struct
```

tk_union

```
tk_union
```

tk_enum

```
tk_enum
```

tk_string

```
tk_string
```

tk_sequence

```
tk_sequence
```

tk_array

```
tk_array
```

tk_alias

```
tk_alias
```

tk_except

```
tk_except
```

tk_longlong

```
tk_longlong
```

tk_ulonglong

```
tk_ulonglong
```

tk_longdouble

```
tk_longdouble
```

tk_wchar

```
tk_wchar
```

tk_wstring

```
tk_wstring
```

tk_fixed

```
tk_fixed
```

tk_value

```
tk_value
```

tk_value_box

```
tk_value_box
```

tk_native

```
tk_native
```

tk_abstract_interface

```
tk_abstract_interface
```

tk_local_interface

```
tk_local_interface
```

ot_structforward

```
ot_structforward
```

ot_unionforward

```
ot_unionforward
```

baseTypeMap

```
baseTypeMap
```

stringTypeMap

```
stringTypeMap
```

wstringTypeMap

```
wstringTypeMap
```

sequenceTypeMap

```
sequenceTypeMap
```

fixedTypeMap

```
fixedTypeMap
```

declaredTypeMap

```
declaredTypeMap
```

Module IDL.idlutil

_valid_chars

```
_valid_chars
```

_valid_unichars

```
_valid_unichars
```

slashName

```
slashName(scopedName, our_scope = [])
```

slashName(list, [list]) -> string Return a scoped name given as a list of strings as a single string with the components separated by '/' characters. If a second list is given, remove a common prefix using pruneScope().

dotName

```
dotName(scopedName, our_scope = [])
```

dotName(list, [list]) -> string Return a scoped name given as a list of strings as a single string with the components separated by '.' characters. If a second list is given, remove a common prefix using pruneScope().

ccolonName

```
ccolonName(scopedName, our_scope = [])
```

ccolonName(list, [list]) -> string Return a scoped name given as a list of strings as a single string with the components separated by '::' strings. If a second list is given, remove a common prefix using pruneScope().

pruneScope

```
pruneScope(target_scope, our_scope)
```

pruneScope(list A, list B) -> list Given two lists of strings (scoped names), return a copy of list A with any prefix it shares with B removed. e.g. pruneScope(['A', 'B', 'C', 'D'], ['A', 'B', 'D']) -> ['C', 'D']

escapifyString

```
escapifyString(s)
```

escapifyString(string) -> string Return the given string with any non-printing characters escaped.

escapifyWString

```
escapifyWString(l, escchar = "u")
```

escapifyWString(int list) -> string Take a list of integers representing Unicode characters and return an ASCII string with all characters outside that range replaced with \u escapes.

reprFloat

```
reprFloat(f)
```

reprFloat(float) -> string Return the string representation of an IDL float type (float, double, long double), with enough precision to completely reconstruct the bit pattern.

relativeScope

```
relativeScope(fromScope, destScope)
```

relativeScope(fromScope, destScope) -> list Given two globally-scoped names, return a minimal scoped name list which identifies the destination scope, without clashing with another identifier. For example, given IDL: module M { typedef short A; typedef long B; module N { typedef string B; interface I { void op(in ::M::A x, in ::M::B y); }; }; }; relativeScope(["M", "N", "I"], ["M", "A"]) -> ["A"] relativeScope(["M", "N", "I"], ["M", "B"]) -> ["M", "B"] If the only valid result is a globally-scoped name, the result list is prefixed with None: module O { typedef short C; }; module P { module O { interface J { void op(in ::O::C z); }; }; }; relativeScope(["P", "O", "J"], ["O", "C"]) -> [None, "O", "C"] If either scoped name does not exist, returns None.

Module IDL.idlvisitor

class AstVisitor

Visitor for AST nodes Functions: visitAST(node) visitModule(node) visitInterface(node) visitForward(node) visitConst(node) visitDeclarator(node) visitTypedef(node) visitMember(node) visitStruct(node) visitStruct-Forward(node) visitException(node) visitCaseLabel(node) visitUnionCase(node) visitUnion(node) visitUnionForward(node) visitEnumerator(node) visitEnum(node) visitAttribute(node) visitParameter(node) visitOperation(node) visitNative(node) visitStateMember(node) visitFactory(node) visitValueForward(node) visitValueBox(node) visitValueAbs(node) visitValue(node)

visitAST

```
visitAST(self, node)
```

visitModule

```
visitModule(self, node)
```

visitInterface

```
visitInterface(self, node)
```

visitForward

```
visitForward(self, node)
```

visitConst

```
visitConst(self, node)
```

visitDeclarator

```
visitDeclarator(self, node)
```

visitTypedef

```
visitTypedef(self, node)
```

visitMember

```
visitMember(self, node)
```

visitStruct

```
visitStruct(self, node)
```

visitStructForward

```
visitStructForward(self, node)
```

visitException

```
visitException(self, node)
```

visitCaseLabel

```
visitCaseLabel(self, node)
```

visitUnionCase

```
visitUnionCase(self, node)
```

visitUnion

```
visitUnion(self, node)
```

visitUnionForward

```
visitUnionForward(self, node)
```

visitEnumerator

```
visitEnumerator(self, node)
```

visitEnum

```
visitEnum(self, node)
```

visitAttribute

```
visitAttribute(self, node)
```

visitParameter

```
visitParameter(self, node)
```

visitOperation

```
visitOperation(self, node)
```

visitNative

```
visitNative(self, node)
```

visitStateMember

```
visitStateMember(self, node)
```

visitFactory

```
visitFactory(self, node)
```

visitValueForward

```
visitValueForward(self, node)
```

visitValueBox

```
visitValueBox(self, node)
```

visitValueAbs

```
visitValueAbs(self, node)
```

visitValue

```
visitValue(self, node)
```

class TypeVisitor

Visitor for Type objects Functions: visitBaseType(type) visitStringType(type) visitWStringType(type) visitSequenceType(type) visitFixedType(type) visitDeclaredType(type)

visitBaseType

```
visitBaseType(self, type)
```

visitStringType

```
visitStringType(self, type)
```

visitWStringType

```
visitWStringType(self, type)
```

visitSequenceType

```
visitSequenceType(self, type)
```

visitFixedType

```
visitFixedType(self, type)
```

visitDeclaredType

```
visitDeclaredType(self, type)
```

Module IDL.omni

class TypeTranslator

maps idltype objects to ASG.TypeId objects in a ASG.Dictionary

types

```
types
```

__result

```
__result
```

__basetypes

```
__basetypes
```

__init__

```
__init__(self, types)
```

internalize

```
internalize(self, idltype)
```

has_key

```
has_key(self, name)
```

add

```
add(self, name, type)
```

get

```
get(self, name)
```

visitBaseType

```
visitBaseType(self, idltype)
```

visitStringType

```
visitStringType(self, idltype)
```

visitWStringType

```
visitWStringType(self, idltype)
```

visitSequenceType

```
visitSequenceType(self, idltype)
```

visitDeclaredType

```
visitDeclaredType(self, idltype)
```

class ASGTranslator

declarations

```
declarations
```

primary_file_only

```
primary_file_only
```

types

```
types
```

__scope

```
__scope
```

__operation

```
__operation
```

__enum

```
__enum
```

__init__

```
__init__(self, declarations, types, primary_file_only)
```

scope

```
scope(self)
```

add_declaration

```
add_declaration(self, declaration)
```

addType

```
addType(self, name, type)
```

getType

```
getType(self, name)
```

visitAST

```
visitAST(self, node)
```

visitModule

```
visitModule(self, node)
```

visitInterface

```
visitInterface(self, node)
```

visitForward

```
visitForward(self, node)
```

visitConst

```
visitConst(self, node)
```

visitTypedef

```
visitTypedef(self, node)
```

visitMember

```
visitMember(self, node)
```

visitStruct

```
visitStruct(self, node)
```

visitException

```
visitException(self, node)
```

visitUnionCase

```
visitUnionCase(self, node)
```

visitUnion

```
visitUnion(self, node)
```

visitEnumerator

```
visitEnumerator(self, node)
```

visitEnum

```
visitEnum(self, node)
```

visitAttribute

```
visitAttribute(self, node)
```

visitParameter

```
visitParameter(self, node)
```

visitOperation

```
visitOperation(self, node)
```

strip_filename

```
strip_filename(filename)
```

This is aliased as strip if -b used and basename set

parse

```
parse(ir, cppfile, src, primary_file_only, base_path, verbose, debug)
```

sourcefile

```
sourcefile
```

Package Python

Modules

- Python.ASGTranslator
- Python.Python
- Python.SXRGenerator

Module Python.ASGTranslator

class TokenParser

text

```
text
```

lines

```
lines
```

generator

```
generator
```

closers

```
closers
```

openers

```
openers
```

del_ws_prefix

```
del_ws_prefix
```

no_ws_suffix

```
no_ws_suffix
```

__init__

```
__init__(self, text)
```

__iter__

```
__iter__(self)
```

next

```
next(self)
```

goto_line

```
goto_line(self, lineno)
```

rhs

```
rhs(self, lineno)
```

Return a whitespace-normalized expression string from the right-hand side of an assignment at line `lineno`.

note_token

```
note_token(self)
```

function_parameters

```
function_parameters(self, lineno)
```

Return a dictionary mapping parameters to defaults (whitespace-normalized strings).

class ASGTranslator

Translate the Python AST into a Synopsis ASG.

scope

```
scope
```

file

```
file
```

types

```
types
```

attributes

```
attributes
```

any_type

```
any_type
```

docformat

```
docformat
```

documentable

```
documentable
```

name

```
name
```

imports

```
imports
```

Tuple with (module, names) pairs.

__init__

```
__init__(self, package, types, docformat)
```

Create an ASGTranslator. package: enclosing package the generated modules are to be part of.

process_file

