



## ***Rocky Enterprise Linux 9.2 Manual Pages on command 'ftm.7'***

**C:\>man ftm.7**

FEATURE\_TEST\_MACROS(7)      Linux Programmer's Manual      FEATURE\_TEST\_MACROS(7)

### NAME

feature\_test\_macros - feature test macros

### DESCRIPTION

Feature test macros allow the programmer to control the definitions that are exposed by system header files when a program is compiled.

NOTE: In order to be effective, a feature test macro must be defined before including any header files. This can be done either in the compilation command (`cc -DMACRO=value`) or by defining the macro within the source code before including any headers. The requirement that the macro must be defined before including any header file exists because header files may freely include one another. Thus, for example, in the following lines, defining the `_GNU_SOURCE` macro may have no effect because the header `<abc.h>` itself includes `<xyz.h>` (POSIX explicitly allows this):

```
#include <abc.h>

#define _GNU_SOURCE

#include <xys.h>
```

Some feature test macros are useful for creating portable applications, by preventing nonstandard definitions from being exposed. Other macros can be used to expose nonstandard definitions that are not exposed by default.

The precise effects of each of the feature test macros described below can be ascertained by inspecting the `<features.h>` header file. Note: applications do not need to directly include `<features.h>`; indeed, doing so is actively discouraged.

See NOTES.

Specification of feature test macro requirements in manual pages

When a function requires that a feature test macro is defined, the manual page SYN?

OPSIS typically includes a note of the following form (this example from the acct(2) manual page):

```
#include <unistd.h>
```

```
int acct(const char *filename);
```

Feature Test Macro Requirements for glibc (see

feature\_test\_macros(7)):

```
acct(): _BSD_SOURCE || (_XOPEN_SOURCE && _XOPEN_SOURCE < 500)
```

The || means that in order to obtain the declaration of acct(2) from <unistd.h>, either of the following macro definitions must be made before including any header files:

```
#define _BSD_SOURCE
```

```
#define _XOPEN_SOURCE /* or any value < 500 */
```

Alternatively, equivalent definitions can be included in the compilation command:

```
cc -D_BSD_SOURCE
```

```
cc -D_XOPEN_SOURCE # Or any value < 500
```

Note that, as described below, some feature test macros are defined by default, so that it may not always be necessary to explicitly specify the feature test macro(s) shown in the SYNOPSIS.

In a few cases, manual pages use a shorthand for expressing the feature test macro requirements (this example from readahead(2)):

```
#define _GNU_SOURCE
```

```
#include <fcntl.h>
```

```
ssize_t readahead(int fd, off64_t *offset, size_t count);
```

This format is employed in cases where only a single feature test macro can be used to expose the function declaration, and that macro is not defined by default.

Feature test macros understood by glibc

The paragraphs below explain how feature test macros are handled in Linux glibc 2.x, x > 0.

First, though a summary of a few details for the impatient:

\* The macros that you most likely need to use in modern source code are

`_POSIX_C_SOURCE` (for definitions from various versions of POSIX.1),  
`_XOPEN_SOURCE` (for definitions from various versions of SUS), `_GNU_SOURCE` (for GNU and/or Linux specific stuff), and `_DEFAULT_SOURCE` (to get definitions that would normally be provided by default).

\* Certain macros are defined with default values. Thus, although one or more macros may be indicated as being required in the SYNOPSIS of a man page, it may not be necessary to define them explicitly. Full details of the defaults are given later in this man page.

