PDT's PDB file format definition

Name

PDB definitions --

Definitions

Each item is decribed by a block of lines
First line is always [id] [name] where [id] consists of prefix as described aboved followed by '#' followed by an unsigned (>=0) and unique number (unique inside the item category). The numbering has no other constraints (especially contiguous allocation or start with 0)
Each other line is of the form: attribute value value...
Attributes and values are separated by exactly one space (" ")
Item description blocks are terminated by an empty line
Values which are references to other items are done using IDs
Values are never enclosed in ""
Attributes with boolean value are only listed if true (value == T) i.e. absence means false
String "[...]" after attribute means "0 or more" instances of this attribute line possible

Name

File Header -- Header information included in each pdb file.

<PDB [version]>
lang [language]

Definition

`[version]` which version of PDB file format this file implements, version 3.0 includes statement level information.
`[language]` the programing language that was parsed to generate this file.

Name

Source Files -- This tag records the each pdb files.

so#[id]
[ssys]
sinc so#[id]
[...]
scom co#[comID] [lang] {Location} {Location} [comment]
[...]

Definition

`[id]` Unique number identifying this source file (used to refer to this file elsewhere.)
`[ssys]` ssys tag is printed if this file is a system include, otherwise ommit this line.
`[id]`Id. of the files included by this file.
`[comID]` Unique number identifying this comment in the source file. `[lang]` the programing language that was parsed to generate this file.
`{Location} {Location}` Two location tags, the start of the comment and the end of the comment.
`[comment]` the contents of this comment.

Name

Routines -- This tag records each routine declared in the source file.

ro#[id] [name]
rloc {Location}
rsig ty#[type]
rlink [link]
rkind [kind]
rvirt [virtual]
rgroup gr#[groupId]
racs [access]
rnspace na#[namespaceId]
rroutine [parentRoutineId]
ralias ro#[aliasId]
rimpl ro#[implementsId]
[rstatic]
[rcrvo]
[rinline]
[rexpl]
rtempl te#[templateId]
[rspecl]
[rarginfo]
[rrec]
[riselem]
rstart {Location}
rcall ro#[calleeId] [virtualCall] {Location}
[...]
rret {Location}
[...]
rstop {Location}
[...]
rbody st#[body]
{Statements}
[...]
rpos {Location} {Location} {Location}
{Location}

Definition

`[id]` Unique number identifying this routine. `[name]` The name given this routine in the source file.
`{Location}` The location of this routine in the source file.
`[type]` The id of the tag in the pdb file that defines the type of this Routine.
`[link]` The kind of link this is either: NA, INTERNAL, C, C++, FINT, F90.
`[kind]` The kind of Routine this is either: NA (not defined), ext (external), tproto (template/protoype), fext (Fortran external), fprog (Fortran program), fbdat (Fortran block data), fint (Fortran internal), fstfu (Fortran statement function), fintrin (Fortran intrinsic), fmproc (Fortran module procedure), funspec (Fortran unspecified).
`[virtual]` Either yes or no.
`[groupId]` The Id. for this routines parent class.
`[access]` The access level for this routine.
`[namespaceId]` Id. for the Parent namespace.
`[parentRoutineId]` Id. for the Parent routine.

[aliasId] Id. for this routines alias (through interface).
[implementsId] Id. for the routine this routine implements (for Fortran alaised functions.)
[rstatic] rstatic if this routine is static. (Ommit line otherwise)
[rcrvo] rcrvo if this routine has a covarient return virtual override
[rinline] rinline if this routine is inlined, ommit this line otherwise.
[rexpl] rexpl if this routine is explicit ctor ommit otherwise.
[templateId] Id for the template, if this routine implements one, ommit otherwise.
[rspecl] rspecl if this routine is specialized
[rarginfo] rarginfo if this routine explicatly defined interface.
[rrec] rrec if this routine is declared recursive.
[riselem] riselem if this routine is elemental.
{Location} The location of the first executable statement.
[calleeId]The Id. that defines the type of the routine this routine calles.[virtualCall] Is this call virtual. {Location} The location of this routine call in this routine.
{Location} Location of the return statements.
{Location} The location of the stop statements.
[body]Id. of the first statement in the body of this Routine.
{statements} Each statement inside the body of this Routine.
{Location} {Location} {Location} {Location} Four location tags, in order, the beginning of the Routine declaration, the ending of the Routine declaration, the begining of the Routine body, and lastly the ending of the Routine body.

Name

Types -- This tag records each Type declared in the source file.

ty#[id] [name]
ykind [kind]
yrett ty#[return type]
yargt ty#[argument type]
yexcept ty#[exception type]
yfkind [float type]
yikind [integer type]
yclen [length]
yshape [shape]
yloc {Location}
ygroup gr#[groupId]
yacs [access]
[ysigned]
yenum [name] [value]
[...]
yptr (ty#|gr#)[pointerId]
yref (ty#|gr#)[referenceId]
[yellip]
yqual [qualifier]
yelem (ty#|gr#)[elementId]
[ystat]
ynelem [number of elements]
yrank [rank]
ydim [dimension]
ytref (ty#|gr#)[typedefId]
ympgroup gr#[ptr_groupId]
ymptype (ty#|gr#)[ptr_typeId]

Definition

`[id]` Unique number identifying this Type. `[name]` The name given to this Type in the source file.
`[kind]` The kind of Routine this is, either: NA (not defined), bool (boolean), enum (enumeration) , err (error/exception), func (function), void, int (integer), float, ptr (pointer), ref (reference to memory location), array, tref (template reference), ptrmem, traram, wchar, c_type, ferr (Fortran error), fvoid (Fortran void), fint (Fortran integer), flogic, ffloat (Fortran float), ffunc (Fortran function), fchar (Fortran character), farray (Fortran array), fcmplx (Fortran complex), funspecfunc, fbldat, fmod, fptr, f_type, group.
`[return type]` Id. of the tag that defines the return part of this type (Ommit line if type is not a function type).
`[argument type]`Id. of the tag that defines the parameter part of this type (Ommit line if type is not a function type).
`[execption type]`
`[float type]` The type of this float, either, float, dbl (double), or longdbl (long double). (Ommit line if this type is integer or is non-primitive.)
`[integer type]` The type of this integer, either char (charater), schar (short charater), uchar (unsigned charater), short, ushort (unsigned short), int (integer), uint (unsigned integer), long, ulong (unsigned long), longlong, ulonglong (unsigned long long). (Ommit line if this type is float or is non-primitive.)
`[length]` The length of the array (Ommit line if type is not array.)

[shape] The shape of this Fortran array, either, explicit (set when the rank and extent are defined explicitly), asize (set when the extend of one or more dimension is undefined), ashape (set when the rank of an array is left undefined), deferred (set when an array is allocated but undefined). (Ommit line if type is not a fortran array.)
{Location} Location of where this type is defined in the source code.
[groupId] This type's parent group.
[access] This type's access level.
[ysigned] If this type is signed, ommit otherwise
[name] The name of the enumeration member. [value] The value of the enumeration member.
[pointerId] the type or group Id. that this type is pointing to, ommit if this is not a pointer type.
[referenceId] the type or group Id. that this type is referencing, ommit if this is not a reference type.
[yellip] yellip if this type has ellipsis, ommit line otherwise.
[qualifier] the qualifiers this type has, if any.
[elementId] the type or group of the elements in the array, ommit if this is not an array type.
[ystat] ystat if this type is static.
[ynelem] The number of element in this array, -1 if this array has variable length, ommit line if this is not an array type.
[rank] The rank of this array, ommit line if this is not an array type.
[dimension] The dimension of this array, ommit line if this is not an array type.
[typedefId] The typedef type if there is one.
[ptr_groupId] The type of group to which the type or group that is pointed to belongs.
[ptr_typeId] The type that the type or group that is pointed to belongs.

Name

Locations -- This simple tag is used to store a location in a source file. Used only within other tags, not found by itself.

so#[id] [line number] [colunm number]

Definition

`[id]` The id number of the source file. `[line number]` The line number of the location. `[colunm number]` The colunm number of the location.

Name

Statements -- This simple tag is used to record a statement in the source file. Used only within other tags, not found by itself.

stmt st#[id] [name] {Location} {Location} [next] [down]

Definition

`[id]` the unique number identifying this statement. `[kind]` the kind of statement, either, na (not defined), switch, case, init, return, if, empty, for, goto, continue, break, label, block, asm, expr, assign, throw, while, do, try, catch, decl, set_vla_size, vla_decl, vla_dealloc, fallocate, fassign, fio, fdo, fdeallocate, freturn, fif, fgoto, fsingle_if, fstop, or flabel. `{Location} {Location}` Two locations, where the statement begins and where it ends. `[next]` Id number of the next statement (NA if there is none). `[down]` Id number of the statement beneath this one, (ie, if this were a block statement, then this would be the first statement within the block.) (NA if there is none).

