



Navigation

• [LLVM Home](#) • | [Documentation](#) • » [Reference](#) • » [LLVM Command Guide](#) » [previous](#) • | [next](#) • | [index](#)

llvm-symbolizer – convert addresses into source code locations ¶

SYNOPSIS ¶

llvm-symbolizer [options] [addresses...]

DESCRIPTION ¶

llvm-symbolizer reads object file names and addresses from the command-line and prints corresponding source code locations to standard output.

If no address is specified on the command-line, it reads the addresses from standard input. If no object file is specified on the command-line, but addresses are, or if at any time an input value is not recognized, the input is simply echoed to the output.

A positional argument or standard input value can be preceded by "DATA" or "CODE" to indicate that the address should be symbolized as data or executable code respectively. If neither is specified, "CODE" is assumed. DATA is symbolized as address and symbol size rather than line number.

Object files can be specified together with the addresses either on standard input or as positional arguments on the command-line, following any "DATA" or "CODE" prefix.

llvm-symbolizer parses options from the environment variable LLVM_SYMBOLIZER_OPTS after parsing options from the command line. LLVM_SYMBOLIZER_OPTS is primarily useful for supplementing the command-line options when llvm-symbolizer is invoked by another program or runtime.

EXAMPLES ¶

All of the following examples use the following two source files as input. They use a mixture of C-style and C++-style linkage to illustrate how these names are printed differently (see [--demangle](#)).

```
// test.h
extern "C" inline int foz() {
    return 1234;
}
```

```
// test.cpp
#include "test.h"
int bar=42;

int foo() {
```

Documentation

- [Getting Started/Tutorials](#)
- [User Guides](#)
- [Reference](#)

Getting Involved

- [Contributing to LLVM](#)
- [Submitting Bug Reports](#)
- [Mailing Lists](#)
- [IRC](#)
- [Meetups and Social Events](#)

Additional Links

- [FAQ](#)
- [Glossary](#)
- [Publications](#)
- [Github Repository](#)

This Page

- [Show Source](#)

Quick search

```

    return bar;
}

int baz() {
    volatile int k = 42;
    return foz() + k;
}

int main() {
    return foo() + baz();
}

```

These files are built as follows:

```

$ clang -g test.cpp -o test.elf
$ clang -g -O2 test.cpp -o inlined.elf

```

Example 1 – addresses and object on command-line:

```

$ llvm-symbolizer --obj=test.elf 0x4004d0 0x400490
foz
/tmp/test.h:1:0

baz()
/tmp/test.cpp:11:0

```

Example 2 – addresses on standard input:

```

$ cat addr.txt
0x4004a0
0x400490
0x4004d0
$ llvm-symbolizer --obj=test.elf < addr.txt
main
/tmp/test.cpp:15:0

baz()
/tmp/test.cpp:11:0

foz
/tmp/./test.h:1:0

```

Example 3 – object specified with address:

```

$ llvm-symbolizer "test.elf 0x400490" "inlined.elf 0x400480"
baz()
/tmp/test.cpp:11:0

foo()
/tmp/test.cpp:8:10

$ cat addr2.txt
test.elf 0x4004a0
inlined.elf 0x400480

$ llvm-symbolizer < addr2.txt
main
/tmp/test.cpp:15:0

foo()
/tmp/test.cpp:8:10

```

Example 4 – CODE and DATA prefixes:

```

$ llvm-symbolizer --obj=test.elf "CODE 0x400490" "DATA 0x601028"
baz()
/tmp/test.cpp:11:0

bar
6295592 4

$ cat addr3.txt
CODE test.elf 0x4004a0

```

```
DATA inlined.elf 0x601028
```

```
$ llvm-symbolizer < addr3.txt  
main  
/tmp/test.cpp:15:0  
  
bar  
6295592 4
```

OPTIONS ¶

`--adjust-vma <offset>` ¶

Add the specified offset to object file addresses when performing lookups. This can be used to perform lookups as if the object were relocated by the offset.

`--basenames, -s` ¶

Strip directories when printing the file path.

`--demangle, -C` ¶

Print demangled function names, if the names are mangled (e.g. the mangled name `_Z3bazv` becomes `baz()`, whilst the non-mangled name `foz` is printed as is). Defaults to true.

`--dwp <path>` ¶

Use the specified DWP file at `<path>` for any CUs that have split DWARF debug data.

`--fallback-debug-path <path>` ¶

When a separate file contains debug data, and is referenced by a GNU debug link section, use the specified path as a basis for locating the debug data if it cannot be found relative to the object.

