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
```