```
process_file(self, file)
```

scope_name

```
scope_name(self)
```

default

```
default(self, node, args)
```

default_visit

```
default_visit(self, node, args)
```

visitDiscard

```
visitDiscard(self, node)
```

visitConst

```
visitConst(self, node)
```

visitStmt

```
visitStmt(self, node)
```

visitAssign

```
visitAssign(self, node)
```

visitModule

```
visitModule(self, node)
```

visitImport

```
visitImport(self, node)
```

visitFrom

```
visitFrom(self, node)
```

visitAssName

```
visitAssName(self, node)
```

visitAssTuple

```
visitAssTuple(self, node)
```

visitAssAttr

```
visitAssAttr(self, node)
```

visitGetattr

```
visitGetattr(self, node, suffix)
```

visitName

```
visitName(self, node, suffix = None)
```

visitFunction

```
visitFunction(self, node)
```

parse_parameter_list

```
parse_parameter_list(self, node)
```

visitClass

```
visitClass(self, node)
```

visitGetattr

```
visitGetattr(self, node, suffix = None)
```

Package Python.Python

__all__

```
__all__
```

class Parser

Python Parser. See http://www.python.org/dev/peps/pep-0258 for additional info.

primary_file_only

```
primary_file_only
```

base_path

```
base_path
```

sxr_prefix

```
sxr_prefix
```

default_docformat

```
default_docformat
```

process

```
process(self, ir, kwds)
```

process_file

```
process_file(self, filename, base_path)
```

Parse an individual python file.

expand_package

```
expand_package(root, verbose = False)
```

Find all modules in a given package.

find_imported

```
find_imported(target, base_path, origin, verbose = False)
```

Lookup imported files, based on current file's location. target: (module, name) pair. base_path: root directory to which to confine the lookup. origin: file name of the module issuing the import.

Module Python.SXRGenerator

class LexerDebugger

lexer

```
lexer
```

__init__

```
__init__(self, lexer)
```

next

```
next(self)
```

class SXRGenerator

debug

```
debug
```

handlers

```
handlers
```

col

```
col
```

lineno

```
lineno
```

parameters

```
parameters
```

scopes

```
scopes
```

__init__

```
__init__(self, debug = False)
```

process_file

```
process_file(self, scope, sourcefile, sxr)
```

handle

```
handle(self, ptree)
```

default_handler

```
default_handler(self, ptree)
```

next_token

```
next_token(self)
```

Return the next visible token. Process tokens that are not part of the parse tree silently.

handle_token

```
handle_token(self, item = None)
```

handle_name_as_xref

```
handle_name_as_xref(self, xref, name, from_ = None, type = None)
```

handle_tokens

```
handle_tokens(self, ptree)
```

handle_end_marker

```
handle_end_marker(self, nodes)
```

handle_newline

```
handle_newline(self, nodes)
```

handle_indent

```
handle_indent(self, indent)
```

handle_dedent

```
handle_dedent(self, dedent)
```

handle_string

```
handle_string(self, content)
```

handle_function

```
handle_function(self, nodes)
```

handle_parameters

```
handle_parameters(self, nodes)
```

handle_class

```
handle_class(self, nodes)
```

handle_name

```
handle_name(self, content)
```

handle_expr_stmt

```
handle_expr_stmt(self, nodes)
```

handle_dotted_name

```
handle_dotted_name(self, dname, rest)
```

handle_op

```
handle_op(self, nodes)
```

handle_power

```
handle_power(self, content)
```

handle_encoding_decl

```
handle_encoding_decl(self, nodes)
```

handle_import_as_names

```
handle_import_as_names(self, nodes)
```

handle_dotted_as_names

```
handle_dotted_as_names(self, nodes)
```

handle_import_from

```
handle_import_from(self, nodes)
```

handle_import_name

```
handle_import_name(self, nodes)
```

handle_import

```
handle_import(self, nodes)
```

handle_decorator

```
handle_decorator(self, nodes)
```

print_token

```
print_token(self, t)
```

print_newline

```
print_newline(self)
```

num_tokens

```
num_tokens(ptree)
```

Count the number of leaf tokens in the given ptree.

escape

```
escape(text)
```

HAVE_ENCODING_DECL

```
HAVE_ENCODING_DECL
```

HAVE_IMPORT_NAME

```
HAVE_IMPORT_NAME
```

HAVE_DECORATOR

```
HAVE_DECORATOR
```

header

```
header
```

trailer

```
trailer
```

The HTML Formatter

Modules

- DirectoryLayout
- Fragment
- Fragments
- Frame
- FrameSet
- HTML
- Markup
- Part
- Parts
- Tags
- View
- Views
- XRefPager

Module DirectoryLayout

class DirectoryLayout

DirectoryLayout.DirectoryLayout

DirectoryLayout defines how the generated html files are organized. The default implementation uses a flat layout with all files being part of a single directory.

_strip

```
_strip(self, filename)
```

init

```
init(self, processor)
```

copy_file

```
copy_file(self, src, dest)
```

Copy src to dest, if dest doesn't exist yet or is outdated.

scope

```
scope(self, scope = None)
```

Return the filename of a scoped name (class or module). The default implementation is to join the names with '-' and append ".html". Additionally, special characters are quoted.

file_index

```
file_index(self, filename)
```

Return the filename for the index of an input file. Default implementation is to join the path with '.', prepend '_file.' and append '.html'

file_source

```
file_source(self, filename)
```

Return the filename for the source of an input file. Default implementation is to join the path with '.', prepend '_source.' and append '.html'

file_details

```
file_details(self, filename)
```

Return the filename for the details of an input file. Default implementation is to join the path with '.', prepend '_file_detail.' and append '.html'

index

```
index(self)
```

Return the name of the main index file. Default is index.html

special

```
special(self, name)
```

Return the name of a special file (tree, etc). Default is _name.html

scoped_special

```
scoped_special(self, name, scope, ext = '.html')
```

Return the name of a special type of scope file. Default is to join the scope with '.' and prepend '.'+name

xref

```
xref(self, page)
```

Return the name of the xref file for the given page

module_tree

```
module_tree(self)
```

Return the name of the module tree index. Default is _modules.html

module_index

```
module_index(self, scope)
```

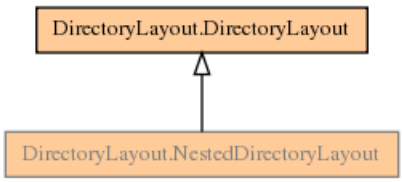
Return the name of the index of the given module. Default is to join the name with '.', prepend '_module' and append '.html'

link

```
link(self, decl)
```

Create a link to the named declaration. This method may have to deal with the directory layout.

class NestedDirectoryLayout



Organizes files in a directory tree.

scope

```
scope(self, scope = None)
```

file_index

```
file_index(self, filename)
```

file_source

```
file_source(self, filename)
```

file_details

```
file_details(self, filename)
```

special

```
special(self, name)
```

scoped_special

```
scoped_special(self, name, scope, ext = '.html')
```

xref

```
xref(self, page)
```

module_tree

```
module_tree(self)
```

module_index

```
module_index(self, scope)
```

Module Fragment

class Fragment

Generates HTML fragment for a declaration. Multiple strategies are combined to generate the output for a single declaration, allowing the user to customise the output by choosing a set of strategies. This follows the Strategy design pattern. The key concept of this class is the format* methods. Any class derived from Strategy that overrides one of the format methods will have that method called by the Summary and Detail parts when they visit that ASG type. Summary and Detail maintain a list of Strategies, and a list for each ASG type. For example, when Strategy.Summary visits a Function object, it calls the formatFunction method on all Strategys registed with Summary that implemented that method. Each of these format methods returns a string, which may contain a TD tag to create a new column. An important point to note is that only Strategies which override a particular format method are called - if that format method is not overridden then it is not called for that declaration type.

register

```
register(self, part)
```

Store part as self.part. The part is either a Summary or Detail, and is used for things like reference() and label() calls. Local references to the part's reference and label methods are stored in self for more efficient use of them.

format_modifiers

```
format_modifiers(self, modifiers)
```

Returns a HTML string from the given list of string modifiers. The modifiers are enclosed in 'keyword' spans.

format_declaration

```
format_declaration(self, decl)
```

format_macro

```
format_macro(self, decl)
```

format_forward

```
format_forward(self, decl)
```

format_group

```
format_group(self, decl)
```

format_scope

```
format_scope(self, decl)
```

format_module

```
format_module(self, decl)
```

format_meta_module

```
format_meta_module(self, decl)
```

format_class

```
format_class(self, decl)
```

format_class_template

```
format_class_template(self, decl)
```

format_typedef

```
format_typedef(self, decl)
```

format_enum

```
format_enum(self, decl)
```

format_variable

```
format_variable(self, decl)
```

format_const

```
format_const(self, decl)
```

format_function

```
format_function(self, decl)
```

format_function_template

```
format_function_template(self, decl)
```

format_operation

```
format_operation(self, decl)
```

format_operation_template

```
format_operation_template(self, decl)
```

Package Fragments

Modules

- Fragments.ClassHierarchyGraph
- Fragments.ClassHierarchySimple
- Fragments.DeclarationCommenter
- Fragments.DeclarationFormatter
- Fragments.Default
- Fragments.DetailCommenter
- Fragments.HeadingFormatter
- Fragments.InheritanceFormatter
- Fragments.SourceLinker
- Fragments.SummaryCommenter
- Fragments.TemplateSpecializations
- Fragments.XRefLinker

Module Fragments.ClassHierarchyGraph

class ClassHierarchyGraph

Fragments.ClassHierarchyGraph.ClassHierarchyGraph

Prints a graphical hierarchy for classes, using the Dot formatter. @see Formatters.Dot

format_class_template

```
format_class_template
```

format_class

```
format_class(self, class_)
```

Module Fragments.ClassHierarchySimple

class ClassHierarchySimple

Fragments.ClassHierarchySimple.ClassHierarchySimple

Prints a simple text hierarchy for classes

format_class_template

