man pages section 3: Extended Library Functions, Volume 2

Beta

Part No: 821–1468–06
November 2010

ORACLE

110425@25097

Contents

Preface

Both novice users and those familar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question "What does it do?" The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9E describes the DDI (Device Driver Interface)/DKI (Driver/Kernel Interface), DDI-only, and DKI-only entry-point routines a developer can include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report,

there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man(1)` for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

- [ ] — Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
- . . . — Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "`filename...`".
- | — Separator. Only one of the arguments separated by this character can be specified at a time.
- { } — Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL — This section occurs only in subsection 3R to indicate the protocol description file.

DESCRIPTION — This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.

IOCTL — This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the `ioctl(2)` system call is called `ioctl` and generates its own heading. `ioctl` calls for a specific device are listed alphabetically (on the man page for that specific device).

	`ioctl` calls are used for a particular class of devices all of which have an `io` ending, such as `mtio(7I)`.
OPTIONS	This secton lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output – standard output, standard error, or output files – generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.
USAGE	This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete

	example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.
FILES	This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
ATTRIBUTES	This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See `attributes(5)` for more information.
SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

REFERENCE

Extended Library Functions, Volume 2

Name acos, acosf, acosl – arc cosine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double acos(double x);

float acosf(float x);

long double acosl(long double x);
```

Description These functions compute the principal value of the arc cosine of *x*. The value of *x* should be in the range [−1,1].

Return Values Upon successful completion, these functions return the arc cosine of *x* in the range [0, π] radians.

For finite values of *x* not in the range [−1,1], a domain error occurs and NaN is returned.

If *x* is NaN, NaN is returned.

If *x* is +1, +0 is returned.

If *x* is ±Inf, a domain error occurs and NaN is returned.

For exceptional cases, `matherr(3M)` tabulates the values to be returned by `acos()` as specified by SVID3 and XPG3.

Errors These functions will fail if:

Domain Error — The *x* argument is finite and not in the range [-1,1], or is ±Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

The `acos()` function sets `errno` to `EDOM` if *x* is not ±Inf or NaN and is not in the range [−1,1].

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `acos()`. On return, if `errno` is non-zero, an error has occurred. The `acosf()` and `acosl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also cos(3M), feclearexcept(3M), fetestexcept(3M), isnan(3M), math.h(3HEAD), matherr(3M), attributes(5), standards(5)

Name acosh, acoshf, acoshl – inverse hyperbolic cosine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double acosh(double x);

float acoshf(float x);

long double acoshl(long double x);
```

Description These functions compute the inverse hyperbolic cosine of their argument *x*.

Return Values Upon successful completion, these functions return the inverse hyperbolic cosine of their argument.

For finite values of $x < 1$, a domain error occurs and NaN is returned.

If *x* is NaN, NaN is returned.

If *x* is +1, +0 is returned.

If *x* is +Inf, +Inf is returned.

If *x* is –Inf, a domain error occurs and NaN is returned.

For exceptional cases, `matherr(3M)` tabulates the values to be returned by `acosh()` as specified by SVID3 and XPG3.

Errors These functions will fail if:

Domain Error — The *x* argument is finite and less than 1.0, or is –Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

The `acosh()` function sets `errno` to `EDOM` if *x* is less than 1.0.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `acosh()`. On return, if `errno` is non-zero, an error has occurred. The `acoshf()` and `acoshl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also cosh(3M), feclearexcept(3M), fetestexcept(3M), math.h(3HEAD), matherr(3M), attributes(5), standards(5)

Name asin, asinf, asinl – arc sine function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double asin(double x);

float asinf(float x);

long double asinl(long double x);
```

Description These functions compute the principal value of the arc sine of their argument *x*. The value of *x* should be in the range [−1,1].

Return Values Upon successful completion, these functions return the arc sine of *x* in the range [−π/2, π/2] radians.

For finite values of *x* not in the range [−1,1], a domain error occurs and a NaN is returned.

If *x* is NaN, NaN is returned.

If *x* is ±0, *x* is returned.

If *x* is ±Inf, a domain error occurs and a NaN is returned.

For exceptional cases, `matherr(3M)` tabulates the values to be returned by `asin()` as specified by SVID3 and XPG3.

Errors These functions will fail if:

Domain Error — The *x* argument is finite and not in the range [−1,1], or is ±Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

The `asin()` function sets `errno` to `EDOM` if *x* is not ±Inf or NaN and is not in the range [−1,1].

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `asin()`. On return, if `errno` is non-zero, an error has occurred. The `asinf()` and `asinl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also isnan(3M), feclearexcept(3M), fetestexcept(3M), math.h(3HEAD), matherr(3M), sin(3M), attributes(5), standards(5)

Name asinh, asinhf, asinhl – inverse hyperbolic sine functions

Synopsis

```
cc [ flag... ] file... -lm [ library... ]
#include <math.h>

double asinh(double x);

float asinhf(float x);

long double asinhl(long double x);
```

Description These functions compute the inverse hyperbolic sine of their argument *x*.

Return Values Upon successful completion, these functions return the inverse hyperbolic sine of their argument.

If *x* is NaN, NaN is returned.

If *x* is ±0 or ±Inf, *x* is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also math.h(3HEAD), sinh(3M), attributes(5), standards(5)

Name atan2, atan2f, atan2l – arc tangent function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double atan2(double y, double x);

float atan2f(float y, float x);

long double atan2l(long double y, long double x);
```

Description These functions compute the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value.

Return Values Upon successful completion, these functions return the arc tangent of y/x in the range $[-\pi,\pi]$ radians.

If y is ± 0 and x is < 0, $\pm\pi$ is returned.

If y is ± 0 and x is > 0, ± 0 is returned.

If y is < 0 and x is ± 0, $-\pi/2$ is returned.

If y is > 0 and x is ± 0, $\pi/2$ is returned.

If x is 0, a pole error does not occur.

If either x or y is NaN, a NaN is returned.

If y is ± 0 and x is -0, $\pm\pi$ is returned.

If y is ± 0 and x is $+0$, ± 0 is returned.

For finite values of $\pm y > 0$, if x is $-$Inf, $\pm\pi$ is returned.

For finite values of $\pm y > 0$, if x is $+$Inf, ± 0 is returned.

For finite values of x, if y is $\pm$Inf, $\pm\pi/2$ is returned.

If y is $\pm$Inf and x is $-$Inf, $\pm 3\pi/4$ is returned.

If y is $\pm$Inf and x is $+$Inf, $\pm\pi/4$ is returned.

If both arguments are 0, a domain error does not occur.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also atan(3M), isnan(3M), math.h(3HEAD)tan(3M), attributes(5), standards(5)

Name atan, atanf, atanl – arc tangent function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double atan(double x);

float atanf(float x);

long double atanl(long double x);
```

Description These functions compute the principal value of the arc tangent of *x*.

Return Values Upon successful completion, these functions return the arc tangent of *x* in the range [−π/2,π/2] radians.

If *x* is NaN, NaN is returned.

If x is ±0, *x* is returned.

If *x* is ±Inf, ±π/2 is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also atan2(3M), isnan(3M), math.h(3HEAD), tan(3M), attributes(5), standards(5)

Name atanh, atanhf, atanhl – inverse hyperbolic tangent functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double atanh(double x);

float atanhf(float x);

long double atanhl(long double x);
```

Description These functions compute the inverse hyperbolic tangent of their argument *x*.

Return Values Upon successful completion, these functions return the inverse hyperbolic tangent of their argument.

If *x* is ±1, a pole error occurs and `atanh()`, `atanhf()`, and `atanhl()` return the value of the macro `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively, with the same sign as the correct value of the function.

For finite $|x| > 1$, a domain error occurs and a NaN is returned.

If *x* is NaN, NaN is returned.

If *x* is +0, *x* is returned.

If *x* is +Inf, a domain error occurs and a NaN is returned.

For exceptional cases, matherr(3M) tabulates the values to be returned by `atanh()` as specified by SVID3 and XPG3.

Errors These functions will fail if:

Domain Error — The *x* argument is finite and not in the range [-1,1], or is ±Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

The `atanh()` function sets `errno` to `EDOM` if the absolute value of *x* is greater than 1.0.

Pole Error — The *x* argument is ±1.

If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception is raised.

The `atanh()` function sets `errno` to `ERANGE` if the absolute value of *x* is equal to 1.0.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `atanh()`. On return, if `errno` is non-zero, an error has occurred. The `atanhf()` and `atanhl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `feclearexcept(3M)`, `fetestexcept(3M)`, `math.h(3HEAD)`, `matherr(3M)`, `tanh(3M)`, `attributes(5)`, `standards(5)`

Name bgets – read stream up to next delimiter

Synopsis

```
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

char *bgets(char *buffer, size_t count, FILE  *stream,
     const char *breakstring);
```

Description The `bgets()` function reads characters from *stream* into *buffer* until either *count* is exhausted or one of the characters in *breakstring* is encountered in the stream. The read data is terminated with a null byte ('`\0`') and a pointer to the trailing null is returned. If a *breakstring* character is encountered, the last non-null is the delimiter character that terminated the scan.

Note that, except for the fact that the returned value points to the end of the read string rather than to the beginning, the call

```
bgets(buffer, sizeof buffer, stream, "\n");
```

is identical to

```
fgets (buffer, sizeof buffer, stream);
```

There is always enough room reserved in the buffer for the trailing null character.

If *breakstring* is a null pointer, the value of *breakstring* from the previous call is used. If *breakstring* is null at the first call, no characters will be used to delimit the string.

Return Values `NULL` is returned on error or end-of-file. Reporting the condition is delayed to the next call if any characters were read but not yet returned.

Examples **EXAMPLE 1** Example of the `bgets()` function.

The following example prints the name of the first user encountered in `/etc/passswd`, including a trailing ":"

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

int main()
{
    char buffer[8];
    FILE *fp;

    if ((fp = fopen("/etc/passwd","r")) == NULL) {
        perror("/etc/passwd");
        return 1;
    }
    if (bgets(buffer, 8, fp, ":") == NULL) {
        perror("bgets");
        return 1;
```

EXAMPLE 1 Example of the `bgets()` function. *(Continued)*

```
    }
    (void) puts(buffer);
    return 0;
}
```

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also

`gets(3C)`, `attributes(5)`

Notes

When compiling multithread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

Name bufsplit – split buffer into fields

Synopsis

```
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

size_t bufsplit(char *buf, size_t n, char **a);
```

Description `bufsplit()` examines the buffer, *buf*, and assigns values to the pointer array, *a*, so that the pointers point to the first *n* fields in *buf* that are delimited by TABs or NEWLINEs.

To change the characters used to separate fields, call `bufsplit()` with *buf* pointing to the string of characters, and *n* and *a* set to zero. For example, to use colon (:), period (.), and comma (,), as separators along with TAB and NEWLINE:

```
bufsplit (":.,\t\n", 0, (char**)0 );
```

Return Values The number of fields assigned in the array *a*. If *buf* is zero, the return value is zero and the array is unchanged. Otherwise the value is at least one. The remainder of the elements in the array are assigned the address of the null byte at the end of the buffer.

Examples **EXAMPLE 1** Example of `bufsplit()` function.

```
/*
 * set a[0] = "This", a[1] = "is", a[2] = "a",
 * a[3] = "test"
 */
bufsplit("This\tis\ta\ttest\n", 4, a);
```

Notes `bufsplit()` changes the delimiters to null bytes in *buf*.

When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also `attributes(5)`

Name cabs, cabsf, cabsl – return a complex absolute value

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double cabs(double complex z);

float cabsf(float complex z);

long double cabsl(long double complex z);
```

Description These functions compute the complex absolute value (also called norm, modulus, or magnitude) of *z*.

Return Values These functions return the complex absolute value.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also complex.h(3HEAD), attributes(5), standards(5)

Name cacos, cacosf, cacosl – complex arc cosine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex cacos(double complex z);

float complex cacosf(float complex z);

long double complex cacosl(long double complex z);
```

Description These functions compute the complex arc cosine of *z*, with branch cuts outside the interval [-1, +1] along the real axis.

Return Values These functions return the complex arc cosine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [0, π] along the real axis.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also ccos(3M), complex.h(3HEAD), attributes(5), standards(5)

Name cacosh, cacoshf, cacoshl – complex arc hyperbolic cosine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex cacosh(double complex z);

float complex cacoshf(float complex z);

long double complex cacoshl(long double complex z);
```

Description These functions compute the complex arc hyperbolic cosine of *z*, with a branch cut at values less than 1 along the real axis.

Return Values These functions return the complex arc hyperbolic cosine value, in the range of a half-strip of non-negative values along the real axis and in the interval [$-i\pi$, $+i\pi$] along the imaginary axis.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also ccosh(3M), complex.h(3HEAD), attributes(5), standards(5)

Name carg, cargf, cargl – complex argument functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double carg(double complex z);

float cargf(float complex z);

long double cargl(long double complex z);
```

Description These functions compute the argument (also called phase angle) of z, with a branch cut along the negative real axis.

Return Values These functions return the value of the argument in the interval $[-\pi, +\pi]$.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also cimag(3M), complex.h(3HEAD), conj(3M), cproj(3M), attributes(5), standards(5)

Name casin, casinf, casinl – complex arc sine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex casin(double complex z);

float complex casinf(float complex z);

long double complex casinl(long double complex z);
```

Description These functions compute the complex arc sine of *z*, with branch cuts outside the interval [−1, +1] along the real axis.

Return Values These functions return the complex arc sine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [$-\pi/2$, $+\pi/2$] along the real axis.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also complex.h(3HEAD), csin(3M), attributes(5), standards(5)

Name casinh, casinhf, casinhl – complex arc hyperbolic sine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex casinh(double complex z);

float complex casinhf(float complex z);

long double complex casinhl(long double complex z);
```

Description These functions compute the complex arc hyperbolic sine of *z*, with branch cuts outside the interval [*-i*, *+i*] along the imaginary axis.

Return Values These functions return the complex arc hyperbolic sine value, in the range of a strip mathematically unbounded along the real axis and in the interval [$-i\pi/2$, $+i\pi/2$] along the imaginary axis.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `complex.h`(3HEAD), `csinh`(3M), `attributes(5)`, `standards(5)`

Name catan, catanf, catanl – complex arc tangent functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex catan(double complex z);

float complex catanf(float complex z);

long double complex catanl(long double complex z);
```

Description These functions compute the complex arc tangent of *z*, with branch cuts outside the interval [$-i$, $+\infty i$] along the imaginary axis.