Name

Groups -- This simple tag is used to record a group (classes in C) in the source code.

gr#[Id] [name]
gloc {Location}
ggroup gr#[groupID]
gacs [access]
gnspace [namespaceID]
gkind [kind]
gtempl te#[templateID]
[gspecl]
[gsparam] [type] (ty#|gr#)[paramId]
[...]
gbase [virtual] [access] gr#[baseId] {Location}
[...]
gfrgroup gr#[friendId] {Location}
[...]
gfrfunc ro#[routineID] {Location}
[...]
gfunc ro#[member_routineId] {Location}
[...]
gmem [members]
[...]
gmloc {Location}
gmgroup gr#[parentId]
gmacs [member_access]
gmkind [member_kind]
gmtype (ty#|gt#)[member_type]
gmtempl te#[templateId]
[gmspecl]
[gmconst]
[gmisbit]
[gmmut]
[gmconst]
[gmtempl] te#[templateId]
gpos {Location} {Location} {Location} {Location}

Definition

`[id]` The unique number identifying this group.
`[name]` The name of this group.
`{Location}` Location where the group header begins.
`[groupId]` The group Id. of this group's parent.
`[gacs]` This group's access level.
`[namespaceId]` This group's parent namespace.
`[kind]` The kind of groups this is.
`[templateId]` The id of the template this group implements, if it implement one.
`[gspecl]` gspecl if this group is specialized.
`[type]` The type of this parameter. `[paramId]` the id. for this parameter's type.
`[virtual]` vir if the direct base group is virtual. `[access]` the access level for the base group. `[base_groupId]` the id of this groups base group. `{Location}`
`[friendId]` The Id. of the friend groups of this group. `{Location}` Location of this friend group.

[routineId] The Id. of the friend functions of this group. {Location} Location of this friend function.
[member_routineId] the Id. for the routines that are members of this group. {Location} The location of this member routine.
[member] The name of the non-function members of this group.
{location} the location of this member.
[parentId] the parent group of this member.
[member_access] the access level of this member group.
[member_kind] the kind of the member this is.
[member_typeId] the type id of this member
[member_templateId] this member's templete Id, if it implements one.
[gmspecl]gmspecl if this member is specialized.
[gmconst] gspecl if this member is initalized.
[gmisbit] gmisbit if this member is a bit field.
[gmmut] gmmut if this member is mutable.
{Location} {Location} {Location} {Location} Four location tags, in order, the beginning of the Group declaration, the ending of the Group declaration, the begining of the Group body, and lastly the ending of the Group body.

Name

Templates -- These tags record templates in the source files.

te#[templateId]
tloc {Location}
tgroup gr#[groupId]
tacs [access]
tnspace na#[namespaceId]
tdecl te#[declareId]
tdef te#[definitionID]
tkind [kind]
tparam [type_name] ty#[typeId]
[...]
tsparam [...] [param_name] (ty#|gr#)[paramId]
[...]
tproto ro#[routineId]
ttype (ty#|gr#)[variable_typeId]
ttext [template_text]
tpos {Location} {Location} {Location} {Location}

Defintions

`[templateId]` The unique number identifying this template item.
`{Location}` The location where this template definition starts.
`[groupId]` the Id. of this template's parent group.
`[access]` this template access level.
`[namespaceId]` this template's parents namespace.
`[declareId]` The Id of the template which declares this one, if one does.
`[definitionId]` The Id of the template which defines this one, if one does.
`[kind]` The kind of template this is.
`[type_name]` The type of any template parameters. `[typeId]` The Id of the type that defines this parameter.
`[param_name]` The name of this template parameter. `[parameterId]` The Id. of the type or groups that defines this template parameter.
`[routineId]` The Id. of this template's prototype instantiation.
`[variable_typeId]` The Id. of the type that defines this variable.
`[template_text]` The actual text of this template.
`{Location} {Location} {Location} {Location}` Four location tags, in order, the beginning of the Template declaration, the ending of the Template declaration, the begining of the Template body, and lastly the ending of the Template body.

Name

Namespaces -- These tags record the namespaces inside the source file.

na#[namespaceID] [name]
nloc {Location}
nnspace na#[parentId>
nmem (ty#|ro#|gr#)[memberId]
[...]
nalias [alias_name]
npos {Location} {Location} {Location} {Location}

Definition

`[namespaceId]` The unique number identifying this namespace. `[name]` The name of this namespace.
`{Location}` The location where the definition of this namespace starts in the source file.
`[parentId]` The Id. of this namespaces parent namespace.
`[memberId]` The Id. of the type, routine, or group that defines this member of the namespace.
`[alias_name]` The name of this namespaces alias.
`{Location} {Location} {Location} {Location}` Four location tags, in order, the beginning of the Namespace declaration, the ending of the Namespace declaration, the begining of the Namespace body, and lastly the ending of the Namespace body.

Name

Macros -- These tags record each occurance of a marco in the source file.

ma#[macroId] [name]
mloc {Location}
mkind [kind]
mtext [text]

Definition

`[macroId]`
`{Location}` The location in the source file where this macro begins.
`[kind]` The kind of macro this is, either defined or undefined.
`[text]` The actual text of this macro.

Name

Pragmas -- These tags record every Pragma within the source file.

pr#[pragmaId] [name]
ploc {Location}
pkind [kind]
ppos {Location} {Location}
ptext [text]

Definition

`{Location}` The location in the source file where this pragma begins.
`[kind]` The kind of Pragma this is.
`{Location}{Location}` The location where this pragma begins and ends.
`[text]` The actual text of the pragma.

PDT's ductape API Tutorial

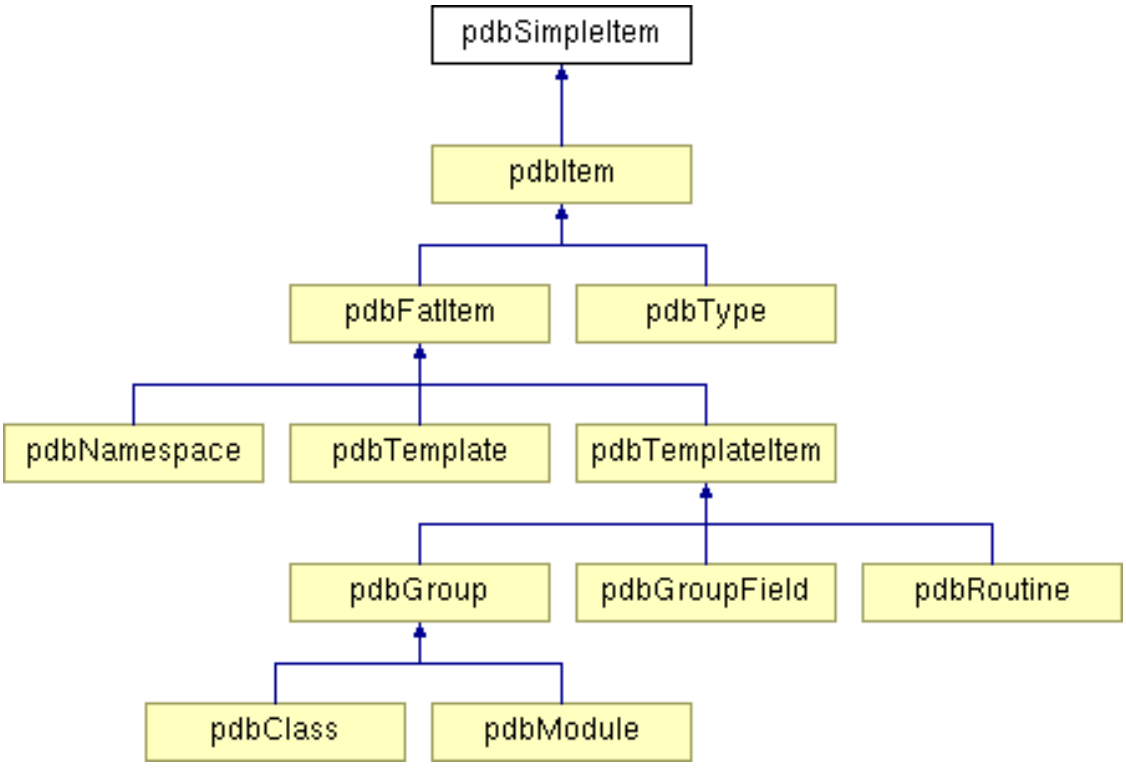
Table of Contents

1. Overview of PDT's Architecture

In this tutorial you will learn how to use PDT's ductape API to read PDB files. A PDB file is created by parsing a C/C++ or Fortran source with the include parser: cparse, cxxparse, f90parse and f95parse. To use these parsers type: `%> export PATH=[path-to-pdt]/[arch]/bin/ %> cxxparse program.cpp %> ls program.ccp program.pdb`

The Ductape API is organized as a heiarchy of classes. Here is an picture repersent some of those classes. More detail can be found at API documentation [http://www.cs.uoregon.edu/research/pdt/docs/ductape/index.html]

Figure 1. Partial heiarchy of the classes in the ductape API



2. Using Iterators

Ductape API allows iterators to used in many of its classes. Using iterators will allow you to access member of a datastructor without needing to know the underlying implementation.

One place where you can use iterators is in the pdb class, here is a simple function that iterates over every class in the file printing its name. File to be parsed and analyized: `class bar { int foo(int v) { return v + 2; } class bar2 { int rountine(bool t) {return 0;} }; int a; };` C source file: `#include "pdbAll.h" #include "stdio.h" int main(int argc, char *argv[]) { // Read the pdb file as input for this program. PDB p(argv[1]); if (!p) return 1; // Iterate through each class in the pdb file and print its name. for (PDB::classvec::iterator r = p.getClassVec().begin(); r!=p.getClassVec().end(); r++) { cout << (*r)->name() << endl; } return 0; }` To run type: `%> g++ -I../inc/ -o vector vector.cc ../lib/libpdb.a %> cxxparse testApp.cc %> ./vector testApp.pdb bar bar2` There is a collection of example source code in the API documentation [http://www.cs.uoregon.edu/research/pdt/docs/ductape/examples.html].