```
format_class_template
```

format_inheritance

```
format_inheritance(self, inheritance)
```

format_class

```
format_class(self, class_)
```

Module Fragments.DeclarationCommenter

class DeclarationCommenter

Fragments.DeclarationCommenter.DeclarationCommenter

Add annotation details to all declarations.

format_declaration

```
format_declaration(self, decl)
```

Module Fragments.DeclarationFormatter

class DeclarationFormatter

Fragments.DeclarationFormatter.DeclarationFormatter

Base class for SummaryFormatter and DetailFormatter. The two classes SummaryFormatter and DetailFormatter are actually very similar in operation, and so most of their methods are defined here. Both of them print out the definition of the declarations, including type, parameters, etc. Some things such as exception specifications are only printed out in the detailed version.

register

```
register(self, part)
```

format_parameters

```
format_parameters(self, parameters)
```

Returns formatted string for the given parameter list.

format_declaration

```
format_declaration(self, decl)
```

The default is to return no type and just the declarations name for the name.

format_macro

```
format_macro(self, decl)
```

format_forward

```
format_forward(self, decl)
```

format_group

```
format_group(self, decl)
```

format_scope

```
format_scope(self, decl)
```

Scopes have their own views, so return a reference to it.

format_module

```
format_module(self, decl)
```

format_meta_module

```
format_meta_module(self, decl)
```

format_class

```
format_class(self, decl)
```

format_class_template

```
format_class_template(self, decl)
```

format_typedef

```
format_typedef(self, decl)
```

(typedef type, typedef name)

format_enumerator

```
format_enumerator(self, decl)
```

This is only called by formatEnum

format_enum

```
format_enum(self, decl)
```

(enum name, list of enumerator names)

format_variable

```
format_variable(self, decl)
```

format_const

```
format_const(self, decl)
```

(const type, const name = const value)

format_function

```
format_function(self, decl)
```

(return type, func + params + exceptions)

format_function_template

```
format_function_template(self, decl)
```

(return type, func + params + exceptions)

format_operation

```
format_operation(self, decl)
```

format_operation_template

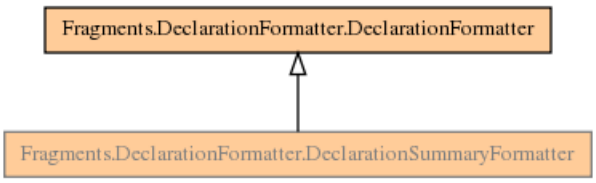
```
format_operation_template(self, decl)
```

format_parameter

```
format_parameter(self, parameter)
```

Returns one string for the given parameter

class DeclarationSummaryFormatter



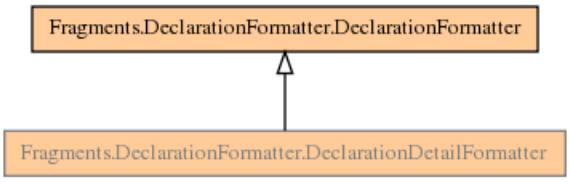
Derives from BaseStrategy to provide summary-specific methods. Currently the only one is format_exceptions

format_exceptions

```
format_exceptions(self, oper)
```

Returns a reference to the detail if there are any exceptions.

class DeclarationDetailFormatter



Provide detail-specific Declaration formatting.

format_exceptions

```
format_exceptions(self, oper)
```

Prints out the full exception spec

format_enum

```
format_enum(self, enum)
```

format_enumerator

```
format_enumerator(self, enumerator)
```

Module Fragments.Default

class Default

A base ASG strategy that calls format_declaration for all types

format_macro

```
format_macro(self, decl)
```

format_forward

```
format_forward(self, decl)
```

format_group

```
format_group(self, decl)
```

format_scope

```
format_scope(self, decl)
```

format_module

```
format_module(self, decl)
```

format_meta_module

```
format_meta_module(self, decl)
```

format_class

```
format_class(self, decl)
```

format_class_template

```
format_class_template(self, decl)
```

format_typedef

```
format_typedef(self, decl)
```

format_enum

```
format_enum(self, decl)
```

format_variable

```
format_variable(self, decl)
```

format_const

```
format_const(self, decl)
```

format_function

```
format_function(self, decl)
```

format_function_template

```
format_function_template(self, decl)
```

format_operation

```
format_operation(self, decl)
```

format_operation_template

```
format_operation_template(self, decl)
```

Module Fragments.DetailCommenter

class DetailCommenter

Fragments.DetailCommenter.DetailCommenter

Add annotation details to all declarations.

format_declaration

```
format_declaration(self, decl)
```

Module Fragments.HeadingFormatter

class HeadingFormatter

Fragments.HeadingFormatter.HeadingFormatter

Formats the top of a view - it is passed only the Declaration that the view is for (a Module or Class).

register

```
register(self, part)
```

format_name

```
format_name(self, qname)
```

Formats a qualified name such that each name component becomes a link to the respective scope.

format_name_in_module

```
format_name_in_module(self, qname)
```

Formats a reference to each parent scope, starting at the first non-module scope.

format_module_of_name

```
format_module_of_name(self, qname)
```

Formats a reference to each parent scope and this one.

format_module

```
format_module(self, module)
```

Formats the module by linking to each parent scope in the name.

format_meta_module

```
format_meta_module(self, module)
```

Calls format_module.

format_class

```
format_class(self, class_)
```

Formats the class by linking to each parent scope in the name.

format_class_template

```
format_class_template(self, class_)
```

Formats the class template by linking to each parent scope in the name.

format_forward

```
format_forward(self, forward)
```

Formats the forward declaration if it is a template declaration.

format_parameter

```
format_parameter(self, parameter)
```

Returns one string for the given parameter

Module Fragments.InheritanceFormatter

class InheritanceFormatter

Prints just the name of each declaration, with a link to its doc

format_declaration

```
format_declaration(self, decl, label = None)
```

format_function

```
format_function(self, decl)
```

format_operation

```
format_operation(self, decl)
```

Module Fragments.SourceLinker

_icons

```
_icons
```

class SourceLinker

Fragments.SourceLinker.SourceLinker

Adds a link to the decl on the file view to all declarations

register

```
register(self, part)
```

format_declaration

```
format_declaration(self, decl)
```

Module Fragments.SummaryCommenter

class SummaryCommenter

Fragments.SummaryCommenter.SummaryCommenter

Adds summary annotations to all declarations.

format_declaration

```
format_declaration(self, decl)
```

Module Fragments.TemplateSpecializations

class TemplateSpecializations

Fragments.TemplateSpecializations.TemplateSpecializations

Cross-link primary templates with their specializations.

format_forward

```
format_forward(self, forward)
```

format_class

```
format_class(self, class_)
```

format_class_template

```
format_class_template(self, template_)
```

Module Fragments.XRefLinker

class XRefLinker

Adds an xref link to all declarations

register

```
register(self, part)
```

format_declaration

```
format_declaration(self, decl)
```

Module Frame

class Frame

A Frame is a mediator for views that get displayed in it (as well as other frames. It supports the creation of links across views.

processor

```
processor
```

views

```
views
```

noframes

```
noframes
```

__init__

```
__init__(self, processor, views, noframes = False)
```

process

```
process(self)
```

navigation_bar

```
navigation_bar(self, view)
```

Generates a navigation bar for the given view.

Module FrameSet

class FrameSet

A class that creates an index with frames

process

```
process(self, output, filename, title, index, detail, content)
```

Creates a frames index file.

Package HTML

class DocCache

_process

```
_process(self, decl, view)
```

Return the documentation for the given declaration.

_processor

```
_processor
```

_markup_formatters

```
_markup_formatters
```

_doc_cache

```
_doc_cache
```

__init__

```
__init__(self, processor, markup_formatters)
```

doc

```
doc(self, decl, view)
```

summary

```
summary(self, decl, view)
```

details

```
details(self, decl, view)
```

class Formatter

title

```
title
```

stylesheet

```
stylesheet
```

directory_layout

```
directory_layout
```

toc_in

```
toc_in
```

toc_out

```
toc_out
```

sxr_prefix

```
sxr_prefix
```

index

```
index
```

detail

```
detail
```

content

```
content
```

markup_formatters

```
markup_formatters
```

graph_color

```
graph_color
```

process

```
process(self, ir, kwds)
```

has_view

```
has_view(self, name)
```

test whether the given view is part of the views list.

register_filename

```
register_filename(self, filename, view, scope)
```

Registers a file for later production. The first view to register the filename gets to keep it.

filename_info

```
filename_info(self, filename)
```

Returns information about a registered file, as a (view,scope) pair. Will return None if the filename isn't registered.

Package Markup

Modules

- Markup.Javadoc
- Markup.Markup
- Markup.RST

Module Markup.Javadoc

class Javadoc

A formatter that formats comments similar to Javadoc. See `Javadoc Spec`_ for info. .. _Javadoc Spec: http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/javadoc.html

class Block

tag

```
tag
```

arg

```
arg
```

body

```
body
```

__init__

```
__init__(self, tag, arg, body)
```

summary

```
summary
```

block_tag

```
block_tag
```

inline_tag

```
inline_tag
```

inline_tag_split

```
inline_tag_split
```

xref

```
xref
```

tag_name

```
tag_name
```

arg_tags

```
arg_tags
```

__init__

```
__init__(self)
```

Create regex objects for regexps

split

```
split(self, doc)
```

Split a javadoc comment into description and blocks.

extract_summary

```
extract_summary(self, description)
```

Generate a summary from the given description.

format

```
format(self, decl, view)
```

Format using javadoc markup.

format_description

```
format_description(self, text, view, decl)
```

format_inlines

```
format_inlines(self, view, decl, text)
```

Formats inline tags in the text.

format_params

```
format_params(self, blocks, view, decl)
```

Formats a list of (param, description) tags

format_throws