Return Values These functions return the complex arc tangent value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [$-\pi/2$, $+\pi/2$] along the real axis.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also complex.h(3HEAD), ctan(3M), attributes(5), standards(5)

Name catanh, catanhf, catanhl – complex arc hyperbolic tangent functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex catanh(double complex z);

float complex catanhf(float complex z);

long double complex catanhl(long double complex z);
```

Description These functions compute the complex arc hyperbolic tangent of z, with branch cuts outside the interval [−1, +1] along the real axis.

Return Values These functions return the complex arc hyperbolic tangent value, in the range of a strip mathematically unbounded along the real axis and in the interval [$-i\pi/2$, $+i\pi/2$] along the imaginary axis.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also complex.h(3HEAD), ctanh(3M), attributes(5), standards(5)

Name cbrt, cbrtf, cbrtl – cube root functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double cbrt(double x);

float cbrtf(float x);

long double cbrtl(long double x);
```

Description These functions compute the real cube root of their argument *x*.

Return Values On successful completion, these functions return the cube root of *x*.

If *x* is NaN, a NaN is returned.

If *x* is ±0 or ±Inf, *x* is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also math.h(3HEAD), attributes(5), standards(5)

Name ccos, ccosf, ccosl – complex cosine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex ccos(double complex z);

float complex ccosf(float complex z);

long double complex ccosl(long double complex z);
```

Description These functions compute the complex cosine of *z*.

Return Values These functions return the complex cosine value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `cacos(3M)`, `complex.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name ccosh, ccoshf, ccoshl – complex hyperbolic cosine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex ccosh(double complex z);

float complex ccoshf(float complex z);

long double complex ccoshl(long double complex z);
```

Description These functions compute the complex hyperbolic cosine of *z*.

Return Values These functions return the complex hyperbolic cosine value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `cacosh(3M)`, `complex.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name ceil, ceilf, ceill – ceiling value function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double ceil(double x);

float ceilf(float x);

long double ceill(long double x);
```

Description These functions compute the smallest integral value not less than *x*.

Return Values Upon successful completion, the `ceil()`, `ceilf()`, and `ceill()` functions return the smallest integral value not less than *x*, expressed as a type `double`, `float`, or `long double`, respectively.

If *x* is NaN, a NaN is returned.

If *x* is ±0 or ±Inf, *x* is returned.

Usage The integral value returned by these functions need not be expressible as an `int` or `long int`. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `feclearexcept(3M)`, `fetestexcept(3M)`, `floor(3M)`, `isnan(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name cexp, cexpf, cexpl – complex exponential functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex cexp(double complex z);

float complex cexpf(float complex z);

long double complex cexpl(long double complex z);
```

Description These functions compute the complex exponent of *z*, defined as e^*z*.

Return Values These functions return the complex exponential value of *z*.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also clog(3M), complex.h(3HEAD), attributes(5), standards(5)

Name cimag, cimagf, cimagl – complex imaginary functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double cimag(double complex z);

float cimagf(float complex z);

long double cimagl(long double complex z);
```

Description These functions compute the imaginary part of *z*.

Return Values These functions return the imaginary part value (as a real).

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also carg(3M), complex.h(3HEAD), conj(3M), cproj(3M), creal(3M), attributes(5), standards(5)

Name clog, clogf, clogl – complex natural logarithm functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex clog(double complex z);

float complex clogf(float complex z);

long double complex clogl(long double complex z);
```

Description These functions compute the complex natural (base *e*) logarithm of *z*, with a branch cut along the negative real axis.

Return Values These functions return the complex natural logarithm value, in the range of a strip mathematically unbounded along the real axis and in the interval [$-i$, $+i$] along the imaginary axis.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also cexp(3M), complex.h(3HEAD), attributes(5), standards(5)

Name conj, conjf, conjl – complex conjugate functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex conj(double complex z);

float complex conjf(float complex z);

long double complex conjl(long double complex z);
```

Description These functions compute the complex conjugate of z, by reversing the sign of its imaginary part.

Return Values These functions return the complex conjugate value.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also carg(3M), cimag(3M), complex.h(3HEAD), cproj(3M), creal(3M), attributes(5), standards(5)

Name copylist – copy a file into memory

Synopsis

```
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

char *copylist(const char *filenm, off_t *szptr);
```

Description The `copylist()` function copies a list of items from a file into freshly allocated memory, replacing new-lines with null characters. It expects two arguments: a pointer *filenm* to the name of the file to be copied, and a pointer *szptr* to a variable where the size of the file will be stored.

Upon success, `copylist()` returns a pointer to the memory allocated. Otherwise it returns NULL if it has trouble finding the file, calling `malloc()`, or reading the file.

Usage The `copylist()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

Examples **EXAMPLE 1** Example of `copylist()` function.

```
/* read "file" into buf */
off_t size;
char *buf;
buf = copylist("file", &size);
if (buf) {
    for (i=0; i<size; i++)
        if (buf[i])
            putchar(buf[i]);
        else
            putchar('\n');
    }
} else {
    fprintf(stderr, "%s: Copy failed for "file".\n", argv[0]);
    exit (1);
}
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also `malloc(3C)`, `attributes(5)`, `lf64(5)`

Notes When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

Name copysign, copysignf, copysignl – number manipulation function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double copysign(double x, double y);

float copysignf(float x, float y);

long double copysignl(long double x, long double y);
```

Description These functions produce a value with the magnitude of *x* and the sign of *y*.

Return Values Upon successful completion, these functions return a value with the magnitude of *x* and the sign of *y*.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also math.h(3HEAD), signbit(3M),attributes(5), standards(5)

Name cos, cosf, cosl – cosine function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double cos(double x);

float cosf(float x);

long double cosl(long double x);
```

Description These functions compute the cosine of *x*, measured in radians.

Return Values Upon successful completion, these functions return the cosine of *x*.

If *x* is NaN, NaN is returned.

If *x* is +0, 1.0 is returned.

If *x* is ±Inf, a domain error occurs and a NaN is returned.

Errors These functions will fail if:

Domain Error — The *x* argument is ±Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `acos(3M)`, `feclearexcept(3M)`, `fetestexcept(3M)`, `isnan(3M)`, `math.h(3HEAD)`, `sin(3M)`, `tan(3M)`, `attributes(5)`, `standards(5)`

Name cosh, coshf, coshl – hyperbolic cosine function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double cosh(double x);

float coshf(float x);

long double coshl(long double x);
```

Description These functions compute the hyperbolic cosine of their argument *x*.

Return Values Upon successful completion, these functions return the hyperbolic cosine of *x*.

If the correct value would cause overflow, a range error occurs and `cosh()`, `coshf()`, and `coshl()` return the value of the macro `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.

If *x* is NaN, a NaN is returned.

If *x* is ±0, 1.0 is returned.

If *x* is ±Inf, ±Inf is returned.

For exceptional cases, `matherr(3M)` tabulates the values to be returned by `cosh()` as specified by SVID3 and XPG3.

Errors These functions will fail if:

Range Error — The result would cause an overflow.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the overflow floating-point exception is raised.

The `cosh()` function sets `errno` to `ERANGE` if the result would cause an overflow.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `cosh()`. On return, if `errno` is non-zero, an error has occurred. The `coshf()` and `coshl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also acosh(3M), feclearexcept(3M), fetestexcept(3M), isnan(3M), math.h(3HEAD), matherr(3M), sinh(3M), tanh(3M), attributes(5), standards(5)

Name cpow, cpowf, cpowl – complex power functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex cpow(double complex x, double complex y);

float complex cpowf(float complex x, float complex y);

long double complex cpowl(long double complex x,
     long double complex y);
```

Description These functions compute the complex power function *x^y*, with a branch cut for the first parameter along the negative real axis.

Return Values These functions return the complex power function value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `cabs(3M)`, `complex.h(3HEAD)`, `csqrt(3M)`, `attributes(5)`, `standards(5)`

Name cproj, cprojf, cprojl – complex projection functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex cproj(double complex z);

float complex cprojf(float complex z);

long double complex cprojl(long double complex z);
```

Description These functions compute a projection of *z* onto the Riemann sphere: *z* projects to *z*, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If *z* has an infinite part, then `cproj`(*z*) is equivalent to:

```
INFINITY + I * copysign(0.0, cimag(z))
```

Return Values These functions return the value of the projection onto the Riemann sphere.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `carg(3M)`, `cimag(3M)`, `complex.h(3HEAD)`, `conj(3M)`, `creal(3M)`, `attributes(5)`, `standards(5)`

Name creal, crealf, creall – complex real functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double creal(double complex z);

float crealf(float complex z);

long double creall(long double complex z);
```

Description These functions compute the real part of z.

Return Values These functions return the real part value.

Errors No errors are defined.

Usage For a variable *z* of complex type:

```
z == creal(z) + cimag(z)*I
```

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also carg(3M), cimag(3M), complex.h(3HEAD), conj(3M), cproj(3M), attributes(5), standards(5)

Name csin, csinf, csinl – complex sine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex csin(double complex z);

float complex csinf(float complex z);

long double complex csinl(long double complex z);
```

Description These functions compute the complex sine of *z*.

Return Values These functions return the complex sine value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `casin(3M)`, `complex.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name csinh, csinhf, csinhl – complex hyperbolic sine functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex csinh(double complex z);

float complex csinhf(float complex z);

long double complex csinhl(long double complex z);
```

Description These functions compute the complex hyperbolic sine of *z*.

Return Values These functions return the complex hyperbolic sine value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `casinh(3M)`, `complex.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name csqrt, csqrtf, csqrtl – complex square root functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex csqrt(double complex z);

float complex csqrtf(float complex z);

long double complex csqrtl(long double complex z);
```

Description These functions compute the complex square root of *z*, with a branch cut along the negative real axis.

Return Values These functions return the complex square root value, in the range of the right half-plane (including the imaginary axis).

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also cabs(3M), complex.h(3HEAD), cpow(3M), attributes(5), standards(5)

Name ctan, ctanf, ctanl – complex tangent functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex ctan(double complex z);

float complex ctanf(float complex z);

long double complex ctanl(long double complex z);
```

Description These functions compute the complex tangent of *z*.

Return Values These functions return the complex tangent value.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also catan(3M), complex.h(3HEAD), attributes(5), standards(5)

Name ctanh, ctanhf, ctanhl – complex hyperbolic tangent functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <complex.h>

double complex ctanh(double complex z);

float complex ctanhf(float complex z);

long double complex ctanhl(long double complex z);
```

Description These functions compute the complex hyperbolic tangent of *z*.

Return Values These functions return the complex hyperbolic tangent value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `catanh(3M)`, `complex.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name erf, erff, erfl – error function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double erf(double x);

float erff(float x);

long double erfl(long double x);
```

Description These functions compute the error function of their argument *x*, defined as:

$$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

Return Values Upon successful completion, these functions return the value of the error function.

If *x* is NaN, a NaN is returned.

If *x* is ±0, ±0 is returned.

If *x* is ±Inf, ±1 is returned.

If *x* is subnormal, 2/sqrt(π) * 2 is returned.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also erfc(3M), feclearexcept(3M), fetestexcept(3M), isnan(3M), math.h(3HEAD), attributes(5), standards(5)

Name erfc, erfcf, erfcl – complementary error function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double erfc(double x);

float erfcf(float x);

long double erfcl(long double x);
```

Description These function compute the complementary error function 1.0 – `erf(`*x*`)`.

Return Values Upon successful completion, these functions return the value of the complementary error function.

If *x* is NaN, a NaN is returned.

If *x* is ±0, +1 is returned.

If *x* is –Inf, +2 is returned.

If *x* is +Inf, 0 is returned.

Errors No errors are defined.

Usage The `erfc()` function is provided because of the extreme loss of relative accuracy if `erf(`*x*`)` is called for large *x* and the result subtracted from 1.0.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `erf(3M)`, `isnan(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name exp2, exp2f, exp2l – exponential base 2 functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double exp2(double x);

float exp2f(float x);

long double exp2l(long double x);
```

Description These functions compute the base-2 exponential of *x*.

Return Values Upon successful completion, these functions return 2^x.

If the correct value would cause overflow, a range error occurs and `exp2()`, `exp2f()`, and `exp2l()` return the value of the macro `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.

If *x* is NaN, a NaN is returned.

If *x* is ±0, 1 is returned.

If *x* is −Inf, +0 is returned.

If *x* is +Inf, *x* is returned.

Errors These functions will fail if:

Range Error — The result overflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the overflow floating-point exception will be raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also exp(3M), feclearexcept(3M), fetestexcept(3M), isnan(3M), log(3M), math.h(3HEAD), attributes(5), standards(5)

Name exp, expf, expl – exponential function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double exp(double x);

float expf(float x);

long double expl(long double x);
```

Description These functions compute the base-*e* exponential of *x*.

Return Values Upon successful completion, these functions return the exponential value of *x*.

If the correct value would cause overflow, a range error occurs and `exp()`, `expf()`, and `expl()` return `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.

If *x* is NaN, a NaN is returned.

If *x* is ±0, 1 is returned.

If *x* is +Inf, *x* is returned.

For exceptional cases, `matherr(3M)` tabulates the values to be returned by `exp()` as specified by SVID3 and XPG3. See `standards(5)`.

Errors These functions will fail if:

Range Error — The result overflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the overflow floating-point exception is raised.

The `exp()` function sets `errno` to `ERANGE` if the result overflows.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `exp()`. On return, if `errno` is non-zero, an error has occurred. The `expf()` and `expl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), isnan(3M), log(3M), math.h(3HEAD), matherr(3M), mp(3MP), attributes(5), standards(5)

Name expm1, expm1f, expm1l – compute exponential function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double expm1(double x);

float expm1f(float x);

long double expm1l(long double x);
```

Description These functions compute $e^x-1.0$.

Return Values Upon successful completion, these functions return $e^x-1.0$.

If x is NaN, a NaN is returned.

If x is ±0, ±0 is returned.

If x is −Inf, −1 is returned.

If x is +Inf, x is returned.

Errors These functions will fail if:

Range Error — The result overflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the overflow floating-point exception is raised.

Usage The value of `expm1(`x`)` can be more accurate than `exp(`x`)`−1.0 for small values of x.

The `expm1()` and `log1p(3M)` functions are useful for financial calculations of $((1+x)^n-1)/x$, namely:

```
expm1(n * log1p(x)) / x
```

when x is very small (for example, when performing calculations with a small daily interest rate). These functions also simplify writing accurate inverse hyperbolic functions.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also exp(3M), feclearexcept(3M), fetestexcept(3M), ilogb(3M), log1p(3M), math.h(3HEAD), attributes(5), standards(5)

Name fabs, fabsf, fabsl – absolute value function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double fabs(double x);

float fabsf(float x);

long double fabsl(long double x);
```

Description These functions compute the absolute value of x, $|x|$.

Return Values Upon successful completion, these functions return the absolute value of x.

If x is NaN, a NaN is returned.

If x is ±0, +0 is returned.

If x is ±Inf, +Inf is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also isnan(3M), math.h(3HEAD), attributes(5), standards(5)

Name fdim, fdimf, fdiml – compute positive difference between two floating-point numbers

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double fdim(double x, double y);

float fdimf(float x, float y);

long double fdiml(long double x, long double y);
```

Description These functions determine the positive difference between their arguments. If *x* is greater than *y*, *x*–*y* is returned. If *x* is less than or equal to *y*, +0 is returned.