TITLE

AUTHOR
Version 1
CREATEDATE

Table of Contents

Table of contents

PDT Ductape API Hierarchical Index

PDT Ductape API Class Hierarchy

PDT Ductape API Data Structure Index

PDT Ductape API Data Structures

Here are the data structures with brief descriptions:

PDT Ductape API Data Structure Documentation

PDB Class Reference

A class to control the reading and writing of pdb files.

```
#include <pdb.h>
```

Public Types

- enum **lang_t**

Public Member Functions

- **PDB** (char *fname)
- typevec & **getTypeVec** ()
- filevec & **getFileVec** ()
- classvec & **getClassVec** ()
- modulevec & **getModuleVec** ()
- croutinevec & **getCRoutineVec** ()
- froutinevec & **getFRoutineVec** ()
- templatevec & **getTemplateVec** ()
- macrovec & **getMacroVec** ()
- pragmavec & **getPragmaVec** ()
- namespacevec & **getNamespaceVec** ()
- itemvec & **getItemVec** ()

Data Structures

- class **classTag**
- class **croutineTag**
- class **fileTag**
- class **froutineTag**
- struct **ltstr**
- class **macroTag**
- class **moduleTag**
- class **namespaceTag**
- class **pragmaTag**
- class **templateTag**
- class **typeTag**

Detailed Description

A class to control the reading and writing of pdb files.

In addition, there is a class PDB that represents an entire PDB file. It provides methods to read, write, and merge PDB files, to get the version of the PDB file format and the programming language it got generated from.

Examples:

froutine.cc, **stmt.cc**, and **vector.cc**.

Member Enumeration Documentation

enum PDB::lang_t

the language of the source files.

Constructor & Destructor Documentation

PDB::PDB (char * *fname*)

A PDB class constructor

Parameters:

**fname* the name of the source file.

Member Function Documentation

PDB::classvec & PDB::getClassVec () `[inline]`

a vector of the classes within the pdb.

PDB::croutinevec & PDB::getCRoutineVec () `[inline]`

a vector of the c/c++ routines within the pdb.

PDB::filevec & PDB::getFileVec () `[inline]`

a vector of the files within the pdb.

PDB::froutinevec & PDB::getFRoutineVec () `[inline]`

a vector of the fortran routines within the pdb.

PDB::itemvec & PDB::getItemVec () `[inline]`

a vector of the items within the pdb.

PDB::macrovec & PDB::getMacroVec () `[inline]`

a vector of the macros within the pdb.

PDB::modulevec & PDB::getModuleVec () `[inline]`

a vector of the modules within the pdb.

PDB::namespacevec & PDB::getNamespaceVec () `[inline]`

a vector of the namespaces within the pdb.

PDB::pragmavec & PDB::getPragmaVec () `[inline]`

a vector of the pragmas within the pdb.

PDB::templatevec & PDB::getTemplateVec () `[inline]`

a vector of the templates within the pdb.

PDB::typevec & PDB::getTypeVec () `[inline]`

a vector of the types within the pdb.

The documentation for this class was generated from the following files:

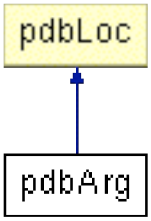
- pdb.h
- pdb.inl
- pdbTdefs.h

pdbArg Class Reference

A class for arguments for routines.

`#include <pdbType.h>`

Inheritance diagram for pdbArg:



Public Member Functions

- const **pdbType** * **type** () const
- const string & **name** () const
- bool **hasDefault** () const

Detailed Description

A class for arguments for routines.

This class describes arguments given to pbdRoutines. It hold information about these arguments and how they are given to Routines as parameters.

Member Function Documentation

bool pdbArg::hasDefault () const `[inline]`

if the argument of this routine has a default parameter for this argument.

const string & pdbArg::name () const `[inline]`

the name of the argument.

const pdbType * pdbArg::type () const `[inline]`

the abstract type of the argument.

The documentation for this class was generated from the following files:

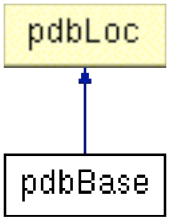
- pdbType.h
- pdbType.inl

pdbBase Class Reference

A class to define a super class.

`#include <pdbClass.h>`

Inheritance diagram for pdbBase:



Public Member Functions

- **pdbBase** ()
- **pdbBase** (pdbItem::virt_t v, **pdbItem::access_t** a, const **pdbClass** *base, pdbFile *file, int line, int col)
- pdbItem::virt_t **virtuality** () const
- **pdbItem::access_t access** () const
- const **pdbClass** * **base** () const
- bool **isVirtual** () const

Detailed Description

A class to define a super class.

pdbBase describes a base class (super class) of a **pdbClass**. It provides methods to ask for the base class, its access mode (e.g., public or private), and whether the derivation was virtual.

Constructor & Destructor Documentation

pdbBase::pdbBase () `[inline]`

A constructor with no arguments

pdbBase::pdbBase (pdbItem::virt_t *v*, pdbItem::access_t *a*, const pdbClass * *base*, pdbFile * *file*, int *line*, int *col*) `[inline]`

A constructor

Parameters:

v the virtual type of this class.
a the access type of this class.
**base* a pointer to the base class.
**file* a pointer to the source file.
line the line number where this class begans.

col the column number where this class begans.

Member Function Documentation

pdbItem::access_t pdbBase::access () const `[inline]`

the access type of this class

const pdbClass * pdbBase::base () const `[inline]`

a pointer to the base class

bool pdbBase::isVirtual () const `[inline]`

is this class derivation virtual?

pdbItem::virt_t pdbBase::virtuality () const `[inline]`

the virtual type of this class

The documentation for this class was generated from the following files:

- pdbClass.h
- pdbClass.inl

pdbBounds Class Reference

A class to repersent Fortran bounds.

`#include <pdbType.h>`

Public Member Functions

- **pdbBounds** (int low, int upp)
- int **lower** () const
- int **upper** () const

Detailed Description

A class to repersent Fortran bounds.

pdbBounds is used to describe the bounds of one dimension of a Fortran array.

Constructor & Destructor Documentation

pdbBounds::pdbBounds (int *low*, int *upp*) `[inline]`

A constructor

Parameters:

low lower bound.
upp upper bound.

Member Function Documentation

int pdbBounds::lower () const `[inline]`

lower bound.

int pdbBounds::upper () const `[inline]`

upper bound.

The documentation for this class was generated from the following files:

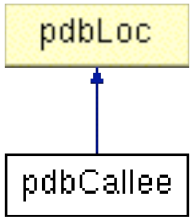
- pdbType.h
- pdbType.inl

pdbCallee Class Reference

A class to repersent a routine call.

`#include <pdbRoutine.h>`

Inheritance diagram for pdbCallee:



Public Member Functions

- const pdbRoutine * **call** () const
- bool **isVirtual** () const

Detailed Description

A class to repersent a routine call.

pdbCallee is used to represent a routine call (i.e., a call site). Attributes are the routine called, whether it is was called virtually, and the location of the call site.

Member Function Documentation

const pdbRoutine * pdbCallee::call () const `[inline]`

A pointer to the routine called.

bool pdbCallee::isVirtual () const `[inline]`

Is this a virtual call.

The documentation for this class was generated from the following files:

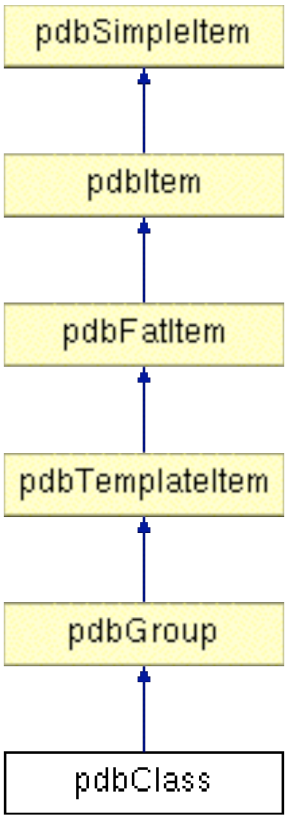
- pdbRoutine.h
- pdbRoutine.inl

pdbClass Class Reference

This class defines the attributes for a class.

```
#include <pdbClass.h>
```

Inheritance diagram for pdbClass:



Public Member Functions

- const classvec & **derivedClasses** () const
- const methodvec & **methods** () const
- const friendclassvec & **friendClasses** () const
- const friendfuncvec & **friendRoutines** () const

Detailed Description

This class defines the attributes for a class.

This class defines the genric class within a class hierarchy, and its associated friend classes.

Member Function Documentation

const pdbClass::classvec & pdbClass::derivedClasses () const `[inline]`

a vector listing the classes derived from this one.

const pdbClass::friendclassvec & pdbClass::friendClasses () const `[inline]`

a vector of friendly classes.

const pdbClass::friendfuncvec & pdbClass::friendRoutines () const `[inline]`

a vector of friendly Routines.

const pdbClass::methodvec & pdbClass::methods () const `[inline]`

a vector of methods.

The documentation for this class was generated from the following files:

- pdbClass.h
- pdbClass.inl

pdbComment Class Reference

`#include <pdbFile.h>`

Public Member Functions

- **pdbComment** (int id)
- **PDB::lang_t kind** () const
- const **pdbLoc** & **cmtBegin** () const
- const **pdbLoc** & **cmtEnd** () const
- const string & **text** () const

Detailed Description

A class to repesent comment in the source file.

pdbComment represents a comment in a source file. Comments are numbered 0 to N inside one file. The kind, its exact location, and the comment text is available.

Constructor & Destructor Documentation

pdbComment::pdbComment (int *id*) `[inline]`

A constructor

Parameters:

id unique identifier.