```
format_throws(self, blocks, view, decl)
```

format_tag

```
format_tag(self, tag, blocks, view, decl)
```

format_inline_tag

```
format_inline_tag(self, tag, content, view, decl)
```

Package Markup.Markup

class Struct

summary

```
summary
```

details

```
details
```

__init__

```
__init__(self, summary = '', details = '')
```

has_details

```
has_details(self)
```

class Formatter

Interface class that takes a 'doc' annotation and formats its text. Markup-specific subclasses should provide appropriate format methods.

_lookup_symbol_in

```
_lookup_symbol_in(self, symbol, scope)
```

_find_method_entry

```
_find_method_entry(self, name, scope)
```

init

```
init(self, processor)
```

format

```
format(self, decl, view)
```

Format the declaration's documentation. @param view the View to use for references and determining the correct relative filename. @param decl the declaration @returns Struct containing summary / details pair.

lookup_symbol

```
lookup_symbol(self, symbol, scope)
```

Given a symbol and a scope, returns an URL. Various methods are tried to resolve the symbol. First the parameters are taken off, then we try to split the symbol using '.' or '::'. The params are added back, and then we try to match this scoped name against the current scope. If that fails, then we recursively try enclosing scopes.

Module Markup.RST

class SummaryExtractor

A SummaryExtractor creates a document containing the first sentence of a source document.

summary

```
summary
```

__init__

```
__init__(self, document)
```

visit_paragraph

```
visit_paragraph(self, node)
```

Copy the paragraph but only keep the first sentence.

unknown_visit

```
unknown_visit(self, node)
```

Ignore all unknown nodes

class RST

Format summary and detail documentation according to restructured text markup.

roles

```
roles
```

format

```
format(self, decl, view)
```

span

```
span(name, rawtext, text, lineno, inliner, options = {}, content = [])
```

Maps a rols to a ....

Module Part

class Part

Base class for formatting a Part of a Scope View. This class contains functionality for modularly formatting an ASG node and its children for display. It is typically used to construct Heading, Summary and Detail formatters. Strategy objects are added according to configuration, and this base class then checks which format methods each strategy implements. For each ASG declaration visited, the Part asks all Strategies which implement the appropriate format method to generate output for that declaration. The final writing of the formatted html is delegated to the write_section_start, write_section_end, and write_section_item methods, which must be implemented in a subclass.

fragments

```
fragments
```

register

```
register(self, view)
```

view

```
view(self)
```

filename

```
filename(self)
```

os

```
os(self)
```

scope

```
scope(self)
```

write

```
write(self, text)
```

type_ref

```
type_ref(self)
```

type_label

```
type_label(self)
```

declarator

```
declarator(self)
```

parameter

```
parameter(self)
```

reference

```
reference(self, name, label = None, keys)
```

Returns a reference to the given name. The name is a scoped name, and the optional label is an alternative name to use as the link text. The name is looked up in the TOC so the link may not be local. The optional keys are appended as attributes to the A tag.

label

```
label(self, name, label = None)
```

Create a label for the given name. The label is an anchor so it can be referenced by other links. The name of the label is derived by looking up the name in the TOC and using the link in the TOC entry. The optional

label is an alternative name to use as the displayed name. If the name is not found in the TOC then the name is not anchored and just label is returned (or name if no label is given).

format_declaration

```
format_declaration(self, decl, method)
```

Format decl using named method of each fragment. Each fragment returns two strings - type and name. All the types are joined and all the names are joined separately. The consolidated type and name strings are then passed to write_section_item.

process

```
process(self, decl)
```

Formats the given decl, creating the output for this Part of the view. This method is implemented in various subclasses in different ways, for example Summary and Detail iterate through the children of 'decl' section by section, whereas Heading only formats decl itself.

visit_declaration

```
visit_declaration(self, decl)
```

visit_macro

```
visit_macro(self, decl)
```

visit_forward

```
visit_forward(self, decl)
```

visit_group

```
visit_group(self, decl)
```

visit_scope

```
visit_scope(self, decl)
```

visit_module

```
visit_module(self, decl)
```

visit_meta_module

```
visit_meta_module(self, decl)
```

visit_class

```
visit_class(self, decl)
```

visit_class_template

```
visit_class_template(self, decl)
```

visit_typedef

```
visit_typedef(self, decl)
```

visit_enum

```
visit_enum(self, decl)
```

visit_variable

```
visit_variable(self, decl)
```

visit_const

```
visit_const(self, decl)
```

visit_function

```
visit_function(self, decl)
```

visit_function_template

```
visit_function_template(self, decl)
```

visit_operation

```
visit_operation(self, decl)
```

visit_operation_template

```
visit_operation_template(self, decl)
```

format_type

```
format_type(self, typeObj, id_holder = None)
```

Returns a reference string for the given type object

visit_builtin_type_id

```
visit_builtin_type_id(self, type)
```

Sets the label to be a reference to the type's name

visit_unknown_type_id

```
visit_unknown_type_id(self, type)
```

Sets the label to be a reference to the type's link

visit_declared_type_id

```
visit_declared_type_id(self, type)
```

Sets the label to be a reference to the type's name

visit_dependent_type_id

```
visit_dependent_type_id(self, type)
```

Sets the label to be the type's name (which has no proper scope)

visit_modifier_type_id

```
visit_modifier_type_id(self, type)
```

Adds modifiers to the formatted label of the modifier's alias

visit_parametrized_type_id

```
visit_parametrized_type_id(self, type)
```

Adds the parameters to the template name in angle brackets

visit_template_id

```
visit_template_id(self, type)
```

Labs the template with the parameters

visit_function_type_id

```
visit_function_type_id(self, type)
```

Labels the function type with return type, name and parameters

write_start

```
write_start(self)
```

Abstract method to start the output, eg table headings

write_section_start

```
write_section_start(self, heading)
```

Abstract method to start a section of declaration types

write_section_end

```
write_section_end(self, heading)
```

Abstract method to end a section of declaration types

write_section_item

```
write_section_item(self, text)
```

Abstract method to write the output of one formatted declaration

write_end

```
write_end(self)
```

Abstract method to end the output, eg close the table

Package Parts

Modules

- Parts.Body
- Parts.Detail
- Parts.Heading
- Parts.Inheritance
- Parts.Summary

Module Parts.Body

class Body

fragments

```
fragments
```

write_section_start

```
write_section_start(self, heading)
```

Start a 'detail' section and write an appropriate heading.

write_section_end

```
write_section_end(self, heading)
```

Close the section.

write_section_item

```
write_section_item(self, text)
```

Add an item.

process

```
process(self, decl)
```

Print out the details for the children of the given decl

Module Parts.Detail

class Detail

fragments

```
fragments
```

write_section_start

```
write_section_start(self, heading)
```

Start a 'detail' section and write an appropriate heading.

write_section_end

```
write_section_end(self, heading)
```

Close the section.

write_section_item

```
write_section_item(self, text)
```

Add an item.

process

```
process(self, decl)
```

Print out the details for the children of the given decl

Module Parts.Heading

class Heading

Heading view part. Displays a header for the view -- its strategies are only passed the object that the view is for; ie a Class or Module

fragments

```
fragments
```

write_section_item

```
write_section_item(self, text)
```

Writes text and follows with a horizontal rule

process

```
process(self, decl)
```

Process this Part by formatting only the given decl

Module Parts.Inheritance

class Inheritance

_process_class

```
_process_class(self, class_, names)
```

Prints info for the given class, and calls _process_superclasses after

_process_superclasses

```
_process_superclasses(self, class_, names)
```

Iterates through the superclasses of clas and calls _process_clas for each

fragments

```
fragments
```

register

```
register(self, view)
```

process

```
process(self, decl)
```

Walk the hierarchy to find inherited members to print.

write_section_start

```
write_section_start(self, heading)
```

Creates a table with one row. The row has a td of class 'heading' containing the heading string

write_section_item

```
write_section_item(self, text)
```

Adds a table row

write_section_end

```
write_section_end(self, heading)
```

short_name

```
short_name(decl)
```

Module Parts.Summary

class Summary

Formatting summary visitor. This formatter displays a summary for each declaration, with links to the details if there is one. All of this is controlled by the ASGFormatters.

fragments

```
fragments
```

register

```
register(self, view)
```

set_link_detail

```
set_link_detail(self, flag)
```

Sets link_detail flag to given value. @see label()

label

```
label(self, ref, label = None)
```

Override to check link_detail flag. If it's set, returns a reference instead - which will be to the detailed info

write_section_start

```
write_section_start(self, heading)
```

Start a 'summary' section and write an appropriate heading.

write_section_end

```
write_section_end(self, heading)
```

Close the section.

write_section_item

```
write_section_item(self, text)
```

Add an item.

process

```
process(self, scope)
```

Print out the summaries from the given scope

Module Tags

rel

```
rel(origin, target)
```

Find link to target relative to origin URL.

attributes

```
attributes(keys)
```

Convert a name/value dict to a string of attributes. A common HTML attribute is 'class'. Since 'class' is a Python keyword, we accept 'class_' instead, and translate that to 'class'.

element

```
element(_, body = None, keys)
```

Wrap the body in a tag of given type and attributes

href

```
href(ref, label, attrs)
```

img

```
img(attrs)
```

name

```
name(ref, label)
```

span

```
span(body, attrs)
```

div

```
div(body, attrs)
```

para

```
para(body, attrs)
```

desc

```
desc(text)
```

Create a description div for the given text

escape

```
escape(text)
```

quote_as_id

```
quote_as_id(text)
```

replace_spaces

```
replace_spaces(text)
```

Replaces spaces in the given string with   sequences. Does NOT replace spaces inside tags

using_frames

```
using_frames
```

Module View

class Format

Default and base class for formatting a view layout. The Format class basically defines the HTML used at the start and end of the view. The default creates an XHTML compliant header and footer with a proper title, and link to the stylesheet.

init