Return Values Upon successful completion, these functions return the positive difference value.

If *x*–*y* is positive and overflows, a range error occurs and `fdim()`, `fdimf()`, and `fdiml()` returns the value of the macro `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.

If *x* or *y* is NaN, a NaN is returned.

Errors These functions will fail if:

Range Error — The result overflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the overflow floating-point exception will be raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `feclearexcept(3M)`, `fetestexcept(3M)`, `fmax(3M)`, `fmin(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name feclearexcept – clear floating-point exception

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int feclearexcept(int excepts);
```

Description The `feclearexcept()` function attempts to clear the supported floating-point exceptions represented by *excepts*.

Return Values If *excepts* is 0 or if all the specified exceptions were successfully cleared, `feclearexcept()` returns 0. Otherwise, it returns a non-zero value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `fenv.h(3HEAD)`, `fegetexceptflag(3M)`, `feraiseexcept(3M)`, `fesetexceptflag(3M)`, `fetestexcept(3M)`, `attributes(5)`, `standards(5)`

Name fegetenv, fesetenv – get and set current floating-point environment

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int fegetenv(fenv_t *envp);

int fesetenv(const fenv_t *envp);
```

Description The `fegetenv()` function attempts to store the current floating-point environment in the object pointed to by *envp*.

The `fesetenv()` function attempts to establish the floating-point environment represented by the object pointed to by *envp*. The *envp* argument points to an object set by a call to `fegetenv()` or `feholdexcept(3M)`, or equals a floating-point environment macro. The `fesetenv()` function does not raise floating-point exceptions, but only installs the state of the floating-point status flags represented through its argument.

Return Values If the representation was successfully stored, fegetenv returns 0. Otherwise, it returns a non-zero value.

If the environment was successfully established, fesetenv returns 0. Otherwise, it returns a non-zero value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `feholdexcept(3M)`, `fenv.h(3HEAD)`, `feupdateenv(3M)`, `attributes(5)`, `standards(5)`

Notes In a multithreaded program, the `fegetenv()` and `fegetenv()` functions affect the floating point environment only for the calling thread.

These functions automatically install and deinstall `SIGFPE` handlers and set and clear the trap enable mode bits in the floating point status register as needed. If a program uses these functions and attempts to install a `SIGFPE` handler or control the trap enable mode bits independently, the resulting behavior is not defined.

As described in `fex_set_handling(3M)`, when a handling function installed in `FEX_CUSTOM` mode is invoked, all exception traps are disabled (and will not be reenabled while `SIGFPE` is blocked). Thus, attempting to change the environment from within a handler by calling `fesetenv` or `feupdateenv(3M)` might not produce the expected results.

Name fegetexceptflag, fesetexceptflag – get and set floating-point status flags

Synopsis

```
cc [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int fegetexceptflag(fexcept_t *flagp, int excepts);

int fesetexceptflag(const fexcept_t *flagp, int excepts);
```

Description The `fegetexceptflag()` function attempts to store an implementation-defined representation of the states of the floating-point status flags indicated by the *excepts* argument in the object pointed to by the *flagp* argument.

The `fesetexceptflag()` function attempts to set the floating-point status flags indicated by the *excepts* argument to the states stored in the object pointed to by *flagp*. The value pointed to by *flagp* will have been set by a previous call to `fegetexceptflag()` whose second argument represented at least those floating-point exceptions represented by the *excepts* argument. This function does not raise floating-point exceptions but only sets the state of the flags.

Return Values If the representation was successfully stored, `fegetexceptflag()` returns 0. Otherwise, it returns a non-zero value.

If the excepts argument is 0 or if all the specified exceptions were successfully set, `fesetexceptflag()` returns 0. Otherwise, it returns a non-zero value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `fenv.h(3HEAD)`, `feclearexcept(3M)`, `feraiseexcept(3M)`, `fesetexceptflag(3M)`, `attributes(5)`, `standards(5)`

Name fegetround, fesetround – get and set current rounding direction

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int fegetround(void);

int fesetround(int round);
```

Description The fegetround function gets the current rounding direction.

The fesetround function establishes the rounding direction represented by its argument round. If the argument is not equal to the value of a rounding direction macro, the rounding direction is not changed.

Return Values The fegetround function returns the value of the rounding direction macro representing the current rounding direction, or a negative value if there is no such rounding direction macro or the current rounding direction is not determinable.

The fesetround function returns a 0 value if and only if the requested rounding direction was established.

Errors No errors are defined.

Examples The following example saves, sets, and restores the rounding direction, reporting an error and aborting if setting the rounding direction fails:

EXAMPLE 1 Save, set, and restore the rounding direction.

```
#include <fenv.h>
#include <assert.h>
void f(int round_dir)
{
      #pragma STDC FENV_ACCESS ON
      int save_round;
      int setround_ok;
      save_round = fegetround();
      setround_ok = fesetround(round_dir);
      assert(setround_ok == 0);
      /* ... */
      fesetround(save_round);
      /* ... */
}
```

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `fenv.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name feholdexcept – save current floating-point environment

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int feholdexcept(fenv_t *envp);
```

Description The `feholdexcept()` function saves the current floating-point environment in the object pointed to by *envp*, clears the floating-point status flags, and then installs a non-stop (continue on floating-point exceptions) mode, if available, for all floating-point exceptions.

Return Values The `feholdexcept()` function returns 0 if and only if non-stop floating-point exception handling was successfully installed.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `fegetenv(3M)`, `fenv.h(3HEAD)`, `feupdateenv(3M)`, `attributes(5)`, `standards(5)`

Notes In a multithreaded program, the `feholdexcept()` function affects the floating point environment only for the calling thread.

The `feholdexcept()` function automatically installs and deinstalls `SIGFPE` handlers and sets and clears the trap enable mode bits in the floating point status register as needed. If a program uses these functions and attempts to install a `SIGFPE` handler or control the trap enable mode bits independently, the resulting behavior is not defined.

Name feraiseexcept – raise floating-point exception

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int feraiseexcept(int excepts);
```

Description The `feraiseexcept()` function attempts to raise the supported floating-point exceptions represented by the *excepts* argument. The order in which these floating-point exceptions are raised is unspecified.

Return Values If *excepts* is 0 or if all the specified exceptions were successfully raised, `feraiseexcept()` returns 0. Otherwise, it returns a non-zero value.

Errors No errors are defined.

Usage The effect is intended to be similar to that of floating-point exceptions raised by arithmetic operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `feclearexcept(3M)`, `fegetexceptflag(3M)`, `fenv.h(3HEAD)`, `fetestexcept(3M)`, `attributes(5)`, `standards(5)`

Name fesetprec, fegetprec – control floating point rounding precision modes

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int fesetprec(int prec);

int fegetprec(void);
```

Description The IEEE 754 standard defines rounding precision modes for systems that always deliver intermediate results to destinations in extended double precision format. These modes allow such systems to deliver correctly rounded single and double precision results (in the absence of underflow and overflow) with only one rounding.

The `fesetprec()` function sets the current rounding precision to the precision specified by *prec*, which must be one of the following values defined in `<fenv.h>`:

`FE_FLTPREC` round to single precision

`FE_DBLPREC` round to double precision

`FE_LDBLPREC` round to extended double precision

The default rounding precision when a program starts is `FE_LDBLPREC`.

The `fegetprec()` function returns the current rounding precision.

Return Values The `fesetprec()` function returns a non-zero value if the requested rounding precision is established and 0 otherwise.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	Intel (see below)
Availability	system/library/math
Interface Stability	Committed
MT-Level	MT-Safe

These functions are not available on SPARC systems because SPARC processors deliver intermediate results to destinations in single or double format as determined by each floating point instruction.

See Also `fegetenv(3M)`, `fesetround(3M)`, `attributes(5)`

Numerical Computation Guide

Name fetestexcept – test floating-point exception flags

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int fetestexcept(int excepts);
```

Description The `fetestexcept()` function determines which of a specified subset of the floating-point exception flags are currently set. The *excepts* argument specifies the floating-point status flags to be queried.

Return Values The `fetestexcept()` function returns the value of the bitwise-inclusive OR of the floating-point exception macros corresponding to the currently set floating-point exceptions included in *excepts*.

Errors No errors are defined.

Examples **EXAMPLE 1** Example using `fetestexcept()`

The following example calls function *f*( ) if an invalid exception is set, and then function *g*( ) if an overflow exception is set:

```
#include <fenv.h>
/* ... */
{
#     pragma STDC FENV_ACCESS ON
      int set_excepts;
      feclearexcept(FE_INVALID | FE_OVERFLOW);
      // maybe raise exceptions
      set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
      if (set_excepts & FE_INVALID) f();
      if (set_excepts & FE_OVERFLOW) g();
      /* ... */
}
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `feclearexcept(3M)`, `fegetexceptflag(3M)`, `fenv.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name feupdateenv – update floating-point environment

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int feupdateenv(const fenv_t *envp);
```

Description The `feupdateenv()` function attempts to save the currently raised floating-point exceptions in its automatic storage, attempts to install the floating-point environment represented by the object pointed to by *envp*, and then attempts to raise the saved floating-point exceptions. The *envp* argument points to an object set by a call to `fegetenv(3M)` or `feholdexcept(3M)`, or equals a floating-point environment macro.

Return Values The `feupdateenv()` function returns 0 if and only if all the required actions were successfully carried out.

Errors No errors are defined.

Examples The following example demonstrates sample code to hide spurious underflow floating-point exceptions:

EXAMPLE 1 Hide spurious underflow floating-point exceptions.

```
#include <fenv.h>
double f(double x)
{
#   pragma STDC FENV_ACCESS ON
    double result;
    fenv_t save_env;
    feholdexcept(&save_env);
    // compute result
    if (/* test spurious underflow */)
    feclearexcept(FE_UNDERFLOW);
    feupdateenv(&save_env);
    return result;
}
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also fegetenv(3M), feholdexcept(3M), fenv.h(3HEAD), attributes(5), standards(5)

Notes In a multithreaded program, the feupdateenv() function affects the floating point environment only for the calling thread.

When the FEX_CUSTOM handling mode is in effect for an exception, raising that exception using feupdateenv() causes the handling function to be invoked. The handling function can then modify the exception flags to be set as described in fex_set_handling(3M). Any result value the handler supplies will be ignored.

The feupdateenv() function automatically installs and deinstalls SIGFPE handlers and sets and clears the trap enable mode bits in the floating point status register as needed. If a program uses these functions and attempts to install a SIGFPE handler or control the trap enable mode bits independently, the resulting behavior is not defined.

As described in fex_set_handling(3M), when a handling function installed in FEX_CUSTOM mode is invoked, all exception traps are disabled (and will not be reenabled while SIGFPE is blocked). Thus, attempting to change the environment from within a handler by calling fesetenv(3M) or feupdateenv might not produce the expected results.

Name fex_merge_flags – manage the floating point environment

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

void fex_merge_flags(const fenv_t *envp);
```

Description The `fex_merge_flags()` function copies into the current environment those exception flags that are set in the environment represented by the object pointed to by *envp*. The argument *envp* must point to an object set by a call to `feholdexcept(3M)` or `fegetenv(3M)` or equal to the macro `FE_DFL_ENV`. The `fex_merge_flags()` function does not raise any exceptions, but only sets its flags.

Return Values The `fex_merge_flags` function does not return a value.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	system/library/math, SUNWlmsx
Interface Stability	Committed
MT-Level	MT-Safe

See Also `feclearexcept(3M)`, `fegetenv(3M)`, `fesetround(3M)`, `fesetprec(3M)`, `fex_set_handling(3M)`, `fex_set_log(3M)`, `attributes(5)`

Numerical Computation Guide

Notes In a multithreaded program, the `fex_merge_flags()` function affects the floating point environment only for the calling thread.

The `fex_merge_flags()` function automatically installs and deinstalls `SIGFPE` handlers and sets and clears the trap enable mode bits in the floating point status register as needed. If a program uses these functions and attempts to install a `SIGFPE` handler or control the trap enable mode bits independently, the resulting behavior is not defined.

Name fex_set_handling, fex_get_handling, fex_getexcepthandler, fex_setexcepthandler – control floating point exception handling modes

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int fex_set_handling(int ex, int mode, void(*handler);

int fex_get_handling(int ex);

void fex_getexcepthandler(fex_handler_t *buf, int ex);

void fex_setexcepthandler(const fex_handler_t *buf, int ex);
```

Description These functions provide control of floating point exception handling modes. For each function, the *ex* argument specifies one or more exceptions indicated by a bitwise-OR of any of the following values defined in <fenv.h>:

FEX_INEXACT	
FEX_UNDERFLOW	
FEX_OVERFLOW	
FEX_DIVBYZERO	division by zero
FEX_INV_ZDZ	0/0 invalid operation
FEX_INV_IDI	infinity/infinity invalid operation
FEX_INV_ISI	infinity–infinity invalid operation
FEX_INV_ZMI	0*infinity invalid operation
FEX_INV_SQRT	square root of negative operand
FEX_INV_SNAN	signaling NaN
FEX_INV_INT	invalid integer conversion
FEX_INV_CMP	invalid comparison

For convenience, the following combinations of values are also defined:

FEX_NONE	no exceptions
FEX_INVALID	all invalid operation exceptions
FEX_COMMON	overflow, division by zero, and invalid operation
FEX_ALL	all exceptions

The fex_set_handling() function establishes the specified *mode* for handling the floating point exceptions identified by *ex*. The selected *mode* determines the action to be taken when one of the indicated exceptions occurs. It must be one of the following values:

FEX_NOHANDLER	Trap but do not otherwise handle the exception, evoking instead whatever ambient behavior would normally be in effect. This is the default behavior when the exception's trap is enabled. The *handler* parameter is ignored.
FEX_NONSTOP	Provide the IEEE 754 default result for the operation that caused the exception, set the exception's flag, and continue execution. This is the default behavior when the exception's trap is disabled. The *handler* parameter is ignored.
FEX_ABORT	Call abort(3C). The *handler* parameter is ignored.
FEX_SIGNAL	Invoke the function **handler* with the parameters normally supplied to a signal handler installed with sigfpe(3C).
FEX_CUSTOM	Invoke the function **handler* as described in the next paragraph.

In FEX_CUSTOM mode, when a floating point exception occurs, the handler function is invoked as though its prototype were:

```
#include <fenv.h>
void handler(int ex, fex_info_t *info);
```

On entry, *ex* is the value (of the first twelve listed above) corresponding to the exception that occurred, info->op indicates the operation that caused the exception, info->op1 and info->op2 contain the values of the operands, info->res contains the default untrapped result value, and info->flags reflects the exception flags that the operation would have set had it not been trapped. If the handler returns, the value contained in info->res on exit is substituted for the result of the operation, the flags indicated by info->flags are set, and execution resumes at the point where the exception occurred. The handler might modify info->res and info->flags to supply any desired result value and flags. Alternatively, if the exception is underflow or overflow, the hander might set

info->res.type = fex_nodata;

which causes the exponent-adjusted result specified by IEEE 754 to be substituted. If the handler does not modify info->res or info->flags, the effect is the same as if the exception had not been trapped.