\* Defining `_XOPEN_SOURCE` with a value of 600 or greater produces the same effects as defining `_POSIX_C_SOURCE` with a value of 200112L or greater. Where one sees

```
_POSIX_C_SOURCE >= 200112L
```

in the feature test macro requirements in the SYNOPSIS of a man page, it is implicit that the following has the same effect:

```
_XOPEN_SOURCE >= 600
```

\* Defining `_XOPEN_SOURCE` with a value of 700 or greater produces the same effects as defining `_POSIX_C_SOURCE` with a value of 200809L or greater. Where one sees

```
_POSIX_C_SOURCE >= 200809L
```

in the feature test macro requirements in the SYNOPSIS of a man page, it is implicit that the following has the same effect:

```
_XOPEN_SOURCE >= 700
```

Linux glibc understands the following feature test macros:

```
__STRICT_ANSI__
```

ISO Standard C. This macro is implicitly defined by `gcc(1)` when invoked with, for example, the `-std=c99` or `-ansi` flag.

```
_POSIX_C_SOURCE
```

Defining this macro causes header files to expose definitions as follows:

? The value 1 exposes definitions conforming to POSIX.1-1990 and ISO C (1990).

? The value 2 or greater additionally exposes definitions for POSIX.2-1992.

? The value 199309L or greater additionally exposes definitions for POSIX.1b (real-time extensions).

? The value 199506L or greater additionally exposes definitions for

POSIX.1c (threads).

? (Since glibc 2.3.3) The value 200112L or greater additionally exposes definitions corresponding to the POSIX.1-2001 base specification (excluding the XSI extension). This value also causes C95 (since glibc 2.12) and C99 (since glibc 2.10) features to be exposed (in other words, the equivalent of defining `_ISOC99_SOURCE`).

? (Since glibc 2.10) The value 200809L or greater additionally exposes definitions corresponding to the POSIX.1-2008 base specification (excluding the XSI extension).

## `_POSIX_SOURCE`

Defining this obsolete macro with any value is equivalent to defining `_POSIX_C_SOURCE` with the value 1.

Since this macro is obsolete, its usage is generally not documented when discussing feature test macro requirements in the man pages.

## `_XOPEN_SOURCE`

Defining this macro causes header files to expose definitions as follows:

? Defining with any value exposes definitions conforming to POSIX.1, POSIX.2, and XPG4.

? The value 500 or greater additionally exposes definitions for SUSv2 (UNIX 98).

? (Since glibc 2.2) The value 600 or greater additionally exposes definitions for SUSv3 (UNIX 03; i.e., the POSIX.1-2001 base specification plus the XSI extension) and C99 definitions.

? (Since glibc 2.10) The value 700 or greater additionally exposes definitions for SUSv4 (i.e., the POSIX.1-2008 base specification plus the XSI extension).

If `__STRICT_ANSI__` is not defined, or `_XOPEN_SOURCE` is defined with a value greater than or equal to 500 and neither `_POSIX_SOURCE` nor `_POSIX_C_SOURCE` is explicitly defined, then the following macros are implicitly defined:

? `_POSIX_SOURCE` is defined with the value 1.

? `_POSIX_C_SOURCE` is defined, according to the value of `_XOPEN_SOURCE`:

`_XOPEN_SOURCE < 500`

`_POSIX_C_SOURCE` is defined with the value 2.

500 <= \_XOPEN\_SOURCE < 600

\_POSIX\_C\_SOURCE is defined with the value 199506L.

600 <= \_XOPEN\_SOURCE < 700

\_POSIX\_C\_SOURCE is defined with the value 200112L.

700 <= \_XOPEN\_SOURCE (since glibc 2.10)

\_POSIX\_C\_SOURCE is defined with the value 200809L.

In addition, defining \_XOPEN\_SOURCE with a value of 500 or greater produces the same effects as defining \_XOPEN\_SOURCE\_EXTENDED.

#### \_XOPEN\_SOURCE\_EXTENDED

If this macro is defined, and \_XOPEN\_SOURCE is defined, then expose definitions corresponding to the XPG4v2 (SUSv1) UNIX extensions (UNIX 95).

Defining \_XOPEN\_SOURCE with a value of 500 or more also produces the same effect as defining \_XOPEN\_SOURCE\_EXTENDED. Use of \_XOPEN\_SOURCE\_EXTENDED in new source code should be avoided.