Member Function Documentation

const pdbLoc & pdbComment::cmtBegin () const `[inline]`

location in the source file where the comment begans.

const pdbLoc & pdbComment::cmtEnd () const `[inline]`

location in the source file where the comment ends.

PDB::lang_t pdbComment::kind () const `[inline]`

returns the language of this source file.

const string & pdbComment::text () const `[inline]`

the contents of the comment.

The documentation for this class was generated from the following files:

- pdbFile.h

- pdbFile.inl

pdbEnum Class Reference

A class to repersent C/C++ enumuations.

`#include <pdbType.h>`

Public Member Functions

- **pdbEnum** (const char *id, int val)

Detailed Description

A class to repersent C/C++ enumuations.

pdbEnum describes one element of an C / C++ enumeration type by its name (identifier) and the corresponding integer value.

Constructor & Destructor Documentation

pdbEnum::pdbEnum (const char * *id*, int *val*) `[inline]`

A constructor

Parameters:

**id* unique idenifier.
val interger value of the enumuation.

The documentation for this class was generated from the following files:

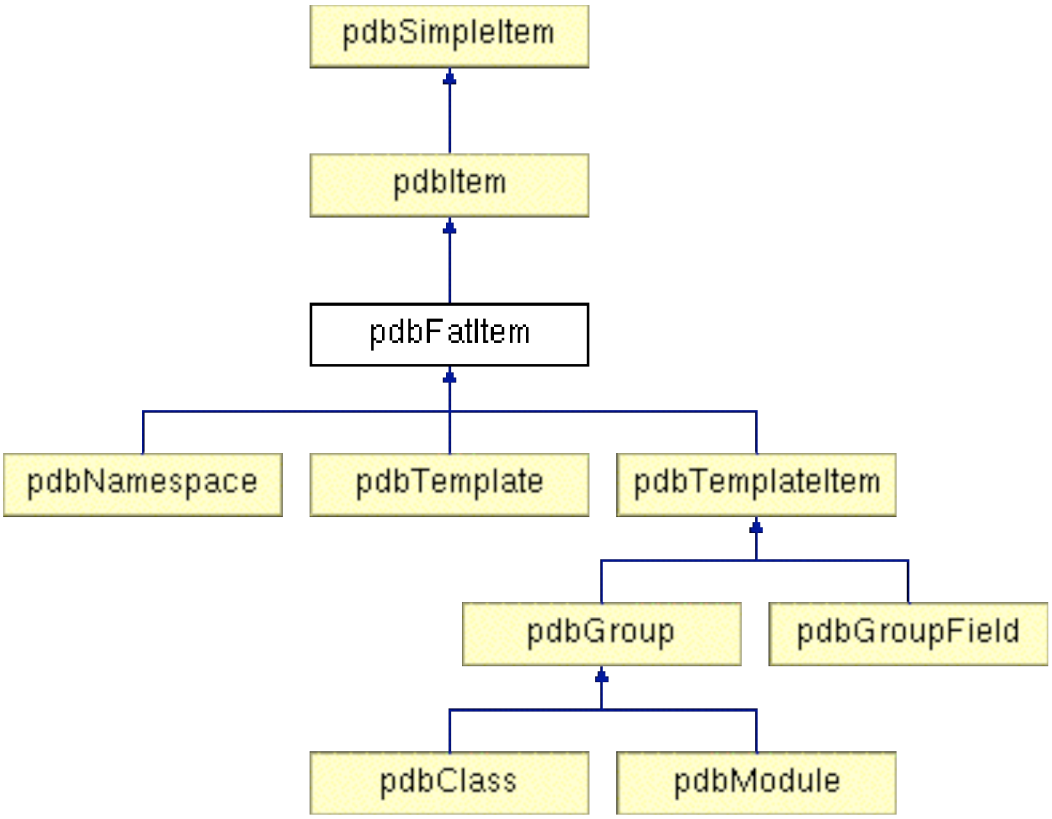
- pdbType.h
- pdbType.inl

pdbFatItem Class Reference

A class for items spaning serval lines of code.

`#include <pdbFatItem.h>`

Inheritance diagram for pdbFatItem:



Public Member Functions

- **pdbFatItem** (int id)
- **pdbFatItem** (const string &name, int id)
- const **pdbLoc** & **headBegin** () const
- const **pdbLoc** & **headEnd** () const
- const **pdbLoc** & **bodyBegin** () const
- const **pdbLoc** & **bodyEnd** () const

Detailed Description

A class for items spaning serval lines of code.

pdbItems are pdbPragmas, pdbMacros, pdbTypes, or so-called fat items. pdbFatItems have a header and a body, and attributes describing the source location of these parts.

Constructor & Destructor Documentation

pdbFatItem::pdbFatItem (int *id*) `[inline]`

pdbFatItem constructor

Parameters:

id an unique idenifier.

pdbFatItem::pdbFatItem (const string & *name*, int *id*) `[inline]`

pdbFatItem constructor

Parameters:

name the name of the item.
id an unique idenifier.

Member Function Documentation

const pdbLoc & pdbFatItem::bodyBegin () const `[inline]`

the line number in the source code that begans the body.

const pdbLoc & pdbFatItem::bodyEnd () const `[inline]`

the line number in the source code that ends the body.

const pdbLoc & pdbFatItem::headBegin () const `[inline]`

the line number in the source code that begans the header.

const pdbLoc & pdbFatItem::headEnd () const `[inline]`

the line number in the source code that ends the header.

The documentation for this class was generated from the following files:

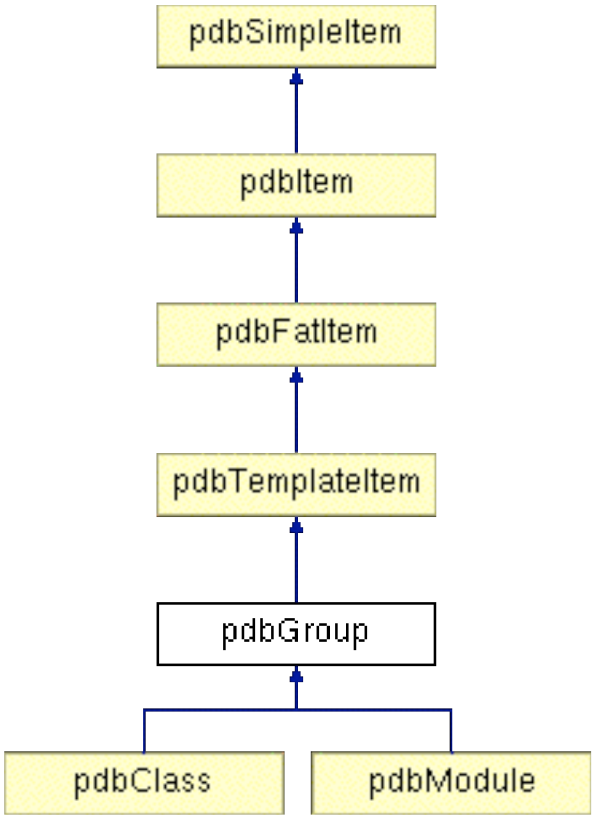
- pdbFatItem.h
- pdbFatItem.inl

pdbGroup Class Reference

pdbGroups repersenting abstract data types.

`#include <pdbGroup.h>`

Inheritance diagram for pdbGroup:



Public Member Functions

- **pdbGroup** (int id)
  pdbGroup constructor.

- **pdbGroup** (const string &name, int id)
  pdbGroup constructor.

- const fieldvec & **dataMembers** () const
- **group_t kind** () const
  the type of this group.

Detailed Description

pdbGroups repersenting abstract data types.

Groups represent abstract data types, i.e. collections of public and private members. Members are divided into data members (described by pdbGroupFields) and member functions/methods (described by pdbRoutines). The different kind of groups are Fortran 90 derived types or modules, or C and C++ structs, unions, or classes.

Constructor & Destructor Documentation

pdbGroup::pdbGroup (int *id*) `[inline]`

pdbGroup constructor.

Parameters:

id an unique idenifier.

pdbGroup::pdbGroup (const string & *name*, int *id*) `[inline]`

pdbGroup constructor.

Parameters:

name the name of the item.
id an unique idenifier.

Member Function Documentation

const pdbGroup::fieldvec & pdbGroup::dataMembers () const `[inline]`

A list of member of this group.

pdbItem::group_t pdbGroup::kind () const `[inline]`

the type of this group.

kind() specifies the abstract type of this group wheather F90 derived types or modules, C/C++ structs, unions, or classes.

The documentation for this class was generated from the following files:

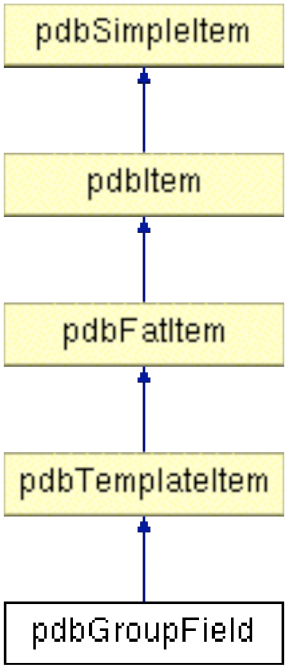
- pdbGroup.h
- pdbGroup.inl

pdbGroupField Class Reference

A class to define field within a group.

`#include <pdbGroupField.h>`

Inheritance diagram for pdbGroupField:



Public Member Functions

- mem_t **kind** () const
- const **pdbType** * **type** () const
- bool **isBitField** () const
- bool **isMutable** () const
- bool **isStaticConst** () const

Detailed Description

A class to define field within a group.

