NAME

llvm-pdbutil - PDB File forensics and diagnostics

SYNOPSIS

llvm-pdbutil [*subcommand*] [*options*]

DESCRIPTION

Display types, symbols, CodeView records, and other information from a PDB file, as well as manipulate and create PDB files. **llvm-pdbutil** is normally used by FileCheck-based tests to test LLVM's PDB reading and writing functionality, but can also be used for general PDB file investigation and forensics, or as a replacement for cvdump.

SUBCOMMANDS

llvm-pdbutil is separated into several subcommands each tailored to a different purpose. A brief summary of each command follows, with more detail in the sections that follow.

- *pretty* - Dump symbol and type information in a format that tries to look as much like the original source code as possible.
- *dump* - Dump low level types and structures from the PDB file, including CodeView records, hash tables, PDB streams, etc.
- *bytes* - Dump data from the PDB file's streams, records, types, symbols, etc as raw bytes.
- *yaml2pdb* - Given a yaml description of a PDB file, produce a valid PDB file that matches that description.
- *pdb2yaml* - For a given PDB file, produce a YAML description of some or all of the file in a way that the PDB can be reconstructed.
- *merge* - Given two PDBs, produce a third PDB that is the result of merging the two input PDBs.

pretty

IMPORTANT:

The **pretty** subcommand is built on the Windows DIA SDK, and as such is not supported on non-Windows platforms.

USAGE: **llvm-pdbutil** pretty [*options*] <input PDB file>

Summary

The *pretty* subcommand displays a very high level representation of your program's debug info. Since it is built on the Windows DIA SDK which is the standard API that Windows tools and debuggers query debug information, it presents a more authoritative view of how a debugger is going to interpret your debug information than a mode which displays low-level CodeView records.

Options

Filtering and Sorting Options

NOTE:

exclude filters take priority over *include* filters. So if a filter matches both an include and an exclude rule, then it is excluded.

-exclude-compilands=<string>

When dumping compilands, compiland source-file contributions, or per-compiland symbols, this

option instructs **llvm-pdbutil** to omit any compilands that match the specified regular expression.

-exclude-symbols=<string>
When dumping global, public, or per-compiland symbols, this option instructs **llvm-pdbutil** to omit any symbols that match the specified regular expression.

-exclude-types=<string>
When dumping types, this option instructs **llvm-pdbutil** to omit any types that match the specified regular expression.

-include-compilands=<string>
When dumping compilands, compiland source-file contributions, or per-compiland symbols, limit the initial search to only those compilands that match the specified regular expression.

-include-symbols=<string>
When dumping global, public, or per-compiland symbols, limit the initial search to only those symbols that match the specified regular expression.

-include-types=<string>
When dumping types, limit the initial search to only those types that match the specified regular expression.

-min-class-padding=<uint>
Only display types that have at least the specified amount of alignment padding, accounting for padding in base classes and aggregate field members.

-min-class-padding-imm=<uint>
Only display types that have at least the specified amount of alignment padding, ignoring padding in base classes and aggregate field members.

-min-type-size=<uint>
Only display types T where sizeof(T) is greater than or equal to the specified amount.

-no-compiler-generated
Don’t show compiler generated types and symbols

-no-enum-definitions
When dumping an enum, don’t show the full enum (e.g. the individual enumerator values).

-no-system-libs

Don’t show symbols from system libraries

Symbol Type Options

-all Implies all other options in this category.

-class-definitions=<format>
Displays class definitions in the specified format.

=all - Display all class members including data, constants, typedefs, functions, etc (default)
=layout - Only display members that contribute to class size.
=none - Don’t display class definitions (e.g. only display the name and base list)

-class-order
Displays classes in the specified order.

=none - Undefined / no particular sort order (default)
=name - Sort classes by name
=size - Sort classes by size
=padding - Sort classes by amount of padding
=padding-pct - Sort classes by percentage of space consumed by padding
=padding-imm - Sort classes by amount of immediate padding
=padding-pct-imm - Sort classes by percentage of space consumed by immediate padding

-class-recurse-depth=<uint>
When dumping class definitions, stop after recursing the specified number of times. The default is 0, which is no limit.

-classes
Display classes

-compilands
Display compilands (e.g. object files)

-enums
Display enums

-externals
Dump external (e.g. exported) symbols

-globals
Dump global symbols

-lines
Dump the mappings between source lines and code addresses.

-module-syms
Display symbols (variables, functions, etc) for each compiland

-sym-types=<types>
Type of symbols to dump when -globals, -externals, or -module-syms is specified. (default all)

=thunks - Display thunk symbols
=data - Display data symbols
=funcs - Display function symbols
=all - Display all symbols (default)

-symbol-order=<order>
For symbols dumped via the -module-syms, -globals, or -externals options, sort the results in specified order.

=none - Undefined / no particular sort order
=name - Sort symbols by name
=size - Sort symbols by size

-typedefs
Display typedef types

-types
Display all types (implies -classes, -enums, -typedefs)

Other Options

-color-output
Force color output on or off. By default, color if used if outputting to a terminal.

-load-address=<uint>
When displaying relative virtual addresses, assume the process is loaded at the given address and display what would be the absolute address.

dump

USAGE: **llvm-pdbutil** dump [*options*] <input PDB file>

Summary

The **dump** subcommand displays low level information about the structure of a PDB file. It is used heavily by LLVM's testing infrastructure, but can also be used for PDB forensics. It serves a role similar to that of Microsoft's *cvdump* tool.