```
init(self, processor, prefix)
```

view_header

```
view_header(self, os, title, body, headextra, view)
```

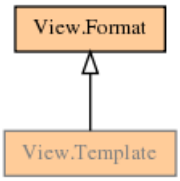
Called to output the view header to the given output stream. @param os a file-like object (use os.write()) @param title the title of this view @param body the body tag, which may contain extra parameters such as onLoad scripts, and may also be empty eg: for the frames index @param headextra extra html to put in the head section, such as scripts

view_footer

```
view_footer(self, os, body)
```

Called to output the view footer to the given output stream. @param os a file-like object (use os.write()) @param body the close body tag, which may be empty eg: for the frames index

class Template



Format subclass that uses a template file to define the HTML header and footer for each view.

template

```
template
```

copy_files

```
copy_files
```

init

```
init(self, processor, prefix)
```

load_file

```
load_file(self)
```

Loads and parses the template file

write

```
write(self, os, text)
```

Writes the text to the output stream, replaceing @PREFIX@ with the prefix for this file

view_header

```
view_header(self, os, title, body, headextra, view)
```

Formats the header using the template file

view_footer

```
view_footer(self, os, body)
```

Formats the footer using the template file

class View

Base class for Views. The base class provides a common interface, and also handles common operations such as opening the file, and delegating the view formatting to a strategy class.

main

```
main
```

_id_counter

```
_id_counter
```

template

```
template
```

__init__

```
__init__(self, kwds)
```

generate_id

```
generate_id(self)
```

Generate an id that is (at least) unique on a particular view, and thus, html document.

register

```
register(self, frame)
```

Registers this View class with its frame.

filename

```
filename(self)
```

Return the filename (currently) associated with the view.

title

```
title(self)
```

Return the title (currently) associated with the view.

root

```
root(self)
```

Return a pair of (url, label) to link to the entry point of this view.

write_navigation_bar

```
write_navigation_bar(self)
```

Generate a navigation bar for this view.

os

```
os(self)
```

Returns the output stream opened with start_file

write

```
write(self, str)
```

Writes the given string to the currently opened file

register_filenames

```
register_filenames(self)
```

Register filenames for each file this View will generate.

toc

```
toc(self)
```

Retrieves the TOC for this view. This method assumes that the view generates info for the the whole ASG, which could be the Scope, the Source (source code) or the XRef (cross reference info). The default implementation returns None.

process

```
process(self)
```

Process the ASG, creating view-specific html pages.

open_file

```
open_file(self)
```

Returns a new output stream. This template method is for internal use only, but may be overriden in derived classes. The default joins output dir and self.filename()

close_file

```
close_file(self)
```

Closes the internal output stream. This template method is for internal use only, but may be overriden in derived classes.

start_file

```
start_file(self, body = '', headextra = '')
```

Start a new file with given filename, title and body. This method opens a file for writing, and writes the html header crap at the top. You must specify a title, which is prepended with the project name. The body argument is optional, and it is preferred to use stylesheets for that sort of stuff. You may want to put an onLoad handler in it though in which case that's the place to do it. The opened file is stored and can be accessed using the os() method.

end_file

```
end_file(self, body = '</body>')
```

Close the file using given close body tag. The default is just a close body tag, but if you specify " then nothing will be written (useful for a frames view)

reference

```
reference(self, name, scope, label = None, keys)
```

Returns a reference to the given name. The name is a scoped name, and the optional label is an alternative name to use as the link text. The name is looked up in the TOC so the link may not be local. The optional keys are appended as attributes to the A tag.

Package Views

Modules

- Views.Directory
- Views.FileDetails
- Views.FileIndex
- Views.FileListing
- Views.FileTree
- Views.InheritanceGraph
- Views.InheritanceTree
- Views.ModuleIndex
- Views.ModuleListing
- Views.ModuleTree
- Views.NameIndex
- Views.RawFile
- Views.Scope
- Views.Source
- Views.Tree
- Views.XRef

Module Views.Directory

class Directory

A view that lists the content of a directory.

src_dir

```
src_dir
```

base_path

```
base_path
```

exclude

```
exclude
```

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

filename_for_dir

```
filename_for_dir(self, dir)
```

Returns the output filename for the given input directory.

register

```
register(self, frame)
```

register_filenames

```
register_filenames(self)
```

process

```
process(self)
```

process_dir

```
process_dir(self, path)
```

end_file

```
end_file(self)
```

Overrides end_file to provide synopsis logo

compile_glob

```
compile_glob(globstr)
```

Returns a compiled regular expression for the given glob string. A glob string is something like "*.?pp" which gets translated into "^.*\..pp$".

Module Views.FileDetails

class FileDetails

A view that creates an index of files, and an index for each file. First the index of files is created, intended for the top-left frame. Second a view is created for each file, listing the major declarations for that file, eg: classes, global functions, namespaces, etc.

register

```
register(self, frame)
```

filename

```
filename(self)
```

since FileTree generates a whole file hierarchy, this method returns the current filename, which may change over the lifetime of this object

title

```
title(self)
```

since FileTree generates a while file hierarchy, this method returns the current title, which may change over the lifetime of this object

register_filenames

```
register_filenames(self)
```

Registers a view for each file indexed.

process

```
process(self)
```

Creates a view for each known source file.

process_file

```
process_file(self, filename, file)
```

Creates a view for the given file. The view is just an index, containing a list of declarations.

Module Views.FileIndex

class FileIndex

A view that creates an index of files, and an index for each file. First the index of files is created, intended for the top-left frame. Second a view is created for each file, listing the major declarations for that file, eg: classes, global functions, namespaces, etc.

register

```
register(self, frame)
```

filename

```
filename(self)
```

since FileTree generates a whole file hierarchy, this method returns the current filename, which may change over the lifetime of this object

title

```
title(self)
```

since FileTree generates a while file hierarchy, this method returns the current title, which may change over the lifetime of this object

register_filenames

```
register_filenames(self)
```

Registers a view for each file indexed.

process

```
process(self)
```

Creates a view for each known file.

process_file

```
process_file(self, filename, file)
```

Creates a view for the given file. The view is just an index, containing a list of declarations.

Module Views.FileListing

class FileListing

Views.FileListing.FileListing

A view that creates an index of files, and an index for each file. First the index of files is created, intended for the top-left frame. Second a view is created for each file, listing the major declarations for that file, eg: classes, global functions, namespaces, etc.

_node_sorter

```
_node_sorter(self, a, b)
```

Compares file nodes a and b depending on whether they are leaves or not

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

register_filenames

```
register_filenames(self)
```

Registers a view for each file indexed.

process

```
process(self)
```

Creates the listing using the recursive process_file_tree_node method

process_file_tree_node

```
process_file_tree_node(self, node)
```

Creates a portion of the tree for the given file node. This method assumes that the file is already in progress, and just appends to it. This method is recursive, calling itself for each child of node (file or directory).

Module Views.FileTree

class FileTree

Views.FileTree.FileTree

Create a javascript-enabled file tree.

link_to_views

```
link_to_views
```

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

process

```
process(self)
```

process_node

```
process_node(self, node)
```

Module Views.InheritanceGraph

class DeclarationFinder

__call__

```
__call__(self, name)
```

types

```
types
```

verbose

```
verbose
```

__init__

```
__init__(self, types, verbose)
```

visit_buitin_type_id

```
visit_buitin_type_id(self, type)
```

visit_unknown_type_id

```
visit_unknown_type_id(self, type)
```

visit_declared_type_id

```
visit_declared_type_id(self, type)
```

visit_modifier_type_id

```
visit_modifier_type_id(self, type)
```

visit_array_type_id

```
visit_array_type_id(self, type)
```

visit_template_id

```
visit_template_id(self, type)
```

visit_parametrized_type_id

```
visit_parametrized_type_id(self, type)
```

visit_function_type_id

```
visit_function_type_id(self, type)
```

class InheritanceGraph

min_size

```
min_size
```

min_group_size

```
min_group_size
```

direction

```
direction
```

register

```
register(self, frame)
```

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

consolidate

```
consolidate(self, graphs)
```

Consolidates small graphs into larger ones

process

```
process(self)
```

Creates a file with the inheritance graph

find_common_name

```
find_common_name(graph)
```

Module Views.InheritanceTree

class InheritanceTree

Views.InheritanceTree.InheritanceTree

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

process

```
process(self)
```

process_inheritance

```
process_inheritance(self, name, rel_name)
```

Module Views.ModuleIndex

class ModuleIndex

A module for indexing ASG.Modules. Each module gets its own view with a list of nested scope declarations with documentation. It is intended to go in the left frame...

register

```
register(self, frame)
```

filename

```
filename(self)
```

title

```
title(self)
```

process

```
process(self)
```

make_view_heading

```
make_view_heading(self, module)
```

Creates a HTML fragment which becomes the name at the top of the index view. This may be overridden, but the default is (now) to make a linked fully scoped name, where each scope has a link to the relevant index.

process_module_index

```
process_module_index(self, module)
```

Index one module

Module Views.ModuleListing

class ModuleListing

Create an index of all modules.

_link_href

```
_link_href(self, module)
```

Returns the link to the given declaration

short_title

```
short_title
```

child_types

```
child_types
```

register

```
register(self, frame)
```

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

process

```
process(self)
```

Create a view with an index of all modules.

get_children

```
get_children(self, decl)
```

Returns the children of the given declaration

index_module

```
index_module(self, module, rel_scope)
```

Write a link for this module and recursively visit child modules.

Module Views.ModuleTree

class ModuleTree

Create a javascript-enabled module tree.

_link_href

```
_link_href(self, module)
```

register

```
register(self, frame)
```

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

process

```
process(self)
```

get_children

```
get_children(self, decl)
```

index_module

```
index_module(self, module, qname)
```

Write a link for this module and recursively visit child modules.

Module Views.NameIndex

class NameIndex

Creates an index of all names on one view in alphabetical order.

_process_item