Although the default untrapped result of an exceptional operation is always available to a FEX_CUSTOM handler, in some cases, one or both operands may not be. In these cases, the handler may be invoked with info->op1.type == fex_nodata or info->op2.type == fex_nodata to indicate that the respective data structures do not contain valid data. (For example, info->op2.type == fex_nodata if the exceptional operation is a unary operation.) Before accessing the operand values, a custom handler should always examine the type field of the operand data structures to ensure that they contain valid data in the appropriate format.

The `fex_get_handling()` function returns the current handling mode for the exception specified by *ex*, which must be one of the first twelve exceptions listed above.

The `fex_getexcepthandler()` function saves the current handling modes and associated data for the exceptions specified by *ex* in the data structure pointed to by *buf*. The type `fex_handler_t` is defined in <`fenv.h`>.

The `fex_setexcepthandler()` function restores the handling modes and associated data for the exceptions specified by *ex* from the data structure pointed to by *buf*. This data structure must have been set by a previous call to `fex_getexcepthandler()`. Otherwise the effect on the indicated modes is undefined.

Return Values The `fex_set_handling()` function returns a non-zero value if the requested exception handling mode is established. Otherwise, it returns 0.

Examples The following example demonstrates how to substitute a predetermined value for the result of a 0/0 invalid operation.

```
#include <math.h>
#include <fenv.h>

double k;

void presub(int ex, fex_info_t *info) {
    info->res.type = fex_double;
    info->res.val.d = k;
}

int main() {
    double x, w;
    int i;
    fex_handler_t buf;
/*
 * save current 0/0 handler
 */
    (void) fex_getexcepthandler(&buf, FEX_INV_ZDZ);
/*
 * set up presubstitution handler for 0/0
 */
    (void) fex_set_handling(FEX_INV_ZDZ, FEX_CUSTOM, presub);
/*
 *  compute (k*x)/sin(x) for k=2.0, x=0.5, 0.4, ..., 0.1, 0.0
 */
    k = 2.0;
    (void) printf("Evaluating f(x) = (k*x)/sin(x)\n\n");
    for (i = 5; i >= 0; i--) {
            x = (double) i * 0.1;
```

```
            w = (k * x) / sin(x);
            (void) printf("\tx=%3.3f\t f(x) = % 1.20e\n", x, w);
    }
/*
 * restore old 0/0 handler
 */
    (void) fex_setexcepthandler(&buf, FEX_INV_ZDZ);
    return 0;
}
```

The output from the preceding program reads:

```
Evaluating f(x) = (k*x)/sin(x)

    x=0.500  f(x) =  2.08582964293348816000e+00
    x=0.400  f(x) =  2.05434596443822626000e+00
    x=0.300  f(x) =  2.03031801709447368000e+00
    x=0.200  f(x) =  2.01339581906893761000e+00
    x=0.100  f(x) =  2.00333722632695554000e+00
    x=0.000  f(x) =  2.00000000000000000000e+00
```

When $x = 0$, $f(x)$ is computed as 0/0 and an invalid operation exception occurs. In this example, the value 2.0 is substituted for the result.

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	system/library/math, SUNWlmxs
Interface Stability	Committed
MT-Level	MT-Safe (see Notes)

See Also

`sigfpe(3C)`, `feclearexcept(3M)`, `fegetenv(3M)`, `fex_set_log(3M)`, `attributes(5)`

Numerical Computation Guide

Notes

In a multithreaded application, the preceding functions affect exception handling modes only for the calling thread.

The functions described on this page automatically install and deinstall `SIGFPE` handlers and set and clear the trap enable mode bits in the floating point status register as needed. If a program uses these functions and attempts to install a `SIGFPE` handler or control the trap enable mode bits independently, the resulting behavior is not defined.

All traps are disabled before a handler installed in `FEX_CUSTOM` mode is invoked. When the `SIGFPE` signal is blocked, as it is when such a handler is invoked, the floating point environment, exception flags, and retrospective diagnostic functions described in

feclearexcept(3M), fegetenv(3M), and fex_set_log(3M) do not re-enable traps. Thus, the handler itself always runs in FEX_NONSTOP mode with logging of retrospective diagnostics disabled. Attempting to change these modes within the handler may not produce the expected results.

Name fex_set_log, fex_get_log, fex_set_log_depth, fex_get_log_depth, fex_log_entry – log retrospective diagnostics for floating point exceptions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <fenv.h>

int fex_set_log(FILE *fp);

FILE *fex_get_log(void);

int fex_set_log_depth(int depth);

int fex_get_log_depth(void);

void fex_log_entry(const char *msg);
```

Description The `fex_set_log()` function enables logging of retrospective diagnostic messages regarding floating point exceptions to the file specified by *fp*. If *fp* is `NULL`, logging is disabled. When a program starts, logging is initially disabled.

The occurrence of any of the twelve exceptions listed in `fex_set_handling(3M)` constitutes an event that can be logged. To prevent the log from becoming exhorbitantly long, the logging mechanism eliminates redundant entries by two methods. First, each exception is associated with a *site* in the program. The site is identified by the address of the instruction that caused the exception together with a stack trace. Only the first exception of a given type to occur at a given site will be logged. Second, when `FEX_NONSTOP` handling mode is in effect for some exception, only those occurrences of that exception that set its previously clear flag are logged. Clearing a flag using `feclearexcept()` allows the next occurrence of the exception to be logged provided it does not occur at a site at which it was previously logged.

Each of the different types of invalid operation exceptions can be logged at the same site. Because all invalid operation exceptions share the same flag, however, of those types for which `FEX_NONSTOP` mode is in effect, only the first exception to set the flag will be logged. When the invalid operation exception is raised by a call to `feraiseexcept(3M)` or `feupdateenv(3M)`, which type of invalid operation is logged depends on the implementation.

If an exception results in the creation of a log entry, the entry is created at the time the exception occurs and before any exception handling actions selected with `fex_set_handling()` are taken. In particular, the log entry is available even if the program terminates as a result of the exception. The log entry shows the type of exception, the address of the instruction that caused it, how it will be handled, and the stack trace. If symbols are available, the address of the excepting instruction and the addresses in the stack trace are followed by the names of the corresponding symbols.

The `fex_get_log()` function returns the current log file.

The `fex_set_log_depth()` sets the maximum depth of the stack trace recorded with each exception to *depth* stack frames. The default depth is 100.

The`fex_get_log_depth()` function returns the current maximum stack trace depth.

The `fex_log_entry()` function adds a user-supplied entry to the log. The entry includes the string pointed to by *msg* and the stack trace. Like entries for floating point exceptions, redundant user-supplied entries are eliminated: only the first user-supplied entry with a given *msg* to be requested from a given site will be logged. For the purpose of a user-supplied entry, the site is defined only by the stack trace, which begins with the function that called `fex_log_entry()`.

Return Values The `fex_set_log()` function returns a non-zero value if logging is enabled or disabled accordingly and returns 0 otherwise. The `fex_set_log_depth()` returns a non-zero value if the requested stack trace depth is established (regardless of whether logging is enabled) and returns 0 otherwise.

Examples The following example demonstrates the output generated when a floating point overflow occurs in `sscanf(3C)`.

```
#include <fenv.h>

int
main() {
    double x;
/*
 * enable logging of retrospective diagnostics
 */
    (void) fex_set_log(stdout);
/*
 * establish default handling for overflows
 */
    (void) fex_set_handling(FEX_OVERFLOW, FEX_NONSTOP, NULL);
/*
 * trigger an overflow in sscanf
 */
    (void) sscanf("1.0e+400", "%lf", &x);
    return 0;
}
```

The output from the preceding program reads:

```
Floating point overflow at 0xef71cac4 __base_conversion_set_exceptio
n, nonstop mode
   0xef71cacc  __base_conversion_set_exception
   0xef721820  _decimal_to_double
   0xef75aba8  number
   0xef75a94c  __doscan_u
   0xef75ecf8  sscanf
   0x00010f20  main
```

Recompiling the program or running it on another system can produce different text addresses from those shown above.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	system/library/math, SUNWlmxs
Interface Stability	Committed
MT-Level	MT-Safe (see NOTES)

See Also feclearexcept(3M), fegetenv(3M), feraiseexcept(3M), feupdateenv(3M), fex_set_handling(3M), attributes(5)

Numerical Computation Guide

Notes All threads in a process share the same log file. Each call to `fex_set_log()` preempts the previous one.

In addition to the log file itself, two additional file descriptors are used during the creation of a log entry in order to obtain symbol names from the executable and any shared objects it uses. These file descriptors are relinquished once the log entry is written. If the file descriptors cannot be allocated, symbols names are omitted from the stack trace.

The functions described on this page automatically install and deinstall `SIGFPE` handlers and set and clear the trap enable mode bits in the floating point status register as needed. If a program uses these functions and attempts to install a `SIGFPE` handler or control the trap enable mode bits independently, the resulting behavior is not defined.

As described in `fex_set_handling()`, when a handling function installed in `FEX_CUSTOM` mode is invoked, all exception traps are disabled (and will not be reenabled while `SIGFPE` is blocked). Thus, retrospective diagnostic messages are not logged for exceptions that occur within such a handler.

Name floor, floorf, floorl – floor function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double floor(double x);

float floorf(float x);

long double floorl(long double x);
```

Description These functions compute the largest integral value not greater than *x*.

Return Values Upon successful completion, these functions return the largest integral value not greater than *x*, expressed as a `double`, `float`, or `long double`, as appropriate for the return type of the function.

If *x* is NaN, a NaN is returned.

If *x* is ±Inf or ±0, *x* is returned.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `ceil(3M)`, `feclearexcept(3M)`, `fetestexcept(3M)`, `isnan(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name fma, fmaf, fmal – floating-point multiply-add

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double fma(double x, double y, double z);

float fmaf(float x, float y, float z);

long double fmal(long double x, long double y, long double z);
```

Description These functions compute $(x * y) + z$, rounded as one ternary operation. They compute the value (as if) to infinite precision and round once to the result format, according to the rounding mode characterized by the value of `FLT_ROUNDS`.

Return Values Upon successful completion, these functions return $(x * y) + z$, rounded as one ternary operation.

If x or y are NaN, a NaN is returned.

If x multiplied by y is an exact infinity and z is also an infinity but with the opposite sign, a domain error occurs and a NaN is returned.

If one of x and y is infinite, the other is 0, and z is not a NaN, a domain error occurs and a NaN is returned.

If $x*y$ is not 0*Inf nor Inf*0 and z is a NaN, a NaN is returned.

Errors These functions will fail if:

Domain Error — The value of $x*y+z$ is invalid or the value $x*y$ is invalid.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception will be raised.

Range Error — The result overflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the overflow floating-point exception will be raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), math.h(3HEAD), attributes(5), standards(5)

Name fmax, fmaxf, fmaxl – determine maximum numeric value of two floating-point numbers

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double fmax(double x, double y);

float fmaxf(float x, float y);

long double fmaxl(long double x, long double y);
```

Description These functions determine the maximum numeric value of their arguments. NaN arguments are treated as missing data: if one argument is a NaN and the other numeric, these functions choose the numeric value.

Return Values Upon successful completion, these functions return the maximum numeric value of their arguments.

If just one argument is a NaN, the other argument is returned.

If *x* and *y* are NaN, a NaN is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also fdim(3M), fmin(3M), math.h(3HEAD), attributes(5), standards(5)

Name fmin, fminf, fminl – determine minimum numeric value of two floating-point numbers

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double fmin(double x, double y);

float fminf(float float x, float y);

long double fminl(long double x, long double y);
```

Description These functions determine the minimum numeric value of their arguments. NaN arguments are treated as missing data: if one argument is a NaN and the other numeric, these functions choose the numeric value.

Return Values Upon successful completion, these functions return the minimum numeric value of their arguments.

If just one argument is a NaN, the other argument is returned.

If *x* and *y* are NaN, a NaN is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also fdim(3M), fmax(3M), math.h(3HEAD), attributes(5), standards(5)

Name fmod, fmodf, fmodl – floating-point remainder value function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double fmod(double x, double y);

float fmodf(float x, float y);

long double fmodl(long double x, long double y);
```

Description These functions return the floating-point remainder of the division of *x* by *y*.

Return Values These functions return the value $x - i * y$, for some integer *i* such that, if *y* is non-zero, the result has the same sign as *x* and magnitude less than the magnitude of *y*.

If *x* or *y* is NaN, a NaN is returned.

If *y* is 0, a domain error occurs and a NaN is returned.

If *x* is infinite, a domain error occurs and a NaN is returned.

If *x* is ±0 and *y* is not 0, ±0 is returned.

If *x* is not infinite and *y* is ±Inf, *x* is returned.

Errors These functions will fail if:

Domain Error — The *x* argument is infinite or *y* is 0.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also feclearexcept(3M), fetestexcept(3M), isnan(3M), math.h(3HEAD), attributes(5), standards(5)

Name fpclassify – classify real floating type

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int fpclassify(real-floating x);
```

Description The `fpclassify()` macro classifies its argument value as NaN, infinite, normal, subnormal, or zero. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then classification is based on the type of the argument.

Return Values The `fpclassify()` macro returns the value of the number classification macro appropriate to the value of its argument.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `isfinite(3M)`, `isinf(3M)`, `isnan(3M)`, `isnormal(3M)`, `math.h(3HEAD)`, `signbit(3M)`, `attributes(5)`, `standards(5)`

Name frexp, frexpf, frexpl – extract mantissa and exponent from a floating-point number

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double frexp(double num, int *exp);

float frexpf(float num, int *exp);

long double frexpl(long double num, int *exp);
```

Description These functions break a floating-point number into a normalized fraction and an integral power of 2. They store the integer exponent in the `int` object pointed to by *exp*.

Return Values For finite arguments, these functions return the value *x*, such that *x* is a `double` with magnitude in the interval [½, 1) or 0, and *num* equals *x* times 2 raised to the power **exp*.

If *num* is NaN, NaN is returned and the value of **exp* is unspecified.

If *num* is ± 0, ± 0 is returned and the value of **exp* is 0.

If *num* is ±Inf, *num* is returned and the value of **exp* is unspecified.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `isnan(3M)`, `ldexp(3M)`, `modf(3M)`, `attributes(5)`, `standards(5)`

Name gmatch – shell global pattern matching

Synopsis

```
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

int gmatch(const char *str, const char *pattern);
```

Description `gmatch()` checks whether the null-terminated string *str* matches the null-terminated pattern string *pattern*. See the sh(1), section `File Name Generation`, for a discussion of pattern matching. A backslash (\) is used as an escape character in pattern strings.