NOTE:

The **dump** subcommand exposes internal details of the file format. As such, the reader should be familiar with *The PDB File Format* before using this command.

Options

MSF Container Options

-streams

dump a summary of all of the streams in the PDB file.

-stream-blocks

In conjunction with *-streams*, add information to the output about what blocks the specified stream occupies.

-summary

Dump MSF and PDB header information.

Module & File Options

-modi=<uint>

For all options that dump information from each module/compiland, limit to the specified module.

-files

Dump the source files that contribute to each displayed module.

-il Dump inlinee line information (DEBUG_S_INLINEELINES CodeView subsection)

-l Dump line information (DEBUG_S_LINES CodeView subsection)

-modules

Dump compiland information

-xme
Dump cross module exports (DEBUG_S_CROSSSCOPEEXPORTS CodeView subsection)

-xmi
Dump cross module imports (DEBUG_S_CROSSSCOPEIMPORTS CodeView subsection)

Symbol Options

-globals
dump global symbol records

-global-extras
dump additional information about the globals, such as hash buckets and hash values.

-publics
dump public symbol records

-public-extras
dump additional information about the publics, such as hash buckets and hash values.

-symbols
dump symbols (functions, variables, etc) for each module dumped.

-sym-data
For each symbol record dumped as a result of the *-symbols* option, display the full bytes of the record in binary as well.

Type Record Options

-types
Dump CodeView type records from TPI stream

-type-extras
Dump additional information from the TPI stream, such as hashes and the type index offsets array.

-type-data
For each type record dumped, display the full bytes of the record in binary as well.

-type-index=<uint>
Only dump types with the specified type index.

-ids Dump CodeView type records from IPI stream.

-id-extras
Dump additional information from the IPI stream, such as hashes and the type index offsets array.

-id-data
For each ID record dumped, display the full bytes of the record in binary as well.

-id-index=<uint>
only dump ID records with the specified hexadecimal type index.

-dependents
When used in conjunction with *-type-index* or *-id-index*, dumps the entire dependency graph for the specified index instead of just the single record with the specified index. For example, if type index 0x4000 is a function whose return type has index 0x3000, and you specify *-dependents=0x4000*, then this would dump both records (as well as any other dependents in the tree).

Miscellaneous Options

-all Implies most other options.

-section-contribs
Dump section contributions.

-section-headers
Dump image section headers.

-section-map
Dump section map.

-string-table
Dump PDB string table.

bytes
USAGE: **llvm-pdbutil** bytes [*options*] <input PDB file>

Summary
Like the **dump** subcommand, the **bytes** subcommand displays low level information about the structure of a PDB file, but it is used for even deeper forensics. The **bytes** subcommand finds various structures

in a PDB file based on the command line options specified, and dumps them in hex. Someone working on support for emitting PDBs would use this heavily, for example, to compare one PDB against another PDB to ensure byte-for-byte compatibility. It is not enough to simply compare the bytes of an entire file, or an entire stream because it's perfectly fine for the same structure to exist at different locations in two different PDBs, and "finding" the structure is half the battle.

Options

MSF File Options

-block-range=<start[-end]>
Dump binary data from specified range of MSF file blocks.

-byte-range=<start[-end]>
Dump binary data from specified range of bytes in the file.

-fpm
Dump the MSF free page map.

-stream-data=<string>
Dump binary data from the specified streams. Format is SN[:Start][@Size]. For example, *-stream-data=7:3@12* dumps 12 bytes from stream 7, starting at offset 3 in the stream.

PDB Stream Options

-name-map
Dump bytes of PDB Name Map

DBI Stream Options

-ec Dump the edit and continue map substream of the DBI stream.

-files
Dump the file info substream of the DBI stream.

-modi
Dump the modi substream of the DBI stream.

-sc Dump section contributions substream of the DBI stream.

-sm Dump the section map from the DBI stream.

-type-server
Dump the type server map from the DBI stream.

Module Options

-mod=<uint>
Limit all options in this category to the specified module index. By default, options in this category will dump bytes from all modules.

-chunks
Dump the bytes of each module's C13 debug subsection.

-split-chunks
When specified with *-chunks*, split the C13 debug subsection into a separate chunk for each subsection type, and dump them separately.

-syms
Dump the symbol record substream from each module.

Type Record Options

-id=<uint>
Dump the record from the IPI stream with the given type index.

-type=<uint>
Dump the record from the TPI stream with the given type index.

pdb2yaml
USAGE: **llvm-pdbutil** pdb2yaml [*options*] <input PDB file>

Summary
Options
yaml2pdb
USAGE: **llvm-pdbutil** yaml2pdb [*options*] <input YAML file>

Summary
Generate a PDB file from a YAML description. The YAML syntax is not described here. Instead, use *llvm-pdbutil pdb2yaml* and examine the output for an example starting point.

Options

-pdb=<file-name>

Write the resulting PDB to the specified file.

merge

USAGE: **llvm-pdbutil** merge [*options*] <input PDB file 1> <input PDB file 2>

Summary

Merge two PDB files into a single file.

Options

-pdb=<file-name>

Write the resulting PDB to the specified file.

AUTHOR

Maintained by the LLVM Team (https://llvm.org/).

COPYRIGHT