Since defining \_XOPEN\_SOURCE with a value of 500 or more has the same effect as defining \_XOPEN\_SOURCE\_EXTENDED, the latter (obsolete) feature test macro is generally not described in the SYNOPSIS in man pages.

#### \_ISOC99\_SOURCE (since glibc 2.1.3)

Exposes declarations consistent with the ISO C99 standard.

Earlier glibc 2.1.x versions recognized an equivalent macro named \_ISOC9X\_SOURCE (because the C99 standard had not then been finalized). Although the use of this macro is obsolete, glibc continues to recognize it for backward compatibility.

Defining \_ISOC99\_SOURCE also exposes ISO C (1990) Amendment 1 ("C95") definitions. (The primary change in C95 was support for international character sets.)

Invoking the C compiler with the option -std=c99 produces the same effects as defining this macro.

#### \_ISOC11\_SOURCE (since glibc 2.16)

Exposes declarations consistent with the ISO C11 standard. Defining this macro also enables C99 and C95 features (like \_ISOC99\_SOURCE).

Invoking the C compiler with the option -std=c11 produces the same effects as defining this macro.

## `_LARGEFILE64_SOURCE`

Expose definitions for the alternative API specified by the LFS (Large File Summit) as a "transitional extension" to the Single UNIX Specification. (See <http://opengroup.org/platform/lfs.html>.) The alternative API consists of a set of new objects (i.e., functions and types) whose names are suffixed with "64" (e.g., `off64_t` versus `off_t`, `lseek64()` versus `lseek()`, etc.). New programs should not employ this macro; instead `_FILE_OFFSET_BITS=64` should be employed.

## `_LARGEFILE_SOURCE`

This macro was historically used to expose certain functions (specifically `fseeko(3)` and `ftello(3)`) that address limitations of earlier APIs (`fseek(3)` and `ftell(3)`) that use long int for file offsets. This macro is implicitly defined if `_XOPEN_SOURCE` is defined with a value greater than or equal to 500. New programs should not employ this macro; defining `_XOPEN_SOURCE` as just described or defining `_FILE_OFFSET_BITS` with the value 64 is the preferred mechanism to achieve the same result.

## `_FILE_OFFSET_BITS`

Defining this macro with the value 64 automatically converts references to 32-bit functions and data types related to file I/O and filesystem operations into references to their 64-bit counterparts. This is useful for performing I/O on large files (> 2 Gigabytes) on 32-bit systems. (Defining this macro permits correctly written programs to use large files with only a recompilation being required.)

64-bit systems naturally permit file sizes greater than 2 Gigabytes, and on those systems this macro has no effect.

## `_BSD_SOURCE` (deprecated since glibc 2.20)

Defining this macro with any value causes header files to expose BSD-derived definitions.

In glibc versions up to and including 2.18, defining this macro also causes BSD definitions to be preferred in some situations where standards conflict, unless one or more of `_SVID_SOURCE`, `_POSIX_SOURCE`, `_POSIX_C_SOURCE`, `_XOPEN_SOURCE`, `_XOPEN_SOURCE_EXTENDED`, or `_GNU_SOURCE` is defined, in which case BSD definitions are disfavored. Since glibc 2.19, `_BSD_SOURCE` no

longer causes BSD definitions to be preferred in case of conflicts.

Since glibc 2.20, this macro is deprecated. It now has the same effect as defining `_DEFAULT_SOURCE`, but generates a compile-time warning (unless `_DEFAULT_SOURCE` is also defined). Use `_DEFAULT_SOURCE` instead. To allow code that requires `_BSD_SOURCE` in glibc 2.19 and earlier and `_DEFAULT_SOURCE` in glibc 2.20 and later to compile without warnings, define both `_BSD_SOURCE` and `_DEFAULT_SOURCE`.

`_SVID_SOURCE` (deprecated since glibc 2.20)

Defining this macro with any value causes header files to expose System V-derived definitions. (SVID == System V Interface Definition; see standards(7).)