Member Function Documentation

bool pdbGroupField::isBitField () const `[inline]`

Is this a bit field?

bool pdbGroupField::isMutable () const `[inline]`

Is this field mutable?

bool pdbGroupField::isStaticConst () const `[inline]`

Is this a static constant

pdbItem::mem_t pdbGroupField::kind () const `[inline]`

the memory type of this field

const pdbType * pdbGroupField::type () const `[inline]`

the type of this field

The documentation for this class was generated from the following files:

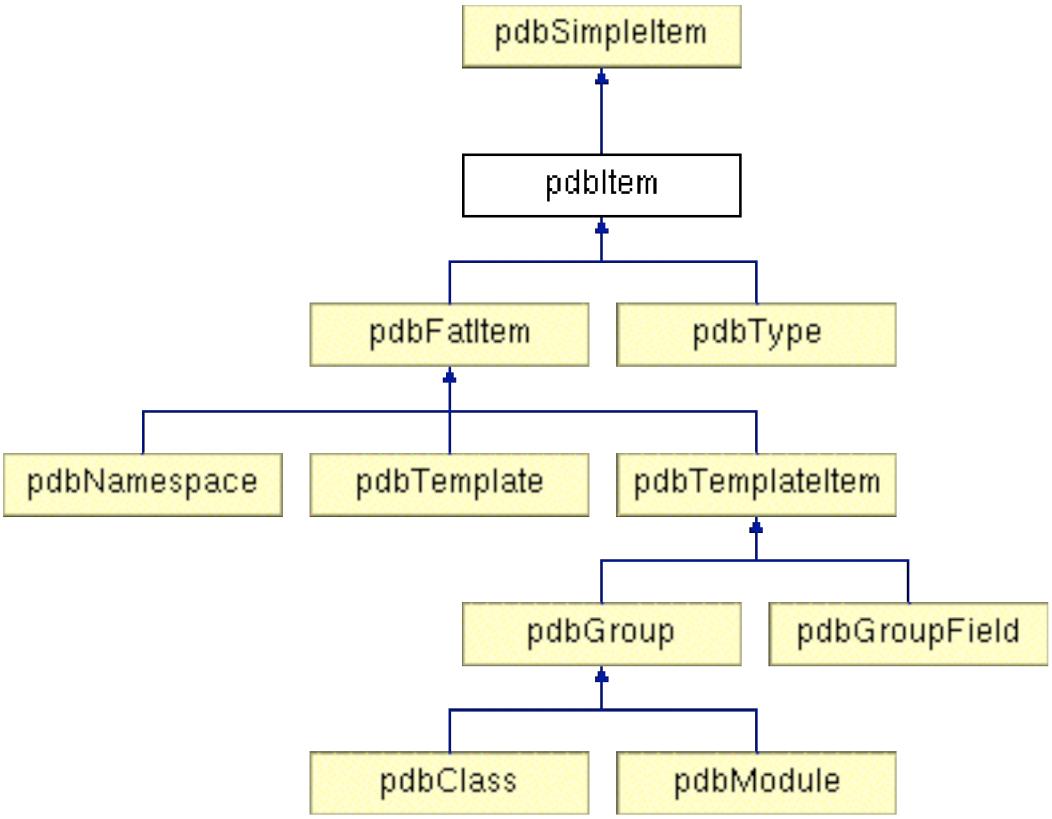
- pdbGroupField.h
- pdbGroupField.inl

pdbItem Class Reference

An item class with more complex members.

`#include <pdbItem.h>`

Inheritance diagram for pdbItem:



Public Types

- enum **access_t** { **AC_NA**, **AC_PRIV**, **AC_PROT**, **AC_PUB** }
- enum **routine_t** { **RO_NA**, **RO_EXT**, **RO_TPROTO**, **RO_FEXT**, **RO_FPROG**, **RO_FBLDAT**, **RO_FINTRIN**, **RO_FINT**, **RO_FSTFN**, **RO_FMPROC**, **RO_FUNSPEC**, **RO_FALIAS** }
- enum **rspec_t**
- enum **templ_t** { , **TE_CLASS**, **TE_FUNC**, **TE_MEMCLASS**, **TE_MEMFUNC** }
- enum **float_t** { **FL_NA**, **FL_FLOAT**, **FL_DBL**, **FL_LONGDBL** }
- enum **int_t** { **I_NA**, **I_CHAR**, **I_SCHAR**, **I_UCHAR**, **I_SHORT**, **I_USHORT**, **I_INT**, **I_UINT**, **I_LONG**, **I_ULONG**, **I_LONGLONG**, **I_ULONGLONG** }
- enum **type_t**
- enum **group_t** { , **GR_CLASS**, **GR_STRUCT**, **GR_UNION**, **GR_TPROTO**, **GR_FDERIVED**, **GR_FMODULE** }
- enum **link_t** { **LK_NA**, **LK_INTERNAL**, **LK_CXX**, **LK_C**, **LK_FINT**, **LK_F90** }
- enum **shape_t** { **SH_NA**, **SH_EXPLICIT**, **SH_ASIZE**, **SH_ASHAPE**, **SH_DEFERRED** }
- enum **qual_t** { **QL_NA**, **QL_CONST**, **QL_VOLATILE**, **QL_RESTRICT** }

Public Member Functions

- **pdbItem** (int id)
  pdbItem constructor.

- **pdbItem** (const string &name, int id)
  pdbItem constructor.

- const string & **fullName** () const
  the full name of the item.

- **access_t access** () const
  access mode for this item.

- const **pdbGroup** * **parentGroup** () const
  the groups this item is a member of.

- const **pdbLoc** & **location** () const
  the location of this item in the source file.

- const **pdbNamespace** * **parentNSpace** () const
  the name space this item is in.

Detailed Description

An item class with more complex members.

Derived from pdbSimpleItems are pdbFiles and more complex pdbItems, which have a source code location, possibly a parent group or namespace, and an access mode (e.g., public or private) if they are member of a group. The method fullname() returns fully-qualified names (including signatures for routines).

Member Enumeration Documentation

enum pdbItem::access_t

defines the types of access modifiers for template items.

Enumerator:

AC_NA defualt

AC_PRIV private

AC_PROT protected

AC_PUB public

enum pdbItem::float_t

defines the types of floating point numbers.

Enumerator:

FL_NA not applicable

FL_FLOAT float type

FL_DBL double type

FL_LONGDBL long double type

enum pdbItem::group_t

defines the types of groups.

Enumerator:

GR_CLASS class group

GR_STRUCT structure group

GR_UNION union group

GR_TPROTO template prototype group

GR_FDERIVED Fortran derived group

GR_FMODULE Fortran module group

enum pdbItem::int_t

defines the types of interger point numbers.

Enumerator:

I_NA not applicable

I_CHAR character

I_SCHAR signed character

I_UCHAR unsigned character

I_SHORT short

I_USHORT unsigned short

I_INT integer

I_UINT unsigned integer

I_LONG long

I_ULONG unsigned long

I_LONGLONG long long

I_ULONGLONG unsigned long long

enum pdbItem::link_t

Enumerator:

LK_NA link not defined

LK_INTERNAL internal linke

LK_CXX c++ link

LK_C c link

LK_FINT Fortran link

LK_F90 Fortran 90 link

enum pdbItem::qual_t

Enumerator:

QL_NA unqualified type

QL_CONST constant type

QL_VOLATILE volatile type

QL_RESTRICT restricted type

enum pdbItem::routine_t

defines the types of routine signatures

Enumerator:

RO_NA default routine

RO_EXT external routines, created by the complier not explicitly written in the source code

RO_TPROTO templete routine

RO_FEXT Fortran external routine

RO_FPROG Fortran program routine

RO_FBLDAT Fortran block data

RO_FINTRIN Fortran intrinsic

RO_FINT Fortran internal

RO_FSTFN Fortran statement function

RO_FMPROC Fortran module procedure

RO_FUNSPEC Fortran unspecified

RO_FALIAS Fortran alias

enum pdbItem::rspec_t

the types of special routines.

enum pdbItem::shape_t

defines the different shapes of fortran arrays.

Enumerator:

SH_NA initalized value. (for debugging purposes).

SH_EXPLICIT set when the rank and extent are defined explicitly.

SH_ASIZE set when the extend of one or more dimension is undefined.

SH_ASHAPE set when the rank of an array is left undefined.

SH_DEFERRED set when an array is allocated but undefined.

enum pdbItem::templ_t

Enumerator:

TE_CLASS template classes

TE_FUNC template function

TE_MEMCLASS classes that are members of a template class

TE_MEMFUNC functions that are members of a template class

enum pdbItem::type_t

defines other types of primatives.

Constructor & Destructor Documentation

pdbItem::pdbItem (int *id*) `[inline]`

pdbItem constructor.

Parameters:

id an unique idenifier.

pdbItem::pdbItem (const string & *name*, int *id*) `[inline]`

pdbItem constructor.

Parameters:

name the name of the item.
id an unique idenifier.

Member Function Documentation

const string & pdbItem::fullName () const `[inline]`

the full name of the item.

The full name contains the full signatures for templates.

The documentation for this class was generated from the following files:

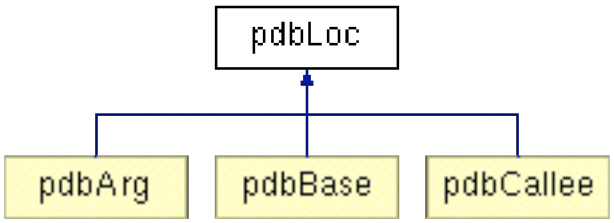
- pdbItem.h
- pdbItem.inl

pdbLoc Class Reference

class that store the location of pdbItems in the source code.