Return Values `gmatch()` returns non-zero if the pattern matches the string, zero if the pattern does not.

Examples **EXAMPLE 1** Examples of `gmatch()` function.

In the following example, `gmatch()` returns non-zero (true) for all strings with "`a`" or "`-`" as their last character.

```
char *s;
gmatch (s, "*[a\-]" )
```

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also sh(1), attributes(5)

Notes When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

Name HBA_GetAdapterAttributes – retrieve attributes about a specific HBA

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_GetAdapterAttributes(HBA_HANDLE handle,
     HBA_ADAPTERATTRIBUTES *hbaattributes);
```

Parameters

handle an open handle returned from HBA_OpenAdapter(3HBAAPI)

hbaattributes a pointer to an HBA_ADAPTERATTRIBUTES structure. Upon successful completion, this structure contains the specified adapter attributes.

Description The HBA_GetAdapterAttributes() function retrieves the adapter attributes structure for a given HBA. The caller is responsible for allocating *hbaattributes*.

Return Values Upon successful completion, HBA_STATUS_OK is returned. Otherwise, an error value is returned and the values in *hbaattributes* are undefined.

Errors See libhbaapi(3LIB) for general error status values.

Examples **EXAMPLE 1** Return adapter attributes.

The following example returns the adapter attributes into hbaAttrs for the given handle.

```
if ((status = HBA_GetAdapterAttributes(handle, &hbaAttrs)) !=
                  HBA_STATUS_OK) {
        fprintf(stderr, "Unable to get adapter attributes for "
               "HBA %d with name \"%s\".\n", hbaCount, adaptername);
        HBA_CloseAdapter(handle);
        continue;
}
```

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also HBA_OpenAdapter(3HBAAPI), libhbaapi(3LIB), attributes(5)

T11 FC-MI Specification

Name HBA_GetAdapterName – retrieve the name of a specific HBA

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_GetAdapterName(HBA_UINT32 adapterindex,
     char *adaptername);
```

Parameters

adapterindex the index of the adapter, between 0 and one less than the value returned by HBA_GetNumberOfAdapters(3HBAAPI).

adaptername the buffer where the name of the adapter will be stored. The recommended size is 256 bytes.

Description The HBA_GetAdapterName() function stores the name of the adapter specified by *adapterindex* in the buffer pointed to by *adaptername*. The caller is responsible for allocating space for the name.

Return Values Upon successful completion, HBA_STATUS_OK is returned. Otherwise, an error value is returned and the content of *adaptername* is undefined.

Errors See libhbaapi(3LIB) for general error status values.

Examples **EXAMPLE 1** Return adapter name.

Given an *hbacount* >= 0 and < total number of adapters on the system, the following example returns the *adaptername* for that adapter.

```
if ((status = HBA_GetAdapterName(hbaCount, adaptername)) !=
            HBA_STATUS_OK) {
        fprintf(stderr, "HBA %d name not available for "
            "reason %d\n", hbaCount, status);
        continue;
}
```

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also HBA_GetNumberOfAdapters(3HBAAPI), libhbaapi(3LIB), attributes(5)

T11 FC-MI Specification

Bugs The `HBA_GetAdapterName()` function does not take a name length argument to define how large the buffer is, yet the specification does not indicate a maximum name length. Failure to pass in a large enough buffer will result in a buffer over-run, which may lead to segmentation faults or other failures. Callers should be sure to allocate a large buffer to ensure the Vendor library will not overrun during the copy.

Name HBA_GetAdapterPortAttributes, HBA_GetDiscoveredPortAttributes, HBA_GetPortAttributesByWWN – retrieve Fibre Channel port attributes for a specific device

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_GetAdapterPortAttributes(HBA_HANDLE handle,
     HBA_UINT32 portindex, HBA_PORTATTRIBUTES *portattributes);

HBA_STATUS HBA_GetDiscoveredPortAttributes(HBA_HANDLE handle,
     HBA_UINT32 portindex, HBA_UINT32 discoveredportindex,
     HBA_PORTATTRIBUTES *portattributes);

HBA_STATUS HBA_GetPortAttributesByWWN(HBA_HANDLE handle,
     HBA_WWN PortWWN, HBA_PORTATTRIBUTES *portattributes);
```

Parameters

handle	an open handle returned from HBA_OpenAdapter(3HBAAPI)
portindex	the index of a specific port on the HBA as returned by a call to HBA_GetAdapterAttributes(3HBAAPI). The maximum value specified should be (HBA_ADAPTERATTRIBUTES.NumberOfPorts - 1).
portattributes	a pointer to an HBA_PORTATTRIBUTES structure. Upon successful completion, this structure contains the specified port attributes.
discoveredportindex	the index of a specific discovered port on the HBA as returned by HBA_GetAdapterPortAttributes(3HBAAPI). The maximum value specified should be (HBA_PORTATTRIBUTES.NumberOfDiscoveredPorts - 1).
PortWWN	the port WWN of the device for which port attributes are retrieved.

Description The HBA_GetAdapterPortAttributes() function retrieves Port Attributes for a specific port on the HBA.

The HBA_GetDiscoveredPortAttributes() function retrieves Port Attributes for a specific discovered device connected to the HBA.

The HBA_GetPortAttributesByWWN() function retrieves Port Attributes for a specific device based on the *PortWWN* argument.

Return Values Upon successful completion, HBA_STATUS_OK is returned. Otherwise, an error value is returned from the underlying VSL and the values in *hbaattributes* are undefined.

Errors See libhbaapi(3LIB) for general error status values.

Examples **EXAMPLE 1** Retrieve the port attributes for each port on the HBA.

The following example retrieves the port attributes for each port on the HBA.

EXAMPLE 1 Retrieve the port attributes for each port on the HBA. *(Continued)*

```
for (hbaPort = 0; hbaPort < hbaAttrs.NumberOfPorts; hbaPort++) {
    if ((status = HBA_GetAdapterPortAttributes(handle,
            hbaPort, &hbaPortAttrs)) != HBA_STATUS_OK) {
        fprintf(stderr, "Unable to get adapter port %d "
            "attributes for HBA %d with name \"%s\".\n",
            hbaPort, hbaCount, adaptername);
        HBA_CloseAdapter(handle);
        continue;
    }
    memcpy(&wwn, hbaPortAttrs.PortWWN.wwn, sizeof (wwn));
    printf(" Port %d: WWN=%016llx\n", hbaPort, wwn);

    /* ... */
}
```

EXAMPLE 2 Retrieve the discovered port target attributes for each discovered target port on the HBA.

The following example retrieves the discovered port target attributes for each discovered target port on the HBA.

```
for (discPort = 0;
        discPort < hbaPortAttrs.NumberofDiscoveredPorts;
        discPort++) {
    if ((status = HBA_GetDiscoveredPortAttributes(
            handle, hbaPort, discPort,
            &discPortAttrs)) != HBA_STATUS_OK) {
        fprintf(stderr, "Unable to get "
            "discovered port %d attributes for "
            "HBA %d with name \"%s\".\n",
            discPort, hbaCount, adaptername);
        continue;
    }
    memcpy(&wwn, discPortAttrs.PortWWN.wwn,
        sizeof (wwn));
    printf("  Discovered Port %d: WWN=%016llx\n",
        discPort, wwn);

    /* ... */
}
```

Attributes

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
	Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also HBA_GetAdapterPortAttributes(3HBAAPI), HBA_OpenAdapter(3HBAAPI), libhbaapi(3LIB), attributes(5)

T11 FC-MI Specification

Name HBA_GetBindingCapability, HBA_GetBindingSupport, HBA_SetBindingSupport – return and sets binding capabilities on an HBA port

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_HANDLE HBA_GetBindingCapability(HBA_HANDLE handle,
     HBA_WWN hbaPortWWN, HBA_BIND_CAPABILITY *pFlags);

HBA_STATUS HBA_GetBindingSupport(HBA_HANDLE handle, HBA_WWN
     hbaPortWWN, HBA_BIND_CAPABILITY *pFlags);

void HBA_SetBindingSupport(HBA_HANDLE handle, HBA_WWN hbaPortWWN,
     HBA_BIND_CAPABILITY Flags);
```

Parameters

handle	an open handle returned from HBA_OpenAdapter(3HBAAPI)
hbaPortWWN	the Port WWN of the local HBA through which the binding capabilities implemented by the HBA is returned
pFlags	a pointer to an HBA_BIND_CAPABILITY structure that returns the persistent binding capabilites implemented by the HBA
Flags	an HBA_BIND_CAPABILITY structure containing the persistent binding capabilites to enable for the HBA

Description The HBA_GetBindingCapability() function returns the binding capabilities implemented by the HBA.

The HBA_GetBindingSupport() function returns the currently enabled binding capabilities for the HBA.

The HBA_SetBindingSupport() function sets the currently enabled binding capabilites for the HBA to a subset of the binding capabilities implemented by the HBA.

Return Values The HBA_GetBindingCapability() and HBA_GetBindingSupport() functions return the following values:

HBA_STATUS_OK	Persistent binding capabilites have been returned.
HBA_STATUS_ERROR_ILLEGAL_WWN	Port WWN hbaPortWWN is not a WWN contained by the HBA referenced by *handle*.
HBA_STATUS_ERROR_NOT_SUPPORTED	The HBA handle specified by *handle* does not support persistent binding.
HBA_STATUS_ERROR	An error occurred. The value of *pFlags* remains unchanged and points to the persistent binding capabilites.

The HBA_SetBindingSupport() function returns:

`HBA_STATUS_OK`	Persistent binding capabilites have been enabled.
`HBA_STATUS_ERROR_ILLEGAL_WWN`	Port WWN hbaPortWWN is not a WWN contained by the HBA referenced by *handle*.
`HBA_STATUS_ERROR_NOT_SUPPORTED`	The HBA handle specified by *handle* does not support persistent binding.
`HBA_STATUS_ERROR_INCAPABLE`	The *flags* argument contains a capability not implemented by the HBA.
`HBA_STATUS_ERROR`	An error occurred.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_GetEventBuffer – remove and return the next event from the HBA's event queue

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_GetEventBuffer(HBA_HANDLE handle,
     HBA_EVENTINFO *EventBuffer, HBA_UINT32 *EventBufferCount);
```

Parameters

handle	an open handle returned from `HBA_OpenAdapter(3HBAAPI)`
EventBuffer	a pointer to an `HBA_EVENTINFO` buffer
EventBufferCount	a pointer to the maximum number of events that can be stored in the `HBA_EVENTINFO` buffer. The value will be changed to the actual number of events placed in the buffer on completion.

Description The `HBA_GetEventBuffer()` function retrieves events from the HBA's event queue. The number of events returned is the lesser of *EventBufferCount* and the number of events on the queue. The returned events are removed from the queue.

Return Values Upon successful completion, `HBA_STATUS_OK` is returned. Otherwise, an error value is returned and the value of *EventBufferCount* is undefined.

Errors See `libhbaapi(3LIB)` for general error status values.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `HBA_OpenAdapter(3HBAAPI)`, `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_GetFcpPersistentBinding, HBA_GetPersistentBindingV2, HBA_SetPersistentBindingV2, HBA_RemovePersistentBinding, HBA_RemoveAllPersistentBindings – handle persistent bindings between FCP-2 discovered devices and operating system SCSI information

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_GetFcpPersistentBinding(HBA_HANDLE handle,
     HBA_FCPBINDING *binding);

HBA_STATUS HBA_GetPersistentBindingV2(HBA_HANDLE handle,
     HBA_WWN hbaPortWWN, HBA_FCPBINDING2 *binding);

HBA_STATUS HBA_SetPersistentBindingV2(HBA_HANDLE handle,
     HBA_WWN hbaPortWWN, HBA_FCPBINDING2 *binding);

HBA_STATUS HBA_RemovePersistentBinding(HBA_HANDLE handle,
     HBA_WWN hbaPortWWN, HBA_FCPBINDING2 *binding);

HBA_STATUS HBA_RemoveAllPersistentBindings(HBA_HANDLE handle,
     HBA_WWN hbaPortWWN);
```

Parameters

handle an open handle returned from HBA_OpenAdapter(3HBAAPI)

binding

HBA_GetFcpPersistentBinding()
: a buffer to store the binding entries in. The *binding*->NumberOfEntries member must indicate the maximum number of entries that fit within the buffer. On completion, the *binding*->NumberOfEntries member will indicate the actual number of binding entries for the HBA. This value can be greater than the number of entries the buffer can store.

HBA_GetPersistentBindingV2()
: a pointer to a HBA_FCPBINDING2 structure. The NumberOfEntries member will be the maximum number of entries returned.

HBA_SetPersistentBindingV2()
: a pointer to a HBA_FCPBINDING2 structure. The NumberOfEntries member will be the number of bindings requested in the structure.

HBA_RemovePersistentBinding()
: a pointer to a HBA_FCPBINDING2 structure. The structure will contain all the bindings to be removed. The NumberOfEntries member will be the number of bindings being requested to be removed in the structure.

hbaPortWWN

`HBA_GetPersistentBindingV2()`	The Port WWN of the local HBA through which persistent bindings will be retrieved.
`HBA_SetPersistentBindingV2()`	The Port WWN of the local HBA through which persistent bindings will be set.
`HBA_RemovePersistentBinding()` `HBA_RemoveAllPersistentBindings()`	The Port WWN of the local HBA through which persistent bindings will be removed.

Description The `HBA_GetFcpPersistentBinding()` function retrieves the set of mappings between FCP LUNs and SCSI LUNs that are reestablished upon initialization or reboot. The means of establishing the persistent bindings is vendor-specific and accomplished outside the scope of the HBA API.

The `HBA_GetPersistentBindingV2()` function retrieves the set of persistent bindings between FCP LUNs and SCSI LUNs for the specified HBA Port that are reestablished upon initialization or reboot. The means of establishing the persistent bindings is vendor-specific and accomplished outside the scope of the HBA API. The binding information can contain bindings to Logical Unit Unique Device Identifiers.

The `HBA_SetPersistentBindingV2()` function sets additional persistent bindings between FCP LUNs and SCSI LUNs for the specified HBA Port. It can also accept bindings to Logical Unit Unique Device Identifiers. Bindings already set will remain set. An error occurs if a request is made to bind to an OS SCSI ID which has already been bound. Persistent bindings will not affect Target Mappings until the OS, HBA, and/or Fabric has been reinitialized. Before then, the effects are not specified.

The `HBA_RemovePersistentBinding()` function removes one or more persistent bindings. The persistent binding will only be removed if both the OS SCSI LUN and the SCSI Lun match a binding specifed in the arguments. Persistent bindings removed will not affect Target Mappings until the OS, HBA, and/or Fabric has been reinitialized. Before then, the effects are not specified.

The `HBA_RemoveAllPersistentBindings()` function removes all persistent bindings. Persistent bindings removed will not affect Target Mappings until the OS, HBA, and/or Fabric has been reinitialized. Before then, the effects are not specified.