Since glibc 2.20, this macro is deprecated in the same fashion as `_BSD_SOURCE`.

`_DEFAULT_SOURCE` (since glibc 2.19)

This macro can be defined to ensure that the "default" definitions are provided even when the defaults would otherwise be disabled, as happens when individual macros are explicitly defined, or the compiler is invoked in one of its "standard" modes (e.g., `cc -std=c99`). Defining `_DEFAULT_SOURCE` without defining other individual macros or invoking the compiler in one of its "standard" modes has no effect.

The "default" definitions comprise those required by POSIX.1-2008 and ISO C99, as well as various definitions originally derived from BSD and System V. On glibc 2.19 and earlier, these defaults were approximately equivalent to explicitly defining the following:

```
cc -D_BSD_SOURCE -D_SVID_SOURCE -D_POSIX_C_SOURCE=200809L
```

`_ATFILE_SOURCE` (since glibc 2.4)

Defining this macro with any value causes header files to expose declarations of a range of functions with the suffix "at"; see `openat(2)`. Since glibc 2.10, this macro is also implicitly defined if `_POSIX_C_SOURCE` is defined with a value greater than or equal to 200809L.

`_GNU_SOURCE`

Defining this macro (with any value) implicitly defines `_ATFILE_SOURCE`,

`_LARGEFILE64_SOURCE`, `_ISOC99_SOURCE`, `_XOPEN_SOURCE_EXTENDED`, `_POSIX_SOURCE`, [Page 7/13](#)

`_POSIX_C_SOURCE` with the value 200809L (200112L in glibc versions before 2.10; 199506L in glibc versions before 2.5; 199309L in glibc versions before 2.1) and `_XOPEN_SOURCE` with the value 700 (600 in glibc versions before 2.10; 500 in glibc versions before 2.2). In addition, various GNU-specific extensions are also exposed.

Since glibc 2.19, defining `_GNU_SOURCE` also has the effect of implicitly defining `_DEFAULT_SOURCE`. In glibc versions before 2.20, defining `_GNU_SOURCE` also had the effect of implicitly defining `_BSD_SOURCE` and `_SVID_SOURCE`.

## `_REENTRANT`

Historically, on various C libraries it was necessary to define this macro in all multithreaded code. (Some C libraries may still require this.) In glibc, this macro also exposed definitions of certain reentrant functions.

However, glibc has been thread-safe by default for many years; since glibc 2.3, the only effect of defining `_REENTRANT` has been to enable one or two of the same declarations that are also enabled by defining `_POSIX_C_SOURCE` with a value of 199606L or greater.

`_REENTRANT` is now obsolete. In glibc 2.25 and later, defining `_REENTRANT` is equivalent to defining `_POSIX_C_SOURCE` with the value 199606L. If a higher POSIX conformance level is selected by any other means (such as `_POSIX_C_SOURCE` itself, `_XOPEN_SOURCE`, `_DEFAULT_SOURCE`, or `_GNU_SOURCE`), then defining `_REENTRANT` has no effect.

This macro is automatically defined if one compiles with `cc -pthread`.

## `_THREAD_SAFE`

Synonym for the (deprecated) `_REENTRANT`, provided for compatibility with some other implementations.

## `_FORTIFY_SOURCE` (since glibc 2.3.4)

Defining this macro causes some lightweight checks to be performed to detect some buffer overflow errors when employing various string and memory manipulation functions (for example, `memcpy(3)`, `memset(3)`, `strcpy(3)`, `strncpy(3)`, `strncat(3)`, `strncat(3)`, `sprintf(3)`, `snprintf(3)`, `vsprintf(3)`, `vsnprintf(3)`, `gets(3)`, and wide character variants thereof).

For some functions, argument consistency is checked; for example, a check

is made that `open(2)` has been supplied with a mode argument when the specified flags include `O_CREAT`. Not all problems are detected, just some common cases.