`#include <pdbSimpleItem.h>`

Inheritance diagram for pdbLoc:



Public Member Functions

- **pdbLoc** ()
- **pdbLoc** (const pdbFile *file, int line, int col)
- const pdbFile * **file** () const
- int **line** () const
- int **col** () const

Detailed Description

class that store the location of pdbItems in the source code.

pdbLoc describes source code locations which are characterized by a source file, a line number (starting with 1), and a character position within this line (starting with 0).

Constructor & Destructor Documentation

pdbLoc::pdbLoc () `[inline]`

A constructor without any argument

pdbLoc::pdbLoc (const pdbFile * *file*, int *line*, int *col*) `[inline]`

A constructor

Parameters:

**file* pointer to the pdbFile.
line the line number of this location in the source file.
col the column number of this location in the source file.

Member Function Documentation

int pdbLoc::col () const `[inline]`

line number in source code.

const pdbFile * pdbLoc::file () const `[inline]`

pointer of a pdbFile.

int pdbLoc::line () const `[inline]`

line number in source code.

The documentation for this class was generated from the following files:

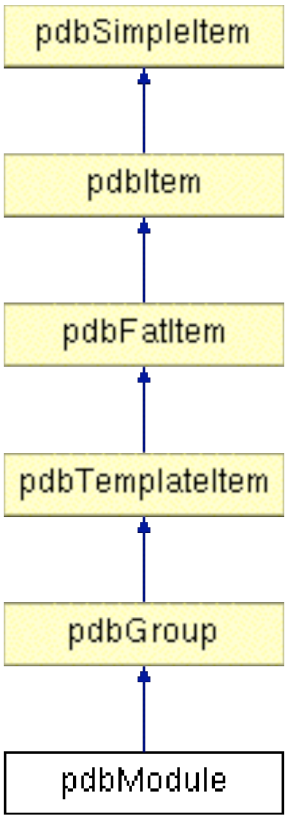
- pdbSimpleItem.h
- pdbSimpleItem.inl

pdbModule Class Reference

A class to define modules.

`#include <pdbModule.h>`

Inheritance diagram for pdbModule:



Public Member Functions

- **pdbModule** (int id)
- **pdbModule** (const string &name, int id)

Detailed Description

A class to define modules.

pdbModule hold information about module and their functions.

Constructor & Destructor Documentation

pdbModule::pdbModule (int *id*) `[inline]`

pdbModule constructor.

Parameters:

id unique idenfier.

pdbModule::pdbModule (const string & *name*, int *id*) `[inline]`

pdbModule constructor.

Parameters:

name the name of this module.
id unique idenfier.

The documentation for this class was generated from the following files:

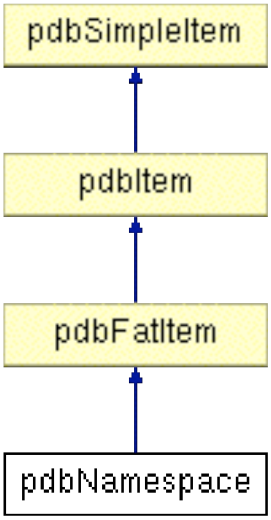
- pdbModule.h
- pdbModule.inl

pdbNamespace Class Reference

A class to define the namespace.

`#include <pdbNamespace.h>`

Inheritance diagram for pdbNamespace:



Public Member Functions

- **pdbNamespace** (int id)
- **pdbNamespace** (const string &name, int id)
- const memvec & **members** () const
- const **pdbNamespace** * **isAlias** () const

Detailed Description

A class to define the namespace.

This class records the members of each namespace.

Constructor & Destructor Documentation

pdbNamespace::pdbNamespace (int *id*) `[inline]`

pdbNamespace constructor

Parameters:

id unique idenfier.

pdbNamespace::pdbNamespace (const string & *name*, int *id*) `[inline]`

pdbNamespace constructor

Parameters:

name this namespace's name.
id unique idenfier.

Member Function Documentation

const pdbNamespace * pdbNamespace::isAlias () const `[inline]`

Pointer to alias namespace.

const pdbNamespace::memvec & pdbNamespace::members () const `[inline]`

A vector of members of this namespace.

The documentation for this class was generated from the following files:

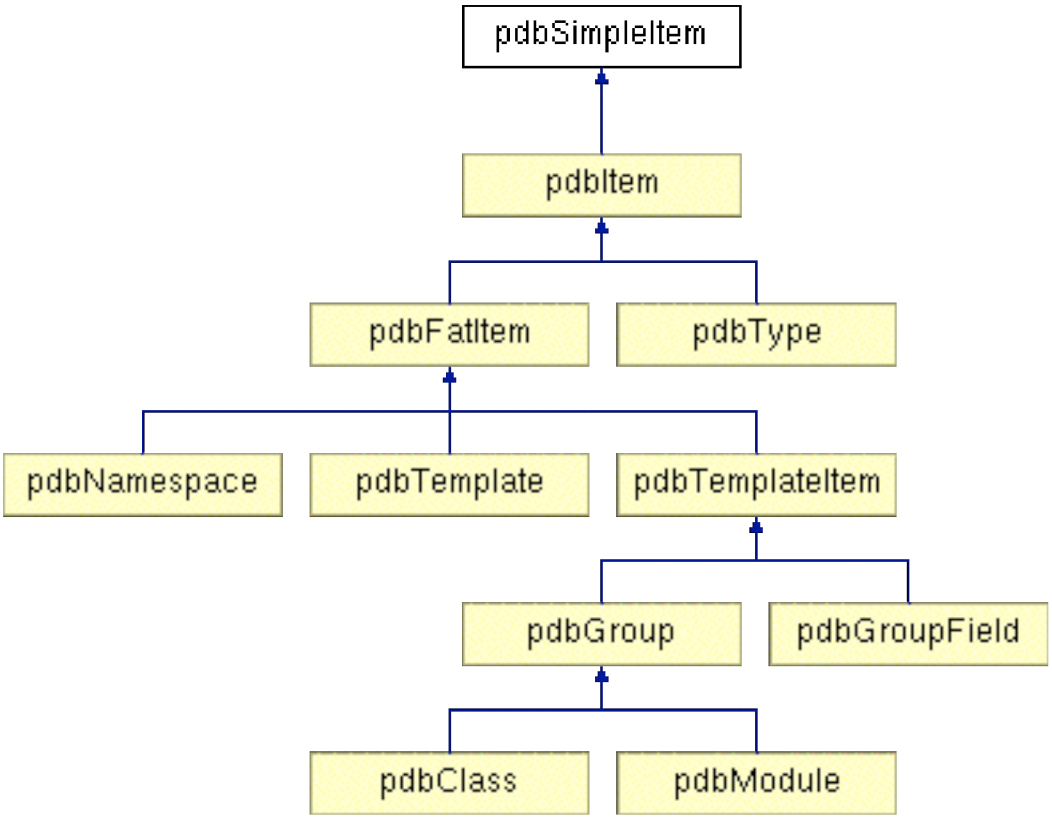
- pdbNamespace.h
- pdbNamespace.inl

pdbSimpleItem Class Reference

The Root class is the pdb hierarchy.

`#include <pdbSimpleItem.h>`

Inheritance diagram for pdbSimpleItem:



Public Member Functions

- **pdbSimpleItem** (int id)
  pdbSimpleItem constructor.

- **pdbSimpleItem** (const string &name, int id)
  pdbSimpleItem constructor.

- const string & **name** () const
  Item's Name.

- int **id** () const
  Unique ID.

Detailed Description

The Root class is the pdb hierarchy.

The root class of the hierarchy is pdbSimpleItem. pdbSimpleItems, and therefore all items derived from it, have two attributes, their name and **PDB** ID.

Constructor & Destructor Documentation

pdbSimpleItem::pdbSimpleItem (int *id*) `[inline]`

pdbSimpleItem constructor.

Parameters:

id an unique identifier.

pdbSimpleItem::pdbSimpleItem (const string & *name*, int *id*) `[inline]`

pdbSimpleItem constructor.

Parameters:

name the name of this item.
id an unique identifier.

Member Function Documentation

int pdbSimpleItem::id () const `[inline]`

Unique ID.

Every pdb Item has a unqiue identifier.

const string & pdbSimpleItem::name () const `[inline]`

Item's Name.

String to hold the Item's name.

The documentation for this class was generated from the following files:

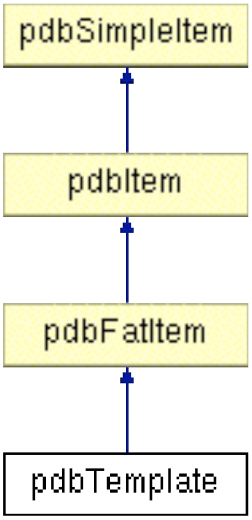
- pdbSimpleItem.h
- pdbSimpleItem.inl

pdbTemplate Class Reference

Template Items class.

`#include <pdbTemplate.h>`

Inheritance diagram for pdbTemplate:



Public Member Functions

- **pdbTemplate** (int id)
- **pdbTemplate** (const string &name, int id)
- **templ_t kind** () const
- const targvec & **arguments** () const
- const targvec & **speclArguments** () const

Detailed Description

Template Items class.

pdbFatItems include pdbTemplates, pdbNamespaces, and pdbTemplateItems. pdbTemplateItems are entities that can be instantiated from templates. Template items are pdbGroups, pdbGroupFields, pdbRoutines.