Return Values The `HBA_GetFcpPersistentBinding()` function returns the following values:

`HBA_STATUS_OK`
The HBA was able to retrieve information.

`HBA_STATUS_ERROR_MORE_DATA`
A a larger buffer is required. The value of *binding*->`NumberOfEntries` after the call indicates the total number of entries available. The caller should reallocate a larger buffer to accomodate the indicated number of entries and reissue the routine.

`HBA_STATUS_ERROR_NOT_SUPPORTED`
The HBA handle specified by *handle* does not support persistent binding.

In the event that other error codes are returned, the value of *binding*->`NumberOfEntries` after the call should be checked, and if greater than the value before the call, a larger buffer should be allocated for a retry of the routine.

The `HBA_GetPersistentBindingV2()` function returns the following values:

`HBA_STATUS_OK`
The HBA was able to retrieve information.

`HBA_STATUS_ERROR_MORE_DATA`
A a larger buffer is required. The value of *binding*->`NumberOfEntries` after the call indicates the total number of entries available. The caller should reallocate a larger buffer to accomodate the indicated number of entries and reissue the routine.

`HBA_STATUS_ERROR_ILLEGAL_WWN`
The Port WWN *hbaPortWWN* is not a WWN contained by the HBA referenced by *handle*.

`HBA_STATUS_ERROR_NOT_SUPPORTED`
The HBA handle specified by *handle* does not support persistent binding.

The value of *binding* remains unchanged. The structure it points to contains binding information. The number of entries returned is the minimum between the number of entries specifed in the binding argument and the total number of bindings.

The `HBA_SetPersistentBindingV2()` function returns the following values.

`HBA_STATUS_OK`
The HBA was able to set bindings.

`HBA_STATUS_ERROR_ILLEGAL_WWN`
The Port WWN *hbaPortWWN* is not a WWN contained by the HBA referenced by *handle*.

`HBA_STATUS_ERROR_NOT_SUPPORTED`
The HBA handle specified by *handle* does not support persistent binding.

The value of *binding* remains unchanged. The success or failure of each Persistent binding set is indicated in the `Status` member of the `HBA_FCPBINDINGENTRY2` structure.

The `HBA_RemovePersistentBinding()` function returns the following values:

`HBA_STATUS_OK`
The HBA was able to retrieve information.

`HBA_STATUS_ERROR_ILLEGAL_WWN`
The Port WWN *hbaPortWWN* is not a WWN contained by the HBA referenced by *handle*.

`HBA_STATUS_ERROR_NOT_SUPPORTED`
The HBA handle specified by *handle* does not support persistent binding.

The value of *binding* remains unchanged. The success or failure of each Persistent binding set is indicated in the `Status` member of the `HBA_FCPBINDINGENTRY2` structure.

The `HBA_RemoveAllPersistentBindings()` function returns the following values:

`HBA_STATUS_OK`
The HBA was able to retrieve information.

`HBA_STATUS_ERROR_ILLEGAL_WWN`
The Port WWN *hbaPortWWN* is not a WWN contained by the HBA referenced by *handle*.

`HBA_STATUS_ERROR_NOT_SUPPORTED`
The HBA handle specified by *handle* does not support persistent binding.

Errors See `libhbaapi(3LIB)` for general error status values.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `HBA_GetFcpTargetMapping(3HBAAPI)`, `HBA_OpenAdapter(3HBAAPI)`, `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Bugs The `HBA_GetFcpTargetMapping(3HBAAPI)` and `HBA_GetFcpPersistentBinding()` functions do not take a *portindex* to define to which port of a multi-ported HBA the command should apply. The behavior on multi-ported HBAs is vendor-specific and could result in mappings or bindings for all ports being intermixed in the response buffer. SNIA version 2 defines a `HBA_GetFcpTargetMappingV2()` that takes a Port WWN as an argument. This fixes the bug with multi-ported HBAs in `HBA_GetFcpTargetMapping()`.

Name HBA_GetFcpTargetMapping, HBA_GetFcpTargetMappingV2 – retrieve mapping between FCP-2 discovered devices and operating system SCSI information

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_GetFcpTargetMapping(HBA_HANDLE handle,
     HBA_FCPTARGETMAPPING *mapping);

HBA_STATUS HBA_GetFcpTargetMappingV2(HBA_HANDLE handle,
     HBA_WWN hbaPortWWN, HBA_FCPTARGETMAPPINGV2 *mapping);
```

Parameters

handle an open handle returned from HBA_OpenAdapter(3HBAAPI)

mapping a buffer in which to store the mapping entries. The *mapping*->NumberOfEntries member must indicate the maximum number of entries that will fit within the buffer. On completion, the *mapping*->NumberOfEntries member indicates the actual number of mapping entries for the HBA. This value can be greater than the number of entries the buffer can store.

hbaPortWWN the Port Name of the local HBA Port for which the caller is requesting target mappings.

Description The HBA_GetFcpTargetMapping() function retrieves the current set of mappings between FCP LUNs and SCSI LUNs for a given HBA port.

The HBA_GetFcpTargetMappingV2() function retrieves the current set of mappings between FCP LUNs and SCSI LUNs for a given HBA. The mapping also includes a Logical Unit Unique Identifier for each logical unit.

Return Values The HBA_GetFcpTargetMappingV2() function returns the following values:

HBA_STATUS_ERROR_ILLEGAL_WWN
The port WWN specified by *hbaPortWWN* is not a valid port WWN on the specified HBA

HBA_STATUS_ERROR_NOT_SUPPORTED
Target mappings are not supported on the HBA.

HBA_STATUS_ERROR
An error occurred.

The HBA_GetFcpTargetMapping() and HBA_GetFcpTargetMappingV2() functions return the following values:

HBA_STATUS_OK
The HBA was able to retrieve information.

HBA_STATUS_ERROR_MORE_DATA
A larger buffer is required. The value of *mapping*->NumberOfEntries after the call indicates the total number of entries available. The caller should reallocate the buffer large

enough to accomodate the indicated number of entries and reissue the routine.

In the event that other error values are returned, the value of *mapping*->`NumberOfEntries` after the call should be checked, and if greater than the value before the call, a larger buffer should be allocated for a retry of the routine.

Errors See `libhbaapi(3LIB)` for general error status values.

Examples **EXAMPLE 1** Return target mapping data.

The following example returns target mapping data. It initially allocates space for one target mapping. If the number of entries returned is greater than the allocated space, a new buffer with sufficient space is allocated and `HBA_GetFcpTargetMapping()` is called again.

```
map = (HBA_FCPTARGETMAPPING *)calloc(1,
    sizeof (HBA_FCPTARGETMAPPING));
status = HBA_GetFcpTargetMapping(handle, map);
if (map->NumberOfEntries > 0) {
    HBA_UINT32  noe = map->NumberOfEntries;
    free(map);
    map = (HBA_FCPTARGETMAPPING *)calloc (1,
        sizeof (HBA_FCPTARGETMAPPING) +
        (sizeof (HBA_FCPSCSIENTRY)*(noe - 1)));
    map->NumberOfEntries = noe;
    if ((status = HBA_GetFcpTargetMapping(handle, map)) !=
            HBA_STATUS_OK) {
        fprintf(stderr, " Failed to get target "
            "mappings %d", status);
        free(map);
    } else {
        printf(" FCP Mapping entries: \n");
        for (cntr = 0;
            cntr < map->NumberOfEntries;
            cntr ++) {
        printf("  Path(%d): \"%s\"\n", cntr,
        map->entry[cntr].ScsiId.OSDeviceName);
        }
    }
}
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Standard	FC-MI 1.92 (API version 1)

See Also `HBA_OpenAdapter(3HBAAPI)`, `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Bugs The `HBA_GetFcpTargetMapping()` routine does not take a *portindex* to define which port of a multi-ported HBA the command should apply to. The behavior on multi-ported HBAs is vendor specific, and may result in mappings or bindings for all ports being intermixed in the response buffer. SNIA version 2 defines a `HBA_GetFcpTargetMappingV2()` which takes a Port WWN as an argument. This fixes the bug with multi-ported HBAs in `HBA_GetFcpTargetMapping()`.

Name HBA_GetNumberOfAdapters – report the number of HBAs known to the Common Library

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_UINT32 HBA_GetNumberOfAdapters(void);
```

Description The `HBA_GetNumberOfAdapters()` function report the number of HBAs known to the Common Library. This number is the sum of the number of HBAs reported by each VSL loaded by the Common Library.

Return Values The `HBA_GetNumberOfAdapters()` function returns the number of adapters known to the Common Library will be returned.

Examples **EXAMPLE 1** Using `HBA_GetNumberOfAdapters()`

```
numberOfAdapters = HBA_GetNumberOfAdapters();
for (hbaCount = 0; hbaCount < numberOfAdapters; hbaCount++) {
    /* ... */
}
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_GetPortStatistics, HBA_GetFC4Statistics, HBA_GetFCPStatistics, HBA_ResetStatistics – Access Port statistics for a specific HBA port.

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_GetPortStatistics(HBA_HANDLE handle,
     HBA_UINT32 portindex, HBA_PORTSTATISTICS *portstatistics);

HBA_STATUS HBA_GetFC4Statistics(HBA_HANDLE handle, HBA_WWN portWWN,
     HBA_UINT8 FC4type, HBA_FC4STATISTICS * statistics);

HBA_STATUS HBA_GetFCPStatistics(HBA_HANDLE handle,
     const HBA_SCSIID * lunid, HBA_FC4STATISTICS * statistics);

void HBA_ResetStatistics(HBA_HANDLE handle, HBA_UINT32 portindex);
```

Parameters

handle	an open handle returned from HBA_OpenAdapter(3HBAAPI)
portindex	the index of a specific port on the HBA as returned by a call to HBA_GetAdapterAttributes(3HBAAPI). The maximum value specified should be (HBA_ADAPTERATTRIBUTES.NumberOfPorts - 1).
portstatistics	a pointer to an HBA_PORTSTATISTICS structure. Upon successful completion, this structure contains the specified port attributes.
portWWN	the Port WWN of the local HBA for which FC-4 statistics is being returned
FC4type	FC-4 protocol Data Structure Type as defined in FC-FS for which statistics are being requested
statistics	a pointer to an HBA_FC4STATISTICS structure where the specified statistics is being returned
lunid	a pointer to an HBA_SCSIID structure specifying the OS SCSI logical unit where statistics are being requested

Description The HBA_GetPortStatistics() function retrieves the statistical information from a given HBA port.

The HBA_GetFC4Statistics() function retrieves the traffic statistics for a specific FC-4 protocol.

The HBA_GetFCPStatistics() function retrieves the traffic statistics for a specific FC-4 protocol on the specified OS SCSI logical unit through that port.

The HBA_ResetStatistics() function sesets the statistical counters to zero for a given HBA port.

Return Values Upon successful completion, `HBA_GetPortStatistics()` returns `HBA_STATUS_OK`. Otherwise, an error value is returned from the underlying VSL and the values in *portstatistics* are undefined. If the VSL does not support a specific statistic, that statistic will have every bit set to 1.

Upon successful completion, `HBA_GetFC4Statistics()` and `HBA_GetFCPStatistics()` return `HBA_STATUS_OK`. Otherwise, an error value is returned from the underlying VSL and the values in *statistics* are undefined. If the VSL does not support a specific statistic, that statistic will have every bit set to 1.

Errors See `libhbaapi(3LIB)` for general error status values.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `HBA_GetAdapterAttributes(3HBAAPI)`, `HBA_OpenAdapter(3HBAAPI)`, `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_GetVersion – determine the version of the API supported by the Common Library

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_UINT32 HBA_GetVersion(void);
```

Description The `HBA_GetVersion()` function returns the version of the API that the Common Library supports.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_GetWrapperLibraryAttributes, HBA_GetVendorLibraryAttributes – return details about the implementation of the wrapper library and the vendor specific library

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_UINT32 HBA_GetWrapperLibraryAttributes(
     HBA_LIBRARYATTRIBUTES *attributes);

HBA_UINT32 HBA_GetVendorLibraryAttributes(HBA_UINT32 adapter_index,
     HBA_LIBRARYATTRIBUTES *attributes);
```

Parameters *attributes*

`HBA_GetWrapperLibraryAttributes()`	a pointer to a `HBA_LIBRARYATTRIBUTES` structure where the wrapper library information is returned
`HBA_GetVendorLibraryAttributes()`	a pointer to a `HBA_LIBRARYATTRIBUTES` structure where the vendor-specific library information is returned

adapter_index index of the HBA. The value ust be withing the range of 1 and the value returned by `HBA_GetNumberOfAdapters(3HBAAPI)`.

Description The `HBA_GetWrapperLibraryAttributes()` function returns details about the wrapper library.

The `HBA_GetVendorLibraryAttributes()` function returns details about the vendor specific library. The vendor-specific library selected is based on the *adapter_index*.

Return Values The `HBA_GetWrapperLibraryAttributes()` and `HBA_GetVendorLibraryAttributes()` functions return the version of the HBA API specification.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `HBA_GetNumberOfAdapters(3HBAAPI)`, `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_LoadLibrary, HBA_FreeLibrary – load and free the resources used by the HBA Common Library

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_LoadLibrary(void);

HBA_STATUS HBA_FreeLibrary(void);
```

Description The `HBA_LoadLibrary()` function loads the Common Library, which in turn loads each VSL specified in the `hba.conf(4)` file.

The `HBA_FreeLibrary()` function releases resources held by the Common Library and each loaded VSL.

Return Values Upon successful completion, `HBA_LoadLibrary()` and `HBA_FreeLibrary()` return `HBA_STATUS_OK`. Otherwise, an error value is returned.

Errors See `libhbaapi(3LIB)` for general error status values.

Examples **EXAMPLE 1** Load the common library and each VSL.

The following example loads the common library and each VSL.

```
if ((status = HBA_LoadLibrary()) != HBA_STATUS_OK) {
    fprintf(stderr, "HBA_LoadLibrary failed: %d\\\
", status);
    return;
}
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `libhbaapi(3LIB)`, `hba.conf(4)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_OpenAdapter, HBA_OpenAdapterByWWN, HBA_CloseAdapter – open and close a specific adapter

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_HANDLE HBA_OpenAdapter(char *adaptername);

HBA_STATUS HBA_OpenAdapterByWWN(HBA_HANDLE *handle, HBA_WWN wwn);

void HBA_CloseAdapter(HBA_HANDLE handle);
```

Parameters

adaptername the name of the adapter to open, as returned by HBA_GetAdapterName(3HBAAPI)

handle

HBA_OpenAdapterByWWN()	a pointer to an HBA_HANDLE
HBA_CloseAdapter()	the open handle of the adapter to close, as returned by HBA_OpenAdapter(3HBAAPI)

wwn the WWN to match the Node WWN or Port WWN of the HBA to open

Description The HBA_OpenAdapter() function opens the adapter specified by *adaptername* and returns a handle used for subsequent operations on the HBA.