If `_FORTIFY_SOURCE` is set to 1, with compiler optimization level 1 (`gcc -O1`) and above, checks that shouldn't change the behavior of conforming programs are performed. With `_FORTIFY_SOURCE` set to 2, some more checking is added, but some conforming programs might fail.

Some of the checks can be performed at compile time (via macros implemented in header files), and result in compiler warnings; other checks take place at run time, and result in a run-time error if the check fails.

Use of this macro requires compiler support, available with `gcc(1)` since version 4.0.

#### Default definitions, implicit definitions, and combining definitions

If no feature test macros are explicitly defined, then the following feature test macros are defined by default: `_BSD_SOURCE` (in `glibc 2.19` and earlier), `_SVID_SOURCE` (in `glibc 2.19` and earlier), `_DEFAULT_SOURCE` (since `glibc 2.19`), `_POSIX_SOURCE`, and `_POSIX_C_SOURCE=200809L` (`200112L` in `glibc` versions before `2.10`; `199506L` in `glibc` versions before `2.4`; `199309L` in `glibc` versions before `2.1`).

If any of `__STRICT_ANSI__`, `_ISOC99_SOURCE`, `_POSIX_SOURCE`, `_POSIX_C_SOURCE`, `_XOPEN_SOURCE`, `_XOPEN_SOURCE_EXTENDED`, `_BSD_SOURCE` (in `glibc 2.19` and earlier), or `_SVID_SOURCE` (in `glibc 2.19` and earlier) is explicitly defined, then `_BSD_SOURCE`, `_SVID_SOURCE`, and `_DEFAULT_SOURCE` are not defined by default.

If `_POSIX_SOURCE` and `_POSIX_C_SOURCE` are not explicitly defined, and either `__STRICT_ANSI__` is not defined or `_XOPEN_SOURCE` is defined with a value of 500 or more, then

- \* `_POSIX_SOURCE` is defined with the value 1; and

- \* `_POSIX_C_SOURCE` is defined with one of the following values:

- ? 2, if `_XOPEN_SOURCE` is defined with a value less than 500;

- ? `199506L`, if `_XOPEN_SOURCE` is defined with a value greater than or equal to 500 and less than 600; or

- ? (since `glibc 2.4`) `200112L`, if `_XOPEN_SOURCE` is defined with a value greater than or equal to 600 and less than 700.

- ? (Since `glibc 2.10`) `200809L`, if `_XOPEN_SOURCE` is defined with a value greater

than or equal to 700.

? Older versions of glibc do not know about the values 200112L and 200809L for `_POSIX_C_SOURCE`, and the setting of this macro will depend on the glibc version.

? If `_XOPEN_SOURCE` is undefined, then the setting of `_POSIX_C_SOURCE` depends on the glibc version: 199506L, in glibc versions before 2.4; 200112L, in glibc 2.4 to 2.9; and 200809L, since glibc 2.10.

Multiple macros can be defined; the results are additive.

## CONFORMING TO

POSIX.1 specifies `_POSIX_C_SOURCE`, `_POSIX_SOURCE`, and `_XOPEN_SOURCE`.

`_XOPEN_SOURCE_EXTENDED` was specified by XPG4v2 (aka SUSv1), but is not present in SUSv2 and later. `_FILE_OFFSET_BITS` is not specified by any standard, but is employed on some other implementations.

`_BSD_SOURCE`, `_SVID_SOURCE`, `_DEFAULT_SOURCE`, `_ATFILE_SOURCE`, `_GNU_SOURCE`, `_FORTIFY_SOURCE`, `_REENTRANT`, and `_THREAD_SAFE` are specific to Linux (glibc).

## NOTES

`<features.h>` is a Linux/glibc-specific header file. Other systems have an analogous file, but typically with a different name. This header file is automatically included by other header files as required: it is not necessary to explicitly include it in order to employ feature test macros.