Constructor & Destructor Documentation

pdbTemplate::pdbTemplate (int *id*) `[inline]`

pdbTemplate constructor

Parameters:

id an unique idenifier.

pdbTemplate::pdbTemplate (const string & *name*, int *id*) `[inline]`

pdbTemplate constructor

Parameters:

name the name of the template.
id an unique idenifier.

Member Function Documentation

const pdbTemplate::targvec & pdbTemplate::arguments () const `[inline]`

the argument for this template.

pdbTemplate::templ_t pdbTemplate::kind () const `[inline]`

the type of this template.

const pdbTemplate::targvec & pdbTemplate::speclArguments () const `[inline]`

a vector containing the argument for a routine.

The documentation for this class was generated from the following files:

- pdbTemplate.h
- pdbTemplate.inl

pdbTemplateArg Class Reference

A class to define argument in a template defintions.

`#include <pdbTemplateArg.h>`

Public Types

- enum **targ_t** { **TA_NA**, **TA_TYPE**, **TA_NONTYPE**, **TA_TEMPL** }

Public Member Functions

- **pdbTemplateArg** (**targ_t** kind, bool specialization=false)
- const string & **name** () const
- const **pdbType** * **type** () const

Detailed Description

A class to define argument in a template defintions.

pdbTemplateArg describes arguments in template definitions and specializations.

Member Enumeration Documentation

enum pdbTemplateArg::targ_t

this enumeration tell the type of template argument Depending on the kind of the template argument, different methods are applicable.

Enumerator:

TA_NA uninitialized

TA_TYPE types (**type()** and defaultType()).

TA_NONTYPE non types (**type()**, **name()** and defaultValue()).

TA_TEMPL templates (templateArg() and defaultTemplateArg())

Constructor & Destructor Documentation

pdbTemplateArg::pdbTemplateArg (targ_t *kind*, bool *specialization* = `false`) `[inline]`

A constructor

Parameters:

kind the kind of template argument.
specialization wheather the arguements are specialized.

Member Function Documentation

const string & pdbTemplateArg::name () const `[inline]`

name of the template arguemnt.

const pdbType * pdbTemplateArg::type () const `[inline]`

the pointer to the type of the arguement.

The documentation for this class was generated from the following files:

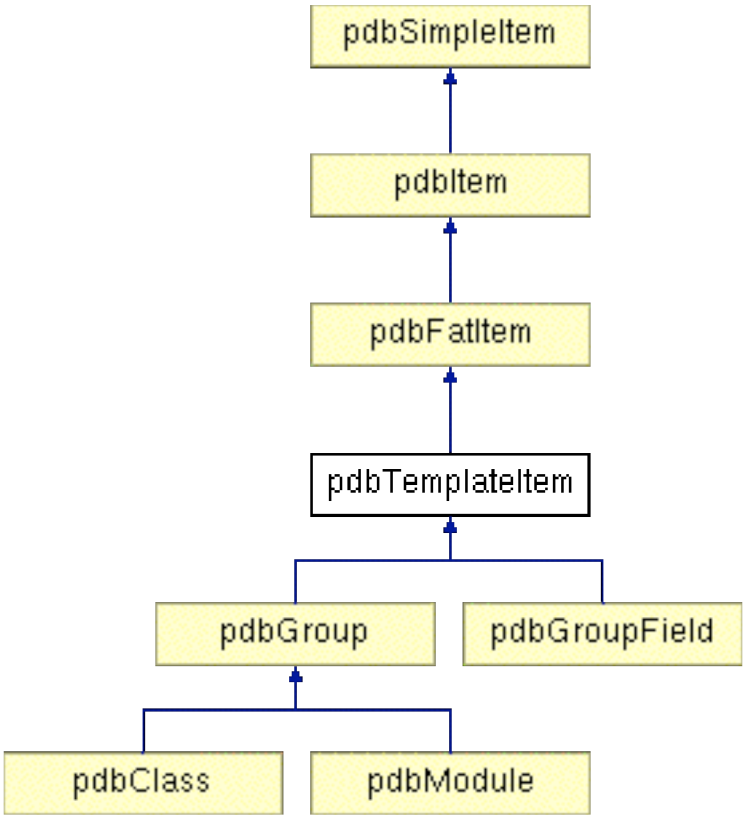
- pdbTemplateArg.h
- pdbTemplateArg.inl

pdbTemplateItem Class Reference

A class to record templates.

`#include <pdbTemplateItem.h>`

Inheritance diagram for pdbTemplateItem:



Public Member Functions

- const **pdbTemplate** * **isTemplate** () const
- bool **isSpecialized** () const
- const targvec & **speclArguments** () const

Detailed Description

A class to record templates.

pdbTemplateItems are entities that can be instantiated from templates. Template items are pdbGroups, pdbGroupFields, pdbRoutines.

Member Function Documentation

bool pdbTemplateItem::isSpecialized () const `[inline]`

Is this item Specialized?

const pdbTemplate * pdbTemplateItem::isTemplate () const `[inline]`

pointer to the pdbTemplateItem

const pdbTemplateItem::targvec & pdbTemplateItem::speclArguments () const `[inline]`

speclized Arguments

The documentation for this class was generated from the following files:

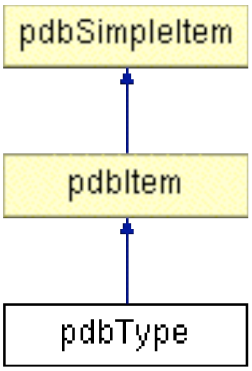
- pdbTemplateItem.h
- pdbTemplateItem.inl

pdbType Class Reference

A class to contain the abstract Type information.

`#include <pdbType.h>`

Inheritance diagram for pdbType:



Public Member Functions

- const **pdbType** * **elementType** () const
- **float_t floatType** () const
- **int_t integerType** () const
- **type_t kind** () const
- const typevec & **exceptionSpec** () const

Detailed Description

A class to contain the abstract Type information.

This class hold the information about Fortran and C/C++ types both abstract and basic types.

Member Function Documentation

const pdbType * pdbType::elementType () const `[inline]`

the abstract type of this argument.

const pdbType::typevec & pdbType::exceptionSpec () const `[inline]`

for C arrays and f90 characters

pdbItem::float_t pdbType::floatType () const `[inline]`

the float type of this argument.

pdbItem::int_t pdbType::integerType () const `[inline]`

the intger type of this argument.

pdbItem::type_t pdbType::kind () const `[inline]`

the kind of the type.

The documentation for this class was generated from the following files:

- pdbType.h
- pdbType.inl

PDT Ductape API Example Documentation

froutine.cc

Routines describes the common part of global functions, Fortran 90 local and module functions, and C++ class methods. The common atrributes are signature, kind (e.g., extern or intrinsic), specialKind (e.g., constructor or operator), a list of routines called from this routine, how often it gets called from other routines, linkage, for C and C++ the statement representing the body and a list of all statements, and for C and Fortran routines the location of the first executable statement and of all return statements.

```
#include "pdbAll.h"
#include "stdio.h"
#include <typeinfo>

int main(int argc, char *argv[]) {
  PDB p(argv[1]); if ( !p ) return 1;

        for (PDB::croutinevec::iterator r = p.getCRoutineVec().begin();
r!=p.getCRoutineVec().end(); r++)
        {
                cout << (*r)->name() << "  ";
                cout << (*r)->specialKind() << endl;

        }
  return 0;
}
```

stmt.cc

This class stores information about the statements (or block of statements) within a source file. This class also keeps track of the next statement within there own context as well as the next statement in childern contexts.

```
#include "pdbAll.h"
#include "stdio.h"
#include <typeinfo>


        int main(int argc, char *argv[]) {
        PDB p(argv[1]); if ( !p ) return 1;

                //Iterate though the C Routines
                for (PDB::croutinevec::iterator r = p.getCRoutineVec().begin();
                        r!=p.getCRoutineVec().end(); r++)
                {
                        //Retrive the statement within the routines.
                        cout << (*(*r)->signature()).name() << endl;
                        if ((*r)->kind() == 0)
                        {
                                const pdbStmt *v = (**r).body();
                                cout << typeid(*v).name() << endl;

                                //Print out the begining and ending locations of the
statement block.
                                cout << "statement begins: " << v->stmtBegin() << endl;
                                cout << "statement ends: " << v->stmtEnd() << endl;
                        }
                }
                return 0;
        }
```

vector.cc

An example to show how to iterate through the elements of a **PDB** file. classvec can be replaced with any other vector type.

```
/* File to be parsed and analyized:
class bar
{
        int foo(int v)
        {
                return v + 2;
        }
        class bar2
        {
                int routine(bool t) {return 0;}
        };
        int a;
};

 To run type:

 %> g++ -I../inc/ -o vector vector.cc ../lib/libpdb.a
 %> cxxparse testApp.cc
 %> ./vector testApp.pdb
 bar
 bar2

 */

#include "pdbAll.h"
#include "stdio.h"

        int main(int argc, char *argv[]) {

                // Read the pdb file as input for this program.
                PDB p(argv[1]); if ( !p ) return 1;

                // Iterate through each class in the pdb file a print its name.
                for (PDB::classvec::iterator r = p.getClassVec().begin();
                        r!=p.getClassVec().end(); r++)
                {
                         cout << (*r)->name() << endl;
                }
                return 0;
        }
```

Index

INDEX

Chapter 1. Tools

Table of Contents

Name

cxxparse -- Shell scripts that executes the right parsers and IL analyzers

`cxxparse` { *`C++ file`* } [`-I` *`directory`*] [`-D` *`define`*]

Description

C++ file is the source file for which a program database (PDB) file is generated. The filename of the PDB file will have the basename of the C++ file and the suffix ".pdb".