```
_process_item(self, type)
```

Process the given name for output

columns

```
columns
```

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

process

```
process(self)
```

make_dictionary

```
make_dictionary(self)
```

Returns a dictionary of items. The keys of the dictionary are the headings - the first letter of the name. The values are each a sorted list of items with that first letter.

end_file

```
end_file(self)
```

Overrides end_file to provide synopsis logo

Module Views.RawFile

class RawFile

A module for creating a view for each file with hyperlinked source

_get_files

```
_get_files(self)
```

Returns a list of (path, output_filename) for each file.

src_dir

```
src_dir
```

base_path

```
base_path
```

exclude

```
exclude
```

register

```
register(self, frame)
```

filename

```
filename(self)
```

title

```
title(self)
```

process

```
process(self)
```

Creates a view for every file.

register_filenames

```
register_filenames(self)
```

process_file

```
process_file(self, original, filename)
```

Creates a view for the given filename.

Module Views.Scope

class Scope

A module for creating a view for each Scope with summaries and details. This module is highly modular, using the classes from ASGFormatter to do the actual formatting. The classes to use may be controlled via the config script, resulting in a very configurable output. @see ASGFormatter The ASGFormatter module @see Config.Formatters.HTML.ScopeViews Config for ScopeViews

parts

```
parts
```

register

```
register(self, frame)
```

toc

```
toc(self)
```

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

scope

```
scope(self)
```

return the current scope processed by this object

process

```
process(self)
```

Creates a view for every Scope.

register_filenames

```
register_filenames(self)
```

Registers a view for every Scope.

process_scope

```
process_scope(self, scope)
```

Creates a view for the given scope

end_file

```
end_file(self)
```

Overrides end_file to provide synopsis logo

Module Views.Source

class SXRTranslator

Read in an sxr file, resolve references, and write it out as part of a Source view.

__init__

```
__init__(self, filename, language, debug)
```

link

```
link(self, linker)
```

translate

```
translate(self, writer)
```

class Source

A module for creating a view for each file with hyperlinked source

external_url

```
external_url
```

register

```
register(self, frame)
```

filename

```
filename(self)
```

title

```
title(self)
```

process

```
process(self)
```

Creates a view for every file

register_filenames

```
register_filenames(self)
```

Registers a view for every source file

process_node

```
process_node(self, file)
```

Creates a view for the given file

lookup_symbol

```
lookup_symbol(self, name)
```

external_ref

```
external_ref(self, name)
```

end_file

```
end_file(self)
```

Overrides end_file to provide synopsis logo

Module Views.Tree

class Tree

View that makes Javascript trees. The trees have expanding and collapsing nodes. call js_init() with the button images and default open/close policy during process

register

```
register(self, frame)
```

get_id

```
get_id(self)
```

write_leaf

```
write_leaf(self, text)
```

Write a leaf node to the output at the current tree level.

write_node_start

```
write_node_start(self, text)
```

Write a non-leaf node to the output at the current tree level, and start a new level.

write_node_end

```
write_node_end(self)
```

Finish a non-leaf node, and close the current tree level.

end_tree

```
end_tree(self)
```

Writes the end of the tree.

Module Views.XRef

class XRef

A module for creating views full of xref infos

link_to_scope

```
link_to_scope
```

register

```
register(self, frame)
```

toc

```
toc(self, start)
```

filename

```
filename(self)
```

title

```
title(self)
```

process

```
process(self)
```

register_filenames

```
register_filenames(self)
```

Registers each view

process_link

```
process_link(self, file, line, scope)
```

Outputs the info for one link

describe_declaration

```
describe_declaration(self, decl)
```

Returns a description of the declaration. Detects constructors and destructors

process_name

```
process_name(self, name)
```

Outputs the info for a given name

end_file

```
end_file(self)
```

Overrides end_file to provide synopsis logo

Module XRefPager

class XRefPager

Generates pages of cross-references.

page_map

```
page_map
```

page_info

```
page_info
```

__init__

```
__init__(self, ir)
```

get

```
get(self, name)
```

Returns the number of the page that the xref info for the given name is on, or None if not found.

pages

```
pages(self)
```

Returns a list of pages, each consisting of a list of names on that page. This method is intended to be used by whatever generates the files...

The DocBook Formatter

Packages

- DocBook
- Markup
- Syntax

Package DocBook

_summary_syntax

```
_summary_syntax
```

_detail_syntax

```
_detail_syntax
```

class Linker

Helper class to be used to resolve references from doc-strings to declarations.

link

```
link(self, decl)
```

class _BaseClasses

classes

```
classes
```

classes_once

```
classes_once
```

__init__

```
__init__(self)
```

visit_declared_type_id

```
visit_declared_type_id(self, declared)
```

visit_class

```
visit_class(self, class_)
```

visit_inheritance

```
visit_inheritance(self, inheritance)
```

class ModuleLister

Maps a module-tree to a (flat) list of modules.

modules

```
modules
```

__init__

```
__init__(self)
```

visit_module

```
visit_module(self, module)
```

class InheritanceFormatter

base_dir

```
base_dir
```

bgcolor

```
bgcolor
```

__init__

```
__init__(self, base_dir, bgcolor)
```

format_class

```
format_class(self, class_, format)
```

class FormatterBase

processor

```
processor
```

output

```
output
```

base_dir

```
base_dir
```

nested_modules

```
nested_modules
```

secondary_index

```
secondary_index
```

inheritance_graphs

```
inheritance_graphs
```

graph_color

```
graph_color
```

__scope

```
__scope
```

__scopestack

```
__scopestack
```

__indent

```
__indent
```

__elements

```
__elements
```

__init__

```
__init__(self, processor, output, base_dir, nested_modules, \
secondary_index, inheritance_graphs, graph_color)
```

scope

```
scope(self)
```

push_scope

```
push_scope(self, newscope)
```

pop_scope

```
pop_scope(self)
```

write

```
write(self, text)
```

Write some text to the output stream, replacing 's with 's and indents.

start_element

```
start_element(self, type, params)
```

Write the start of an element, ending with a newline

end_element

```
end_element(self)
```

Write the end of an element, starting with a newline

write_element

```
write_element(self, element, body, end = '\n', params)
```

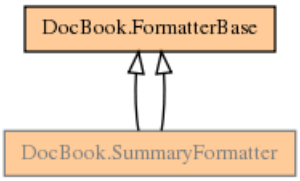
Write a single element on one line (though body may contain newlines)

element

```
element(self, element, body, params)
```

Return but do not write the text for an element on one line

class SummaryFormatter



A SummaryFormatter.

visit_module

```
visit_module
```

visit_class

```
visit_class
```

visit_function

```
visit_function
```

process_doc

```
process_doc(self, decl)
```

visit_declaration

```
visit_declaration(self, declaration)
```

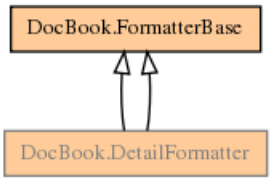
visit_meta_module

```
visit_meta_module(self, meta)
```

visit_enum

```
visit_enum(self, enum)
```

class DetailFormatter



visit_function

```
visit_function
```

visit_builtin_type_id

```
visit_builtin_type_id(self, type)
```

visit_unknown_type_id

```
visit_unknown_type_id(self, type)
```

visit_declared_type_id

```
visit_declared_type_id(self, type)
```

visit_modifier_type_id

```
visit_modifier_type_id(self, type)
```

visit_parametrized_type_id

```
visit_parametrized_type_id(self, type)
```

visit_function_type_id

```
visit_function_type_id(self, type)
```

process_doc

```
process_doc(self, decl)
```

visit_declaration

```
visit_declaration(self, declaration)
```

generate_module_list

```
generate_module_list(self, modules)
```

format_module_or_group

```
format_module_or_group(self, module, title, sort)
```

visit_module

```
visit_module(self, module)
```

visit_group

```
visit_group(self, group)
```

visit_class

```
visit_class(self, class_)
```

visit_inheritance

```
visit_inheritance(self, inheritance)
```

visit_parameter

```
visit_parameter(self, parameter)
```

visit_enum

```
visit_enum(self, enum)
```

class DocCache

_process

```
_process(self, decl)
```

Return the documentation for the given declaration.

_processor

```
_processor
```

_markup_formatters

```
_markup_formatters
```

_doc_cache

```
_doc_cache
```

__init__

```
__init__(self, processor, markup_formatters)
```

summary

```
summary(self, decl)
```

details

```
details(self, decl)
```

class Formatter

Generate a DocBook reference.

markup_formatters

```
markup_formatters
```

title

```
title
```

nested_modules

```
nested_modules
```

generate_summary

```
generate_summary
```

hide_undocumented

```
hide_undocumented
```

inline_inherited_members

```
inline_inherited_members
```

secondary_index_terms

```
secondary_index_terms
```

with_inheritance_graphs

```
with_inheritance_graphs
```

graph_color

```
graph_color
```

process

```
process(self, ir, kwds)
```

escape

```
escape(text)
```

reference

```
reference(name)
```

Generate an id suitable as a 'linkend' / 'id' attribute, i.e. for linking.

Package Markup

Modules

- Markup.Javadoc
- Markup.Markup
- Markup.RST

Module Markup.Javadoc

class Javadoc

A formatter that formats comments similar to Javadoc. See `Javadoc Spec`_ for info. .. _Javadoc Spec: http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/javadoc.html

class Block

tag

```
tag
```

arg

```
arg
```

body

```
body
```

__init__

```
__init__(self, tag, arg, body)
```

summary

```
summary
```

block_tag

```
block_tag
```

inline_tag

```
inline_tag
```

inline_tag_split

```
inline_tag_split
```

xref

```
xref
```

tag_name

```
tag_name
```

arg_tags

```
arg_tags
```

__init__

```
__init__(self)
```

Create regex objects for regexps

split

```
split(self, doc)
```

Split a javadoc comment into description and blocks.

extract_summary