According to which of the above feature test macros are defined, `<features.h>` internally defines various other macros that are checked by other glibc header files.

These macros have names prefixed by two underscores (e.g., `__USE_MISC`). Programs should never define these macros directly: instead, the appropriate feature test macro(s) from the list above should be employed.

## EXAMPLE

The program below can be used to explore how the various feature test macros are set depending on the glibc version and what feature test macros are explicitly set. The following shell session, on a system with glibc 2.10, shows some examples of what we would see:

```
$ cc ftm.c
```

```
$ ./a.out
```

```
  _POSIX_SOURCE defined
```

\_POSIX\_C\_SOURCE defined: 200809L

\_BSD\_SOURCE defined

\_SVID\_SOURCE defined

\_ATFILE\_SOURCE defined

\$ cc -D\_XOPEN\_SOURCE=500 ftm.c

\$ ./a.out

\_POSIX\_SOURCE defined

\_POSIX\_C\_SOURCE defined: 199506L

\_XOPEN\_SOURCE defined: 500

\$ cc -D\_GNU\_SOURCE ftm.c

\$ ./a.out

\_POSIX\_SOURCE defined

\_POSIX\_C\_SOURCE defined: 200809L

\_ISOC99\_SOURCE defined

\_XOPEN\_SOURCE defined: 700

\_XOPEN\_SOURCE\_EXTENDED defined

\_LARGEFILE64\_SOURCE defined

\_BSD\_SOURCE defined

\_SVID\_SOURCE defined

\_ATFILE\_SOURCE defined

\_GNU\_SOURCE defined

Program source

```
/* ftm.c */
```

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <stdlib.h>
```

```
int
```

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

```
{
```

```
#ifdef _POSIX_SOURCE
```

```
    printf("_POSIX_SOURCE defined\n");
```

```
#endif
```

```
#ifdef _POSIX_C_SOURCE
```

```
    printf("_POSIX_C_SOURCE defined: %ld\n", (long) _POSIX_C_SOURCE);
#endif

#ifdef _ISOC99_SOURCE
    printf("_ISOC99_SOURCE defined\n");
#endif

#ifdef _ISOC11_SOURCE
    printf("_ISOC11_SOURCE defined\n");
#endif

#ifdef _XOPEN_SOURCE
    printf("_XOPEN_SOURCE defined: %d\n", _XOPEN_SOURCE);
#endif

#ifdef _XOPEN_SOURCE_EXTENDED
    printf("_XOPEN_SOURCE_EXTENDED defined\n");
#endif

#ifdef _LARGEFILE64_SOURCE
    printf("_LARGEFILE64_SOURCE defined\n");
#endif

#ifdef _FILE_OFFSET_BITS
    printf("_FILE_OFFSET_BITS defined: %d\n", _FILE_OFFSET_BITS);
#endif

#ifdef _BSD_SOURCE
    printf("_BSD_SOURCE defined\n");
#endif

#ifdef _SVID_SOURCE
    printf("_SVID_SOURCE defined\n");
#endif

#ifdef _DEFAULT_SOURCE
    printf("_DEFAULT_SOURCE defined\n");
#endif

#ifdef _ATFILE_SOURCE
    printf("_ATFILE_SOURCE defined\n");
#endif

#ifdef _GNU_SOURCE
```

```
    printf("_GNU_SOURCE defined\n");
#endif
#ifdef _REENTRANT
    printf("_REENTRANT defined\n");
#endif
#ifdef _THREAD_SAFE
    printf("_THREAD_SAFE defined\n");
#endif
#ifdef _FORTIFY_SOURCE
    printf("_FORTIFY_SOURCE defined\n");
#endif
    exit(EXIT_SUCCESS);
}
```

#### SEE ALSO

libc(7), standards(7)

The section "Feature Test Macros" under info libc.

/usr/include/features.h

#### COLOPHON

This page is part of release 5.05 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.