The HBA_OpenAdapterByWWN() function opens a handle to the HBA whose Node or Port WWN matches the *wwn* argument.

The HBA_CloseAdapter() function closes the open handle.

Return Values Upon successful completion, HBA_OpenAdapter() returns a valid HBA_HANDLE with a numeric value greater than 0. Otherwise, 0 is returned.

The HBA_OpenAdapterByWWN() function returns the following values:

HBA_STATUS_OK	The *handle* argument contains a valid HBA handle.
HBA_STATUS_ERROR_ILLEGAL_WWN	The *wwn* argument is not a valid port WWN on the specified HBA.
HBA_STATUS_ERROR_AMBIGUOUS_WWN	The WWN is matched to multiple adapters.
HBA_STATUS_ERROR	An error occurred while opening the adapter.

Examples **EXAMPLE 1** Open an adapter.

The following example opens the specified adapter.

```
handle = HBA_OpenAdapter(adaptername);
if (handle == 0) {
    fprintf(stderr, "Unable to open HBA %d with name "
```

EXAMPLE 1 Open an adapter. *(Continued)*

```
        "\"%s\".\n", hbaCount, adaptername);
    continue;
}
```

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also

`HBA_GetAdapterName(3HBAAPI)`, `HBA_OpenAdapter(3HBAAPI)`, `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_RefreshInformation, HBA_RefreshAdapterConfiguration – refresh information for a specific HBA

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

void HBA_RefreshInformation(HBA_HANDLE handle);

void HBA_RefreshAdapterConfiguration(void);
```

Parameters *handle* an open handle returned from HBA_OpenAdapter(3HBAAPI)

Description The HBA_RefreshInformation() function requests that the underlying VSL reload all information about the given HBA. This function should be called whenever any function returns HBA_STATUS_ERROR_STALE_DATA, or if an index that was previously valid returns HBA_STATUS_ERROR_ILLEGAL_INDEX. Because the underlying VSL can reset all indexes relating to the HBA, all old index values must be discarded by the caller.

The HBA_RefreshAdapterConfiguration() function updates information about the HBAs present on the system. This function does not change any of the relationships between the HBA API and adapters that have not been reconfigured. HBA handles continue to refer to the same HBA even if it is no longer installed. The HBA name or index assigned by the library remains assigned to the same HBA even if it has been removed and reinstalled, as long as the bus position, WWN, and OS device have not changed. Adapter that have been removed and not replaced cannot have their HBA handles, HBA names, and HBA indexes reassigned. Calls to these adapters will generate HBA_STATUS_ERROR_UNAVAILABLE.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also HBA_OpenAdapter(3HBAAPI), libhbaapi(3LIB), attributes(5)

T11 FC-MI Specification

Name HBA_RegisterForAdapterEvents, HBA_RegisterForAdapterAddEvents, HBA_RegisterForAdapterPortEvents, HBA_RegisterForAdapterPortStatEvents, HBA_RegisterForTargetEvents, HBA_RegisterForLinkEvents, HBA_RemoveCallback – SNIA event handling functions

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_RegisterForAdapterEvents(void (*pCallback)
     (void *pData, HBA_WWN PortWWN, HBA_UINT32 eventType),
     void *pUserData, HBA_HANDLE handle,
     HBA_CALLBACKHANDLE *pCallbackHandle);

HBA_STATUS HBA_RegisterForAdapterAddEvents(void (*pCallback)
     (void *pData, HBA_WWN PortWWN, HBA_UINT32 eventType),
     void *pUserData, HBA_CALLBACKHANDLE *pCallbackHandle);

HBA_STATUS HBA_RegisterForAdapterPortEvents(void (*pCallback)
     (void *pData, HBA_WWN PortWWN, HBA_UINT32 eventType,
     HBA_UINT32 fabricPortID), void *pUserData, HBA_HANDLE handle,
     HBA_WWN PortWWN, HBA_CALLBACKHANDLE *pCallbackHandle);

HBA_STATUS HBA_RegisterForAdapterPortStatEvents(void (*pCallback)
     (void *pData, HBA_WWN PortWWN, HBA_UINT32 eventType),
     void *pUserData, HBA_HANDLE handle, HBA_WWN PortWWN,
     HBA_PortStatistics stats, HBA_UINT32 statType,
     HBA_CALLBACKHANDLE *pCallbackHandle);

HBA_STATUS HBA_RegisterForTargetEvents(void (*pCallback)
     (void *pData, HBA_WWN hbaPortWWN, HBA_WWN discoveredPortWWN,
     HBA_UINT32 eventType), void * pUserData, HBA_HANDLE handle,
     HBA_WWN hbaPortWWN, HBA_WWN discoveredPortWWN,
     HBA_CALLBACKHANDLE *pCallbackHandle, HBA_UINT32 allTargets);

HBA_STATUS HBA_RegisterForLinkEvents(void (*pCallback)
     (void *pData, HBA_WWN adapterWWN, HBA_UINT32 eventType,
     void * pRLIRBuffer, HBA_UINT32 RLIRBufferSize),
     void *pUserData, void *PLIRBuffer, HBA_UINT32 RLIRBufferSize,
     HBA_HANDLE handle, HBA_CALLBACKHANDLE *pCallbackHandle);

HBA_STATUS HBA_RemoveCallback(HBA_CALLBACKHANDLE *pCallbackHandle);
```

Parameters *pCallback*

A pointer to the entry of the callback routine.

pData

the *pUserData* that is passed in from registration. This parameter can be used to correlate the event with the source of its event registration.

PortWWN

The Port WWN of the HBA for which the event is being reported.

hbaPortWWN
The Port WWN of the HBA for which the target event is being reported.

discoveredPortWWN
The Port WWN of the target for which the target event is being reported.

adapterWWN
The Port WWN of the of the HBA for which the link event is being reported.

eventType
a value indicating the type of event that has occured.

`HBA_RegisterForAdapterEvents()`	Possible values are `HBA_EVENT_ADAPTER_REMOVE` and `HBA_EVENT_ADAPTER_CHANGE`.
`HBA_RegisterForAdaterAddEvents()`	The only possible value is `HBA_EVENT_ADAPTER_ADD`.
`HBA_RegisterForAdaterPortEvents()`	Possible values are `HBA_EVENT_PORT_OFFLINE`, `HBA_EVENT_PORT_ONLINE`, `HBA_EVENT_PORT_NEW_TARGETS`, `HBA_EVENT_PORT_FABRIC`, and `HBA_EVENT_PORT_UNKNOWN`.
`HBA_RegisterForAdapterPortStatEvents()`	Possible values are `HBA_EVENT_PORT_STAT_THRESHOLD` and `HBA_EVENT_PORT_STAT_GROWTH`.
`HBA_RegisterForTargetEvents()`	If the value is `HBA_EVENT_LINK_INCIDENT`, RLIR has occured and information is in the RLIRBuffer. If the value is `HBA_EVENT_LINK_UNKNOWN`, a fabric link or topology change has occured and was not detected by RLIR. The RLIRBuffer is ignored
`HBA_RegisterForLinkEvents()`	Possible values are `HBA_EVENT_TARGET_OFFLINE`, `HBA_EVENT_TARGET_ONLINE`, `HBA_EVENT_TARGET_REMOVED`, and `HBA_EVENT_TARGET_UNKNOWN`.

fabricPortID
If the event is of type `HBA_EVENT_PORT_FABRIC`, this parameter will be the RSCN-affected Port ID page as defined in FC-FS. It is ignored for all other event types.

pRLIRBuffer
A pointer to a buffer where RLIR data may be passed to the callback function. The buffer will be overwritten for each fabric link callback function, but will not be overwritten within a single call to the callback function.

RLIRBufferSize
Size in bytes of the RLIRBuffer.

pUserData
a pointer passed with each event to the callback routine that can be used to correlate the event with the source of its event registration

pRLIRBuffer
A pointer to a buffer where RLIR data may be passed to the callback function. The buffer will be overwritten for each fabric link callback function, but will not be overwritten within a single call to the callback function.

RLIRBufferSize
Size in bytes of the RLIRBuffer.

handle
a handle to the HBA that event callbacks are being requested

PortWWN
The Port WWN of the HBA for which the event is being reported.

hbaPortWWN
The Port WWN of the HBA of which the event callbacks are being requested.

stats
an HBA_PortStatistics structure which indicates the counters to be monitored. If *statType* is `HBA_EVENT_PORT_STAT_THRESHOLD`, any non-null values are thresholds for which to watch. If *statType* is `HBA_EVET_PORT_STAT_GROWTH`, any non-null values are growth rate numbers over 1 minute.

statType
A value either `HBA_EVENT_PORT_STAT_TRHESHOLD` or `HBA_EVENT_PORT_STAT_GROWTH` used to determine whether counters registered are for threshold crossing or growth rate.

discoveredPortWWN
The Port WWN of the target that the event callbacks are being requested of.

pCallbackHandle
A pointer to structure in which an opaque identifier is returned that is used to deregister the callback. To deregister this event, call `HBA_RemoveCallback()` with this *pCallbackHandle* as an argument.

allTargets
If value is non-zero, *discoveredPortWWN* is ignored. Events for all discovered targets will be registered by this call. If value is zero, only events for *discoveredPortWWN* will be registered.

pcallbackHandle
A handle returned by the event registration function of the routine that is to be removed.

Description The `HBA_RegisterForAdapterEvents()` function registers an application-defined function that is called when an HBA category asynchronous event occurs. An HBA catetory event can have one of the following event types: `HBA_EVENT_ADAPTER_REMOVE` or `HBA_EVENT_ADAPTER_CHANGE`. If either of these events occur, the callback function is called, regardless of whether the HBA handle specified at registration is open. The `HBA_RemoveCallback()` function must be called to end event delivery.

The `HBA_RegisterForAdapterAddEvents()` function registers an application-defined function that is called whenever an HBA add category asynchronous event occurs. The callback function is called when a new HBA is added to the local system. The `HBA_RemoveCallback()` function must be called to end event delivery.

The `HBA_RegisterForAdapterPortEvents()` function registers an application-defined function that is called on the specified HBA whenever a port category asynchronous event occurs. A port catetory event can be one of the following event types: `HBA_EVENT_PORT_OFFLINE`, `HBA_EVENT_PORT_ONLINE`, `HBA_EVENT_PORT_NEW_TARGETS`, `HBA_EVENT_PORT_FABRIC`, or `HBA_EVENT_PORT_UNKNOWN`. The handle need not be open for callbacks to occur. The `HBA_RemoveCallback()` function must be called to end event delivery.

The `HBA_RegisterForAdapterPortStatEvents()` function defines conditions that would cause an HBA port statistics asynchronous event and registers an application-defined function that is called whenever one of these events occur. An HBA port statistics asynchronous event can be one of the following event types: `HBA_EVENT_PORT_STAT_THRESHOLD` or `HBA_EVENT_PORT_STAT_GROWTH`. More than one statistic can be registered with one call by setting multiple statistics in the *stats* argument. For threshold events, once a specific threshold has been crossed, the callback is automatically deregistered for that statistic. The handle need not be open for callbacks to occur. The `HBA_RemoveCallback()` function must be called to end event delivery.

The `HBA_RegisterForTargetEvents()` function registers an application-defined function that is called on the specified HBA whenever a target category asynchronous event occurs. A Target category event can be one of the following event types: `HBA_EVENT_TARGET_OFFLINE`, `HBA_EVENT_TARGET_ONLINE`, `HBA_EVENT_TARGET_REMOVED`, `HBA_EVENT_TARGET_UNKNOWN`. The handle need not be open for callbacks to occur. The `HBA_RemoveCallback()` function must be called to end event delivery.

The `HBA_RegisterForLinkEvents()` function registers an application defined function that is called on the specified HBA whenever a link category asynchronous event occurs. A link category event can be one of the following event types: `HBA_EVENT_LINK_INCIDENT` or `HBA_EVENT_LINK_UNKNOWN`. RLIR ELS is the only fabric link event type and the callback function is called whenever is it detected by the HBA. The handle need not be open for callbacks to occur. The `HBA_RemoveCallback()` function must be called to end event delivery.

The `HBA_RemoveCallback()` function removes the `HBA_CALLBACKHANDLE` instance of the callback routine.

Return Values Upon successful completion, `HBA_RegisterForAdapterEvents()`, `HBA_RegisterForAdapterAddEvents()`, `HBA_RegisterForAdapterPortEvents()`, `HBA_RegisterForAdapterPortStatEvents()`, `HBA_RegisterForTargetEvents()`, and `HBA_RegisterForLinkEvents()` return `HBA_STATUS_OK` and *pCallbackHandle* may be used to deregister the callback. Otherwise, an error value is returned and *pCallbackHandle* is not valid.

Upon successful completion, `HBA_RemoveCallback()` returns `HBA_STATUS_OK`. Otherwise, an error value is returned.

Errors See `libhbaapi(3LIB)` for general error status values.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_SendCTPassThru, HBA_SendCTPassThruV2 – end a Fibre Channel Common Transport request to a Fabric

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_SendCTPassThru(HBA_HANDLE handle,
     void *pReqBuffer, HBA_UINT32 ReqBufferSize,
     void *pRspBuffer, HBA_UINT32 RspBufferSize);

HBA_STATUS HBA_SendCTPassThruV2(HBA_HANDLE handle,
     HBA_WWN hbaPortWWN, void *pReqBuffer,
     HBA_UINT32 ReqBufferSize, void *pRspBuffer,
     HBA_UINT32 *RspBufferSize);
```

Parameters

handle an open handle returned from HBA_OpenAdapter(3HBAAPI)

hbaPortWWN the Port Name of the local HBA Port through which the caller is issuing the CT request

pReqBuffer a pointer to a CT_IU request. The contents of the buffer must be in big-endian byte order

ReqBufferSize the length of the CT_IU request buffer *pReqBuffer*

pRspBuffer a pointer to a CT_IU response buffer. The response received from the fabric is copied into this buffer in big-endian byte order. Success of the function need not imply success of the command. The CT_IU Command/Response field should be checked for the Accept Response code.

RspBufferSize

HBA_SendCTPassThru()	the length of the CT_IU accept response buffer *pRspBuffer*.
HBA_SendCTPassThruV2()	a Pointer to the length of the CT_IU accept response buffer *pRspBuffer*.

Description The HBA_SendCTPassThru() and HBA_SendCTPassThruV2() functions provide access to the standard in-band fabric management interface. The *pReqBuffer* argument is interpreted as a CT_IU request, as defined by the T11 specification FC-GS-3, and is routed in the fabric based on the GS_TYPE field.

Return Values Upon successful transport and receipt of a CT_IU response, HBA_SendCTPassThru() returns HBA_STATUS_OK. The CT_IU payload indicates whether the command was accepted by the fabric based on the Command/Response code returned. Otherwise, an error value is returned from the underlying VSL and the values in *pRspBuffer* are undefined.

Upon successful transport and receipt of a CT_IU response, HBA_SendCTPassThruV2() returns HBA_STATUS_OK. The CT_IU payload indicates whether the command was accepted by

the fabric based on the Command/Response code returned. Otherwise, an error code is returned from the underlying VSL, and the values in *pRspBuffer* are undefined. The `HBA_SendCTPassThruV2()` function returns the following values:

`HBA_STATUS_ERROR_ILLEGAL_WWN`	The value of *hbaPortWWN* is not a valid port WWN on the specified HBA.
`HBA_STATUS_ERROR`	An error occurred.

Errors See `libhbaapi(3LIB)` for general error status values.

Examples **EXAMPLE 1** Data structures for the GIEL command.