`--functions [<none|short|linkage>], -f` ¶

Specify the way function names are printed (omit function name, print short function name, or print full linkage name, respectively). Defaults to linkage.

`--help, -h` ¶

Show help and usage for this command.

`--help-list` ¶

Show help and usage for this command without grouping the options into categories.

`--inlining, --inlines, -i` ¶

If a source code location is in an inlined function, prints all the inlined frames. Defaults to true.

`--no-demangle` ¶

Don't print demangled function names.

`--obj <path>, --exe, -e` ¶

Path to object file to be symbolized. If `-` is specified, read the object directly from the standard input stream.

`--output-style <LLVM|GNU>` ¶

Specify the preferred output style. Defaults to LLVM. When the output style is set to GNU, the tool follows the style of GNU's `addr2line`. The differences from the LLVM style are:

- Does not print the column of a source code location.
- Does not add an empty line after the report for an address.
- Does not replace the name of an inlined function with the name of the topmost caller when inlined frames are not shown and `--use-symbol-table` is on.

```
$ llvm-symbolizer --obj=inlined.elf 0x4004be 0x400486 -p
baz() at /tmp/test.cpp:11:18
(inlined by) main at /tmp/test.cpp:15:0

foo() at /tmp/test.cpp:6:3

$ llvm-symbolizer --output-style=LLVM --obj=inlined.elf 0x4004be 0x400486 -p -i=0
main at /tmp/test.cpp:11:18

foo() at /tmp/test.cpp:6:3

$ llvm-symbolizer --output-style=GNU --obj=inlined.elf 0x4004be 0x400486 -p -i=0
baz() at /tmp/test.cpp:11
foo() at /tmp/test.cpp:6
```

`--pretty-print, -p` ¶

Print human readable output. If `--inlining` is specified, the enclosing scope is prefixed by (inlined by).

```
$ llvm-symbolizer --obj=inlined.elf 0x4004be --inlining --pretty-print
baz() at /tmp/test.cpp:11:18
(inlined by) main at /tmp/test.cpp:15:0
```

`--print-address, --addresses, -a` ¶

Print address before the source code location. Defaults to false.

```
$ llvm-symbolizer --obj=inlined.elf --print-address 0x4004be
0x4004be
baz()
/tmp/test.cpp:11:18
main
/tmp/test.cpp:15:0

$ llvm-symbolizer --obj=inlined.elf 0x4004be --pretty-print --print-address
0x4004be: baz() at /tmp/test.cpp:11:18
(inlined by) main at /tmp/test.cpp:15:0
```

`--print-source-context-lines <N>` ¶

Print N lines of source context for each symbolized address.

```
$ llvm-symbolizer --obj=test.elf 0x400490 --print-source-context-lines=2
baz()
/tmp/test.cpp:11:0
10 : volatile int k = 42;
11 >: return foz() + k;
12 : }
```

`--use-symbol-table` ¶

Prefer function names stored in symbol table to function names in debug info sections. Defaults to true.

--verbose ¶

Print verbose line and column information.

```
$ llvm-symbolizer --obj=inlined.elf --verbose 0x4004be
baz()
  Filename: /tmp/test.cpp
  Function start line: 9
  Line: 11
  Column: 18
main
  Filename: /tmp/test.cpp
  Function start line: 14
  Line: 15
  Column: 0
```

--version ¶

Print version information for the tool.

@<FILE> ¶

Read command-line options from response file <FILE>.

MACH-O SPECIFIC OPTIONS ¶

--default-arch <arch> ¶

If a binary contains object files for multiple architectures (e.g. it is a Mach-O universal binary), symbolize the object file for a given architecture. You can also specify the architecture by writing `binary_name:arch_name` in the input (see example below). If the architecture is not specified in either way, the address will not be symbolized. Defaults to empty string.

```
$ cat addr.txt
/tmp/mach_universal_binary:i386 0x1f84
/tmp/mach_universal_binary:x86_64 0x100000f24

$ llvm-symbolizer < addr.txt
_main
/tmp/source_i386.cc:8

_main
/tmp/source_x86_64.cc:8
```

--dsym-hint <path/to/file.dSYM> ¶

If the debug info for a binary isn't present in the default location, look for the debug info at the .dSYM path provided via this option. This flag can be used multiple times.

EXIT STATUS ¶

llvm-symbolizer returns 0. Other exit codes imply an internal program error.

SEE ALSO ¶

[llvm-addr2line\(1\)](#)

Navigation

• [LLVM Home](#) • | [Documentation](#) • » [Reference](#) • » [LLVM Command Guide](#) » [previous](#) • | [next](#) • | [index](#)