```
extract_summary(self, description)
```

Generate a summary from the given description.

format

```
format(self, decl)
```

Format using javadoc markup.

format_description

```
format_description(self, text, decl)
```

format_inlines

```
format_inlines(self, decl, text)
```

Formats inline tags in the text.

format_params

```
format_params(self, blocks, decl)
```

Formats a list of (param, description) tags

format_throws

```
format_throws(self, blocks, decl)
```

format_variablelist

```
format_variablelist(self, tag, blocks, decl)
```

Generate a variablelist for the given tag. Each matching block is formatted to a varlistentry, with the value of its 'arg' member becoming the term.

format_varlistentry

```
format_varlistentry(self, tag, blocks, decl)
```

Generate a varlistentry for the given tag. The tag value itself becomes the term. If multiple blocks match, format them as an (inlined) simplelist, otherwise as a para.

format_inline_tag

```
format_inline_tag(self, tag, content, decl)
```

attributes

```
attributes(keys)
```

Convert a name/value dict to a string of attributes

element

```
element(_type, body, keys)
```

Wrap the body in a tag of given type and attributes

title

```
title(name)
```

para

```
para(body)
```

listitem

```
listitem(body)
```

term

```
term(body)
```

link

```
link(linkend, label)
```

Package Markup.Markup

class Struct

summary

```
summary
```

details

```
details
```

__init__

```
__init__(self, summary = '', details = '')
```

class Formatter

Interface class that takes a 'doc' annotation and formats its text. Markup-specific subclasses should provide appropriate format methods.

_lookup_symbol_in

```
_lookup_symbol_in(self, symbol, scope)
```

_find_method_entry

```
_find_method_entry(self, name, scope)
```

init

```
init(self, processor)
```

format

```
format(self, decl)
```

Format the declaration's documentation. @param view the View to use for references and determining the correct relative filename. @param decl the declaration @returns Struct containing summary / details pair.

lookup_symbol

```
lookup_symbol(self, symbol, scope)
```

Given a symbol and a scope, returns an URL. Various methods are tried to resolve the symbol. First the parameters are taken off, then we try to split the symbol using '.' or '::'. The params are added back, and then we try to match this scoped name against the current scope. If that fails, then we recursively try enclosing scopes.

escape

```
escape(text)
```

Module Markup.RST

class Writer

settings_spec

```
settings_spec
```

DocBook does its own section numbering

settings_default_overrides

```
settings_default_overrides
```

output

```
output
```

Final translated form of `document`.

translate

```
translate(self)
```

class DocBookTranslator

language

```
language
```

doctype

```
doctype
```

body

```
body
```

section

```
section
```

context

```
context
```

colnames

```
colnames
```

footnotes

```
footnotes
```

footnote_map

```
footnote_map
```

docinfo

```
docinfo
```

title

```
title
```

subtitle

```
subtitle
```

visit_problematic

```
visit_problematic
```

depart_problematic

```
depart_problematic
```

visit_system_message

```
visit_system_message
```

depart_system_message

```
depart_system_message
```

__init__

```
__init__(self, document)
```

astext

```
astext(self)
```

encode

```
encode(self, text)
```

Encode special characters in `text` & return.

encodeattr

```
encodeattr(self, text)
```

Encode attributes characters > 128 as &#XXX;

rearrange_footnotes

```
rearrange_footnotes(self)
```

Replaces ``foonote_reference`` placeholders with ``footnote`` element content as DocBook and reST handle footnotes differently. DocBook defines footnotes inline, whereas they may be anywere in reST. This function replaces the first instance of a ``footnote_reference`` with the ``footnote`` element itself, and later references of the same a footnote with ``footnoteref`` elements.

attval

```
attval(self, text, transtable = string.maketrans('\n\r\t\v\f', '    '))
```

Cleanse, encode, and return attribute value text.

starttag

```
starttag(self, node, tagname, suffix = '\n', infix = '', attributes)
```

Construct and return a start tag given a node (id & class attributes are extracted), tag name, and optional attributes.

emptytag

```
emptytag(self, node, tagname, suffix = '\n', attributes)
```

Construct and return an XML-compatible empty tag.

visit_Text

```
visit_Text(self, node)
```

depart_Text

```
depart_Text(self, node)
```

visit_address

```
visit_address(self, node)
```

depart_address

```
depart_address(self, node)
```

visit_admonition

```
visit_admonition(self, node, name = '')
```

depart_admonition

```
depart_admonition(self, node = None)
```

visit_attention

```
visit_attention(self, node)
```

depart_attention

```
depart_attention(self, node)
```

visit_attribution

```
visit_attribution(self, node)
```

depart_attribution

```
depart_attribution(self, node)
```

visit_author

```
visit_author(self, node)
```

visit_authors

```
visit_authors(self, node)
```

visit_block_quote

```
visit_block_quote(self, node)
```

depart_block_quote

```
depart_block_quote(self, node)
```

visit_bullet_list

```
visit_bullet_list(self, node)
```

depart_bullet_list

```
depart_bullet_list(self, node)
```

visit_caption

```
visit_caption(self, node)
```

depart_caption

```
depart_caption(self, node)
```

visit_caution

```
visit_caution(self, node)
```

depart_caution

```
depart_caution(self, node)
```

visit_citation

```
visit_citation(self, node)
```

depart_citation

```
depart_citation(self, node)
```

visit_citation_reference

```
visit_citation_reference(self, node)
```

depart_citation_reference

```
depart_citation_reference(self, node)
```

visit_classifier

```
visit_classifier(self, node)
```

depart_classifier

```
depart_classifier(self, node)
```

visit_colspec

```
visit_colspec(self, node)
```

depart_colspec

```
depart_colspec(self, node)
```

visit_comment

```
visit_comment(self, node, sub = re.compile('-(?=-)').sub)
```

Escape double-dashes in comment text.

visit_contact

```
visit_contact(self, node)
```

visit_copyright

```
visit_copyright(self, node)
```

visit_danger

```
visit_danger(self, node)
```

depart_danger

```
depart_danger(self, node)
```

visit_date

```
visit_date(self, node)
```

visit_decoration

```
visit_decoration(self, node)
```

depart_decoration

```
depart_decoration(self, node)
```

visit_definition

```
visit_definition(self, node)
```

depart_definition

```
depart_definition(self, node)
```

visit_definition_list

```
visit_definition_list(self, node)
```

depart_definition_list

```
depart_definition_list(self, node)
```

visit_definition_list_item

```
visit_definition_list_item(self, node)
```

depart_definition_list_item

```
depart_definition_list_item(self, node)
```

visit_description

```
visit_description(self, node)
```

depart_description

```
depart_description(self, node)
```

visit_docinfo

```
visit_docinfo(self, node)
```

Collects all docinfo elements for the document. Since reST's bibliography elements don't map very cleanly to DocBook, rather than maintain state and check dependencies within the different visitor fuctions all processing of bibliography elements is dont within this function. .. NOTE:: Skips processing of all child nodes as everything should be collected here.

depart_docinfo