You can also specify additional flags necessary for your program to compile. The configure script will determine most, if not all, flags and incorporate these in cxxparse. Local options, such as an application include directory, can be specified here.

Options

`-I directory` Adds a directory dir to the list of directories searched for INCLUDE statements.

`-D define` A list of file that this C file defines.

Example

cxxparse example.cc

Name

cparse -- Shell scripts that executes the right parsers and IL analyzers

`cparse { C file } [ -I directory ] [ -D define ]`

Description

C file is the source file for which a program database (PDB) file is generated. The filename of the PDB file will have the basename of the C file and the suffix ".pdb".

You can also specify additional flags necessary for your program to compile. The configure script will determine most, if not all, flags and incorporate these in cparse. Local options, such as an application include directory, can be specified here.

Options

`-I directory` Adds a directory dir to the list of directories searched for INCLUDE statements.

`-D define` A list of file that this C file defines.

Example

cparse example.c

Name

f90parse -- Shell scripts that executes the right parsers and IL analyzers

`f90parse` { *`Fortran file`* } [`-F`] [`-I` *`directory`*] [`-M` *`directory`*] [`-R`] [`-r`] [`-U`] [`-u`] [`-A`] [`-Llfile`]

Description

Fortran file is the source file for which a program database (PDB) file is generated. The filename of the PDB file will have the basename of the Fortran file and the suffix ".pdb".

You can also specify additional flags necessary for your program to compile. The configure script will determine most, if not all, flags and incorporate these in cparse. Local options, such as an application include directory, can be specified here.

Options

`-F` Fixed form for source. By default, the form is free. In fixed form positions (columns) 1-5 can be used only for lables, position 6 is for continuation and a "C" or "*" is for comment lines. The main program must fall in positions 7-72.

`-I<dir>` Adds a directory dir to the list of directories searched for INCLUDE statements.

`-M<dir>` Specifies a list of directories for searching module definition files. Members of the list must be separated by semicolon. While modules are defined or used this option is mandatory.

`-R` Suppress folding constant expressions but those that either are public constant values of modules or define parameters of type.

`-r` Issue remarks, which are diagnostic messages even milder than warnings.

`-U` Case sensitivity for identifiers.

`-u` Disable implicit typing of identifiers. This has the same effect as IMPLICIT NONE statement as applied to the file.

`-A` Warn about use of non-F90 features, disable features that conflict with F90.

`-Llfile` Generate raw listing information in the file lfile. This information is used to generate a formatted listing where each line begins with a key character that identifies the type: N - Normal line S - Comment line R - Remark diagnostics W - Warning diagnostics E - Error diagnostics C - Catastrophic error diagnostics.

Note

The Fortran 90 parser included in PDT adheres very strictly to the F90 language specification and does not comply with extensions to the language typically implemented by vendors. This includes real*8 or integer*8 types, kind parameters, and some continuation fields in fixed form. In some cases, the source must be modified to comply with the standard before the PDT front-end can parse the program. f95parse may be used to parse codes that f90parse cannot handle.

Example

f90parse example.f90

Name

f95parse -- Shell scripts that executes the right parsers and IL analyzers

`f95parse` { *`Fortran file`* } [`-F`] [`-I` *`directory`*] [`-M` *`directory`*] [`-R`] [`-r`] [`-U`] [`-u`] [`-A`] [`-Llfile`] [`-o` *`pdbfile`*]

Description

Fortran file is the source file for which a program database (PDB) file is generated. The filename of the PDB file will have the basename of the Fortran file and the suffix ".pdb".

You can also specify additional flags necessary for your program to compile. The configure script will determine most, if not all, flags and incorporate these in cparse. Local options, such as an application include directory, can be specified here.

Options

`-v` Verbose flag. In this mode, all error messages and warnings are displayed.

`-R free` Specifies free form, -R fixed specifies fixed form for the Fortran source code. If your Fortran source has a .f file extension and uses free form, it is important to specify this flag. By default the parser assumes fixed form for F77. For other flags that f95parse accepts, please refer to the etc/flint.hls file.

`-p` invoke preprocessor.

`-o<pdbfile>` Specifies the name of the PDB file. Note: there is no space between -o and the file name.

Note

You may specify multiple fortran files on the command-line to resolve module dependencies. e.g.,

```
% f95parse `find . -name "*.f90" -print` -omerged.pdb
```

parses all files with .f90 suffix to produce merged.pdb file.

Currently, f95parse can produce PDB files that have enough information for use with the TAU profiling package. However, it does not have argument and calltree information that may be needed for other tools such as CHASM. This will be added in future releases.

Example

f95parse example.f95

Name

pdbconv -- Simple tool that checks the consistency/correctness of a PDB file and converts it to a more verbose, human-readable format.

`pdbconv [ -c ][ -o <outfile> ][ -A ][ -G ][ -M ][ -N ][ -P ][ -R ][ -S ][ -T ][ -Y ]{ <pdbfile> }`

Description

Called without any options, pdbconv reads the PDB file <pdbfile> checks it for correctness, and prints it out again in a standard form. The following options are available:

Options

`-c` Check for correctness only

`-o <outfile>` Write output to file >outfile<

`-A` Convert (A)ll item output to verbose format

`-G` Print only (G)roup items (in verbose format)

`-M` Print only (M)acro items (in verbose format)

`-N` Print only (N)amespace items (in verbose format)

`-P` Print only (P)ragma items (in verbose format)

`-R` Print only (R)outine items (in verbose format)

`-S` Print only (S)ource file items (in verbose format)

`-T` Print only (T)emplate items (in verbose format)

`-Y` Print only t(Y)pe items (in verbose format)

Example

```
%>pdbconv -Y testApp.pdb
TY#1 double
location:           [UNKNOWN]
kind:               c/c++Float
floatKind:          double
...
```

Name

pdbhtml -- Produces "htmlized" versions of all source and header files

`pdbhtml` { *<pdbfile>* }

Description

Produces "htmlized" versions of all source and header files contained in the program database file <pdbfile>. It also produces an HTML index of Classes, Templates, Namespaces, and Functions called "index.html".

Options

`<pdbfile>`

Name

pdbmerge -- Takes a set of program database files and merges them into one.

`pdbmerge [ -v ][ -o <outfile> ][ <pdbfiles> ]`

Description

`-v` Verbose

`-o <outfile>` Write merged database to file <outfile> instead of cout.

Note

Namespace definitions spread over several files are not merged correctly yet.

Example

```
%>pdbmerge -o new.pdb test1.pdb test2.pdb
```

Name

pdbtree -- Prints the source file inclusion tree, class hierarchy (IS-A + HAS-A), and function call graph.

`pdbtree [ -C ] [ -R ] [ -S ] { <pdbfile> }`

Description

Prints the source file inclusion tree, class hierarchy (IS-A + HAS-A), and function call graph.

Options

`-C` Print only the (C)lass hierarchy

`-R` Print only the (R)outine call graph

`-S` Print only the (S)ource file inclusion tree

Note

Class hierarchy is a DAG, not a tree, and therefore display is bad.

Example

```
%>pdbtree -R tutorial.pdb
void B::callA(A)
`--> bool A::isZero(int)
```

Name

pdbcomment -- Scans all (non-system) source files related to a PDB file for C, C++, Fortran comments, C/C++ pragmas, and Fortran directives and prints out a new enhanced PDB file containing this additional information.

`pdbcomment [ -o <outfile> ] [ -c ] [ -p ] [ -d ] [ -v ] [ -D` *`string`* `]`

Description

Scans all (non-system) source files related to a PDB file for C, C++, Fortran comments, C/C++ pragmas, and Fortran directives and prints out a new enhanced PDB file containing this additional information.

Options

`-o <outfile>` Write output to file <outfile>

`-c` Only scan for comments (ignore pragmas)

`-p` Only scan for pragmas (ignore comments)

`-d` Fortran only: Consider lines with a 'D' in the first column also as comments.

`-v` Verbose mode

`-D string` Fortran only: Scan also for directives which are marked with the sentinel 'string'. This option can be specified several times, once for each directive sentinel to scan for. Case does NOT matter when specifiying 'string'. pdbcomment recognizes OpenMP (sentinel $omp) by default.

Example

```
% pdbcomment test.pdb
<PDB 3.0>
lang c++

so#1 testApp.cc
scom co#0 c so#1 1 1 so#1 3 2 /*\n      class T\n*/
scom co#1 c++ so#1 8 3 so#1 8 19 // create float.
```

Name

pdbStmt -- Scans pdb file and prints every statement in the pdb file.

`pdbStmt` { *`pdbFile`* }

Description

Scans pdb file and prints every statement in the pdb file.

Example

```
% pdbstmt testApp.pdb
--------------- int T::foo(double, int):
st#1  (  7, 2)         :  {
st#2  (  9, 3) (  9,12):    decl
st#0  ( 10, 3) ( 10,13):    return
               ( 11, 2):  }
```

Name

xmlgen -- Converts a PDB file to an XML file.

`xmlgen[ -c ][ -s ][ -l ]`

Options

`-c` optional flag. forces generator to assume C, not C++.

`-s` optional flag. causes generator to emit statement level info.

`-l` optional flag. causes generator to emit locations with -s.

Note

In the current release, support for Fortran is mostly complete. C++ templates or multiple inheritance are not yet supported. The CHASM project [http://sourceforge.net/projects/chasm-interop] at LANL developed xmlgen.

Example

```
% xmlgen testApp.pdb
```