```
depart_docinfo(self, node)
```

visit_doctest_block

```
visit_doctest_block(self, node)
```

depart_doctest_block

```
depart_doctest_block(self, node)
```

visit_document

```
visit_document(self, node)
```

depart_document

```
depart_document(self, node)
```

visit_emphasis

```
visit_emphasis(self, node)
```

depart_emphasis

```
depart_emphasis(self, node)
```

visit_entry

```
visit_entry(self, node)
```

depart_entry

```
depart_entry(self, node)
```

visit_enumerated_list

```
visit_enumerated_list(self, node)
```

depart_enumerated_list

```
depart_enumerated_list(self, node)
```

visit_error

```
visit_error(self, node)
```

depart_error

```
depart_error(self, node)
```

visit_field

```
visit_field(self, node)
```

depart_field

```
depart_field(self, node)
```

visit_field_argument

```
visit_field_argument(self, node)
```

depart_field_argument

```
depart_field_argument(self, node)
```

visit_field_body

```
visit_field_body(self, node)
```

depart_field_body

```
depart_field_body(self, node)
```

visit_field_list

```
visit_field_list(self, node)
```

depart_field_list

```
depart_field_list(self, node)
```

visit_field_name

```
visit_field_name(self, node)
```

depart_field_name

```
depart_field_name(self, node)
```

visit_figure

```
visit_figure(self, node)
```

depart_figure

```
depart_figure(self, node)
```

visit_footer

```
visit_footer(self, node)
```

depart_footer

```
depart_footer(self, node)
```

visit_footnote

```
visit_footnote(self, node)
```

depart_footnote

```
depart_footnote(self, node)
```

visit_footnote_reference

```
visit_footnote_reference(self, node)
```

visit_header

```
visit_header(self, node)
```

depart_header

```
depart_header(self, node)
```

visit_generated

```
visit_generated(self, node)
```

depart_generated

```
depart_generated(self, node)
```

visit_hint

```
visit_hint(self, node)
```

depart_hint

```
depart_hint(self, node)
```

visit_image

```
visit_image(self, node)
```

depart_image

```
depart_image(self, node)
```

visit_important

```
visit_important(self, node)
```

depart_important

```
depart_important(self, node)
```

visit_interpreted

```
visit_interpreted(self, node)
```

depart_interpreted

```
depart_interpreted(self, node)
```

visit_label

```
visit_label(self, node)
```

depart_label

```
depart_label(self, node)
```

visit_legend

```
visit_legend(self, node)
```

depart_legend

```
depart_legend(self, node)
```

visit_line_block

```
visit_line_block(self, node)
```

depart_line_block

```
depart_line_block(self, node)
```

visit_list_item

```
visit_list_item(self, node)
```

depart_list_item

```
depart_list_item(self, node)
```

visit_literal

```
visit_literal(self, node)
```

depart_literal

```
depart_literal(self, node)
```

visit_literal_block

```
visit_literal_block(self, node)
```

depart_literal_block

```
depart_literal_block(self, node)
```

visit_note

```
visit_note(self, node)
```

depart_note

```
depart_note(self, node)
```

visit_option

```
visit_option(self, node)
```

depart_option

```
depart_option(self, node)
```

visit_option_argument

```
visit_option_argument(self, node)
```

depart_option_argument

```
depart_option_argument(self, node)
```

visit_option_group

```
visit_option_group(self, node)
```

depart_option_group

```
depart_option_group(self, node)
```

visit_option_list

```
visit_option_list(self, node)
```

depart_option_list

```
depart_option_list(self, node)
```

visit_option_list_item

```
visit_option_list_item(self, node)
```

depart_option_list_item

```
depart_option_list_item(self, node)
```

visit_option_string

```
visit_option_string(self, node)
```

depart_option_string

```
depart_option_string(self, node)
```

visit_organization

```
visit_organization(self, node)
```

visit_paragraph

```
visit_paragraph(self, node)
```

depart_paragraph

```
depart_paragraph(self, node)
```

visit_raw

```
visit_raw(self, node)
```

visit_reference

```
visit_reference(self, node)
```

depart_reference

```
depart_reference(self, node)
```

visit_revision

```
visit_revision(self, node)
```

visit_row

```
visit_row(self, node)
```

depart_row

```
depart_row(self, node)
```

visit_rubric

```
visit_rubric(self, node)
```

depart_rubric

```
depart_rubric(self, node)
```

visit_section

```
visit_section(self, node)
```

depart_section

```
depart_section(self, node)
```

visit_sidebar

```
visit_sidebar(self, node)
```

depart_sidebar

```
depart_sidebar(self, node)
```

visit_status

```
visit_status(self, node)
```

visit_strong

```
visit_strong(self, node)
```

depart_strong

```
depart_strong(self, node)
```

visit_subscript

```
visit_subscript(self, node)
```

depart_subscript

```
depart_subscript(self, node)
```

visit_substitution_definition

```
visit_substitution_definition(self, node)
```

visit_substitution_reference

```
visit_substitution_reference(self, node)
```

visit_subtitle

```
visit_subtitle(self, node)
```

depart_subtitle

```
depart_subtitle(self, node)
```

visit_superscript

```
visit_superscript(self, node)
```

depart_superscript

```
depart_superscript(self, node)
```

visit_table

```
visit_table(self, node)
```

depart_table

```
depart_table(self, node)
```

visit_target

```
visit_target(self, node)
```

depart_target

```
depart_target(self, node)
```

visit_tbody

```
visit_tbody(self, node)
```

depart_tbody

```
depart_tbody(self, node)
```

visit_term

```
visit_term(self, node)
```

depart_term

```
depart_term(self, node)
```

visit_tgroup

```
visit_tgroup(self, node)
```

depart_tgroup

```
depart_tgroup(self, node)
```

visit_thead

```
visit_thead(self, node)
```

depart_thead

```
depart_thead(self, node)
```

visit_tip

```
visit_tip(self, node)
```

depart_tip

```
depart_tip(self, node)
```

visit_title

```
visit_title(self, node)
```

depart_title

```
depart_title(self, node)
```

visit_title_reference

```
visit_title_reference(self, node)
```

depart_title_reference

```
depart_title_reference(self, node)
```

visit_topic

```
visit_topic(self, node)
```

depart_topic

```
depart_topic(self, node)
```

visit_transition

```
visit_transition(self, node)
```

depart_transition

```
depart_transition(self, node)
```

visit_version

```
visit_version(self, node)
```

visit_warning

```
visit_warning(self, node)
```

depart_warning

```
depart_warning(self, node)
```

unimplemented_visit

```
unimplemented_visit(self, node)
```

class SummaryExtractor

A SummaryExtractor creates a document containing the first sentence of a source document.

summary

```
summary
```

__init__

```
__init__(self, document)
```

visit_paragraph

```
visit_paragraph(self, node)
```

Copy the paragraph but only keep the first sentence.

unknown_visit

```
unknown_visit(self, node)
```

Ignore all unknown nodes

class RST

Format summary and detail documentation according to restructured text markup.

format

```
format(self, decl)
```

Module Syntax

class Syntax

Even though DocBook provides markup for some programming artifacts, it is incomplete, and the XSL stylesheets are buggy, resulting in incorrect syntax. Thus, we use the 'synopsis' element, and attempt to reproduce the original syntax with language-specific subclasses.

output

```
output
```

__init__

```
__init__(self, output)
```

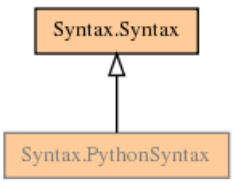
finish

```
finish(self)
```

typeid

```
typeid(self, type)
```

class PythonSyntax



visit_function

```
visit_function(self, node)
```

visit_parameter

```
visit_parameter(self, parameter)
```

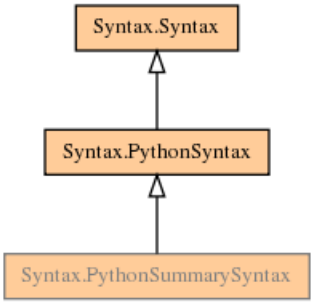
visit_variable

```
visit_variable(self, variable)
```

visit_const

```
visit_const(self, const)
```

class PythonSummarySyntax



Generate DocBook Synopsis for Python declarations.

__init__

```
__init__(self, output)
```

finish

```
finish(self)
```

visit_group

```
visit_group(self, node)
```

visit_module

```
visit_module(self, module)
```

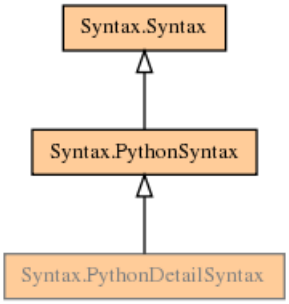
visit_class

```
visit_class(self, class_)
```

visit_inheritance

```
visit_inheritance(self, node)
```

class PythonDetailSyntax



Generate DocBook Synopsis for Python declarations.

__init__

```
__init__(self, output)
```

finish

```
finish(self)
```

visit_group

```
visit_group(self, node)
```

visit_module

```
visit_module(self, module)
```

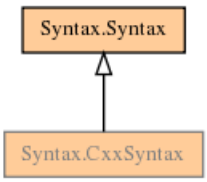
visit_class

```
visit_class(self, class_)
```

visit_inheritance

```
visit_inheritance(self, node)
```

class CxxSyntax



visit_function

```
visit_function(self, node)
```

visit_parameter

```
visit_parameter(self, parameter)
```

visit_typedef

```
visit_typedef(self, node)
```

visit_variable

```
visit_variable(self, variable)
```

visit_const

```
visit_const(self, const)
```

visit_builtin_type_id

```
visit_builtin_type_id(self, type)
```

visit_unknown_type_id

```
visit_unknown_type_id(self, type)
```

visit_declared_type_id

```
visit_declared_type_id(self, type)
```

visit_modifier_type_id

```
visit_modifier_type_id(self, type)
```

visit_array_type_id

```
visit_array_type_id(self, type)
```

visit_template_id

```
visit_template_id(self, type)
```

visit_parametrized_type_id

```
visit_parametrized_type_id(self, type)
```

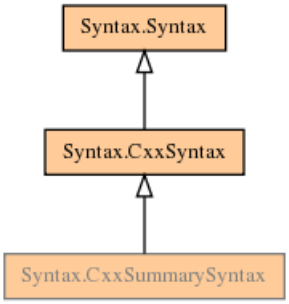
visit_function_type_id

```
visit_function_type_id(self, type)
```

visit_dependent_type_id

```
visit_dependent_type_id(self, type)
```

class CxxSummarySyntax



Generate DocBook Synopsis for C++ declarations.

__init__

```
__init__(self, output)
```

finish

```
finish(self)
```

visit_macro

```
visit_macro(self, macro)
```

visit_forward

```
visit_forward(self, node)
```

visit_group

```
visit_group(self, node)
```

visit_module

```
visit_module(self, module)
```

visit_class

```
visit_class(self, class_)
```

visit_class_template

```
visit_class_template(self, class_)
```

visit_enumerator

```
visit_enumerator(self, node)
```

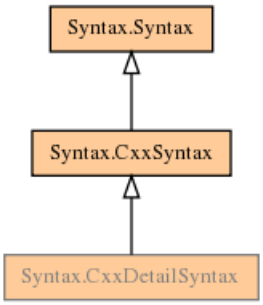
visit_enum

```
visit_enum(self, node)
```

visit_inheritance

```
visit_inheritance(self, node)
```

class CxxDetailSyntax



Generate DocBook Synopsis for C++ declarations.

__init__

```
__init__(self, output)
```

finish

```
finish(self)
```

visit_macro

```
visit_macro(self, macro)
```

visit_forward

```
visit_forward(self, node)
```

visit_group

```
visit_group(self, node)
```

visit_module

```
visit_module(self, module)
```

visit_class

```
visit_class(self, class_)
```

visit_class_template

```
visit_class_template(self, node)
```

visit_enumerator

```
visit_enumerator(self, node)
```

visit_enum

```
visit_enum(self, node)
```

visit_inheritance

```
visit_inheritance(self, node)
```

escape

```
escape(text)
```

The SXR Formatter

class Formatter

This is a facade to the HTML.Formatter. It adds an 'url' parameter and dispatches it to various 'views'.

title

```
title
```

url

```
url
```

sxr_prefix

```
sxr_prefix
```

src_dir

```
src_dir
```

exclude

```
exclude
```

sxr_template

```
sxr_template
```

stylesheet

```
stylesheet
```

process

```
process(self, ir, kwds)
```

class SXRIndex

Top level Index View. This is the starting point for the SXR browser.

sxr_cgi

```
sxr_cgi
```

filename

```
filename(self)
```

title

```
title(self)
```

root

```
root(self)
```

process

```
process(self)
```

Recursively visit each directory below the base path given in the config.

Types

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Functions

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X