```
struct ct_iu_preamble {
    uint32_t  ct_rev        : 8,
              ct_inid       : 24;
    uint32_t  ct_fcstype    : 8,
              ct_fcssubtype : 8,
              ct_options    : 8,
              ct_reserved1  : 8;
    uint32_t  ct_cmdrsp     : 16,
              ct_aiusize    : 16;
    uint32_t  ct_reserved2  : 8,
              ct_reason     : 8,
              ct_expln      : 8,
              ct_vendor     : 8;
};
struct gs_ms_ic_elem {
    uchar_t   elem_name[8];
    uint32_t  reserved1   : 24,
              elem_type   : 8;
};
struct gs_ms_giel_rsp {
    struct ct_iu_preamble  ct_header;
    uint32_t               num_elems;
struct gs_ms_ic_elem   elem_list[1];
};
#define MAX_PAYLOAD_LEN 65536 /* 64K */
```

EXAMPLE 2 Send an GIEL Management Service command through the given HBA handle.

The following example sends an GIEL Management Service command through the given HBA handle.

```
req.ct_rev        = 0x01;
req.ct_fcstype    = 0xFA;   /* Management Service */
req.ct_fcssubtype = 0x01;   /* Config server */
req.ct_cmdrsp     = 0x0101; /* GIEL command */
req.ct_aiusize    = MAX_PAYLOAD_LEN / 4 -
```

EXAMPLE 2 Send an GIEL Management Service command through the given HBA handle. *(Continued)*

```
         sizeof (struct ct_iu_preamble) / 4;
if ((status = HBA_SendCTPassThru(handle, &req, sizeof (req),
        rsp, MAX_PAYLOAD_LEN)) != HBA_STATUS_OK) {
    fprintf(stderr, "Unable to issue CT command on \\"%s\\""
        " for reason %d\
", adaptername, status);
} else {
    giel = (struct gs_ms_giel_rsp *)rsp;
    if (giel->ct_header.ct_cmdrsp != 0x8002) {
        fprintf(stderr, "CT command rejected on HBA "
            "\\"%s\\"\
", adaptername);
    } else {
        for (cntr = 0; cntr < giel->num_elems; cntr++) {
        memcpy(&wwn, giel->elem_list[cntr].elem_name, 8);
        printf(" Fabric element name: %016llx\
", wwn);
        }
    }
}
```

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also

`HBA_OpenAdapter(3HBAAPI)`, `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Bugs

The `HBA_SendCTPassThru()` function does not take a *portindex* to define through which port of a multi-ported HBA to send the command. The behavior on multi-ported HBAs is vendor specific, and can result in the command always being sent on port 0 of the HBA. SNIA version 2 defines `HBA_SendCTPassThruV2()` which takes a Port WWN as an argument. This fixes the bug with multi-ported HBAs in `HBA_SendCTPassThru()`.

Name HBA_SendRLS, HBA_SendRPL, HBA_SendRPS, HBA_SendSRL, HBA_SendLIRR – issue an Extended Link Service through the local HBA Port

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS  HBA_SendRLS(HBA_HANDLE handle, HBA_WWN hbaPortWWN,
     HBA_WWN destWWN, void * pRspBuffer,
     HBA_UINT32 *pRspBufferSize);

HBA_STATUS HBA_SendRPL(HBA_HANDLE handle, HBA_WWN hbaPortWWN,
     HBA_WWN agentWWN, HBA_UINT32 agent_domain,
     HBA_UINT32 portIndex, void * pRspBuffer,
     HBA_UINT32 *pRspBufferSize);

HBA_STATUS HBA_SendRPS(HBA_HANDLE handle, HBA_WWN hbaPortWWN,
     HBA_WWN agentWWN, HBA_UINT32 agent_domain,
     HBA_WWN object_wwn, HBA_UINT32 object_port_number,
     void * pRspBuffer, HBA_UINT32 *pRspBufferSize);

HBA_STATUS HBA_SendSRL(HBA_HANDLE handle, HBA_WWN hbaPortWWN,
     HBA_WWN wwn, HBA_UINT32 domain,
     void * pRspBuffer, HBA_UINT32 *pRspBufferSize);

HBA_STATUS HBA_SendLIRR(HBA_HANDLE handle, HBA_WWN hbaPortWWN,
     HBA_WWN destWWN, HBA_UINT8 function, HBA_UINT8 type,
     void * pRspBuffer, HBA_UINT32 *pRspBufferSize);
```

Parameters *handle*

an open handle returned from HBA_OpenAdapter(3HBAAPI)

hbaPortWWN

HBA_SendRLS()	the Port WWN of the local HBA through which to send the RLS
HBA_SendRPL()	the Port WWN of the local HBA through which to send the RPL
HBA_SendRPS()	the Port WWN of the local HBA through which to send the RPS
HBA_SendSRL()	the Port WWN of the local HBA through which to send the SRL
HBA_SendLIRR()	the Port WWN of the local HBA through which to send the LIRR

destWWN

HBA_SendRLS()	the Port WWN of the remote Target to which the RLS is sent
HBA_SendLIRR()	he Port WWN of the remote Target to which the LIRR is sent

wwn

If non-zero, *wwn* is the port WWN to be scanned. If *wwn* is zero, it is ignored.

domain
If *wwn* is zero, *domain* is the domain number for which loops will be scanned. If *wwn* is non-zero, domain is ignored.

agent_wwn
If non-zero, *agent_wwn* is the port WWN for which the port list is requested. If *agent_wwn* is zero, it is ignored.

agent_domain
If *agent_wwn* is non-zero, *agent_domain* is the domain number and the domain controller for which the port list is requested. If *agent_wwn* is zero, it is ignored.

port_index
index of the first FC_Port returned in the response list

object_wwn
If non-zero, *object_wwn* is the port WWN for which the Port Status is requested. If *object_wwn* is zero, it is ignored.

object_port_number
If *object_wwn* is zero, *object_port_number* is the relative port number of the FC_Port for which the Port Status is requested. If *object_wwn* is non-zero, *object_port_number* is ignored.

function
the registration function to be performed

type
If *type* is non-zero, the type is the FC-4 device TYPE for which specific link incident information requested is requested. If *type* is zero, only common link incident information is requested.

pRspBuffer

`HBA_SendRLS()`	a pointer to a buffer into which the RLS response is copied
`HBA_SendRPL()`	a pointer to a buffer into which the RPL response is copied
`HBA_SendRPS()`	a pointer to a buffer into which the RPS response is copied
`HBA_SendSRL()`	a pointer to a buffer into which the SRL response is copied
`HBA_SendLIRR()`	A pointer to a buffer into which the LIRRresponse is copied.

RspBufferSize
a pointer to the size of the buffer

`HBA_SendRLS()`	
`HBA_SendLIRR()`	A size of 28 is sufficient for the largest response.
`HBA_SendRPS()`	A size of 58 is sufficient for the largest response.
`HBA_SendSRL()`	A size of 8 is sufficient for the largest response.

Description The `HBA_SendRLS()` function returns the Link Error Status Block associated with the agent WWN or agent-domain. For more information see "Read Link Status Block (RLS)" in FC-FS.

The `HBA_SendRPL()` function returns the Read Port List associated with the agent WWN or agent-domain. For more information see "Read Port List (RPL)" in FC-FS.

The `HBA_SendRPS()` function returns the Read Port Status Block associated with the agent WWN or agent-domain. For more information see "Read Port Status Block(RPS)" in FC-FS.

The `HBA_SendSRL()` function returns the Scan Remote Loop associated with the agent WWN or agent-domain. For more information see "Scan Remote Loop(SRL)" in FC-FS.

The `HBA_SendLIRR()` function returns the Link Incident Record Registration associated with the destportWWN. For more information see "Link Incident Record Registration (LIRR) in FC-FS.

Return Values These functions return the following values:

`HBA_STATUS_OK`	The LS_ACC for the ELS has been returned.
`HBA_STATUS_ERROR_ELS_REJECT`	The ELS has been rejeced by the local HBA Port.
`HBA_STATUS_ERROR_ILLEGAL_WWN`	The value of *hbaPortWWN* is not a valid port WWN on the specified HBA.
`HBA_STATUS_ERROR`	An error occurred.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `HBA_OpenAdapter(3HBAAPI)`, `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Name HBA_SendScsiInquiry, HBA_ScsiInquiryV2, HBA_SendReportLUNs, HBA_ScsiReportLUNsV2, HBA_SendReadCapacity, HBA_ScsiReadCapacityV2 – gather SCSI information from discovered ports

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_SendScsiInquiry(HBA_HANDLE handle, HBA_WWN PortWWN,
     HBA_UINT64 fcLUN, HBA_UINT8 EVPD, HBA_UINT32 PageCode,
     void *pRspBuffer, HBA_UINT32 RspBufferSize,
     void *pSenseBuffer, HBA_UINT32 SenseBufferSize);

HBA_STATUS HBA_ScsiInquiryV2(HBA_HANDLE handle, HBA_WWN hbaPortWWN,
     HBA_WWN discoveredPortWWN, HBA_UINT64 fcLUN, HBA_UINT8 CDB_BYTE1,
     HBA_UINT8 CDB_BYTE2, void *pRspBuffer, HBA_UINT32 *pRspBufferSize,
     HBA_UINT8 *pScsiStatus, void *pSenseBuffer,
     HBA_UINT32 *pSenseBufferSize);

HBA_STATUS HBA_SendReportLUNs(HBA_HANDLE handle, HBA_WWN PortWWN,
     void *pRspBuffer, HBA_UINT32 RspBufferSize,
     void *pSenseBuffer, HBA_UINT32 SenseBufferSize);

HBA_STATUS HBA_ScsiReportLUNsV2(HBA_HANDLE handle, HBA_WWN hbaPortWWN,
     HBA_WWN discoveredPortWWN, void *pRspBuffer,
     HBA_UINT32 *pRspBufferSize, HBA_UINT8 *pScsiStatus,
     void *pSenseBuffer, HBA_UINT32 *pSenseBufferSize);

HBA_STATUS HBA_SendReadCapacity(HBA_HANDLE handle, HBA_WWN PortWWN,
     HBA_UINT64 fcLUN, void *pRspBuffer, HBA_UINT32 RspBufferSize,
     void *pSenseBuffer, HBA_UINT32 SenseBufferSize);

HBA_STATUS HBA_ScsiReadCapacityV2(HBA_HANDLE handle
     HBA_WWN hbaPortWWN, HBA_WWN discoveredPortWWN,
     HBA_UINT64 fcLUN, void *pRspBuffer, HBA_UINT32 *pRspBufferSize,
     HBA_UINT8 *pScsiStatus, void *pSenseBuffer,
     HBA_UINT32 *pSenseBufferSize);
```

Parameters

handle
an open handle returned from HBA_OpenAdapter(3HBAAPI)

PortWWN
the port WWN of the discovered remote device to which the command is sent

hbaPortWWN

HBA_ScsiInquiryV2()
the Port WWN of the local HBA through which the SCSI INQUIRY command is issued

HBA_ScsiReportLUNsV2()
the Port WWN of the local HBA through which the SCSI REPORT LUNS command is issued

`HBA_ScsiReadCapacityV2()`
the Port WWN of a local HBA through which the `SCSI READ CAPACITY` command is issued

discoveredPortWWN

`HBA_ScsiInquiryV2()`	the Remote Port WWN to which the `SCSI INQUIRY` command is being sent
`HBA_ScsiReportLUNsV2()`	the Remote Port WWN to which the `SCSI REPORT LUNS` command is sent
`HBA_ScsiReadCapacityV2()`	the Remote Port WWN to which the `SCSI READ CAPACITY` command is sent

fcLUN
the FCP LUN as defined in the T10 specification SAM-2 to which the command is sent

EVPD
If set to 0, indicates a Standard Inquiry should be returned. If set to 1, indicates Vital Product Data should be returned.

PageCode
If *EVPD* is set to 1, *PageCode* indicates which Vital Product Data page should be returned.

CDB_Byte1
the second byte of the CDB for the `SCSI INQUIRY` command

CDB_Byte2
the third byte of the CDB for the `SCSI INQUIRY` command

pRspBuffer
a buffer in which to store the response payload

RspBufferSize
the size of the response buffer

pRspBufferSize
a pointer to the size of the response buffer

pScsiStatus
a buffer to receive SCSI sense data

pSenseBuffer
a buffer in which to store any SCSI sense data

SenseBufferSize
the size of the sense buffer

pSenseBufferSize
a pointer to the size of the sense buffer

Description The `HBA_SendScsiInquiry()` and `HBA_SendScsiInquiryV2()` functions send a SCSI Inquiry command as defined in the T10 specification SPC-2 to a remote FCP port.

The `HBA_SendReportLUNs()` and `HBA_SendReportLUNsV2()` functions send a SCSI Report LUNs command as defined in the T10 specification SPC-2 to a remote FCP port.

The `HBA_SendReadCapacity()` and `HBA_SendReadCapacityV2()` functions end a SCSI Read Capacity command as defined in the T10 specification SBC-2 to a remote FCP port.

Return Values The `HBA_SendScsiInquiry()` function returns the following value:

`HBA_STATUS_OK`
The command has completed. Success or failure should be determined by verifying that the sense data does not contain a check-condition. If a check-condition is present, the content of *pRspBuffer* is undefined.

The `HBA_ScsiInquiryV2()` function returns the following values:

`HBA_STATUS_OK`
The command has completed. The complete payload of the `SCSI INQUIRY` command is returned in *pRspBuffer*.

`HBA_STATUS_ERROR_ILLEGAL_WWN`
The port WWN *hbaPortWWN* is not a WWN contained by the HBA specified by *handle*.

`HBA_STATUS_ERROR_NOT_A_TARGET`
The identified remote Port does not have SCSI Target functionality.

`HBA_STATUS_ERROR_TARGET_BUSY`
The command cannot be sent due to a SCSI overlapped command condition.

`HBA_STATUS_ERROR`
An error occurred.

The `HBA_SendReportLUNs()` function returns the following values:

`HBA_STATUS_OK`
The command has completed. Success or failure should be determined by verifying the sense data does not contain a check-condition. If a check-condition is present, the content of *pRspBuffer* is undefined.

`HBA_STATUS_SCSI_CHECK_CONDITION`
The HBA detected a check-condition state. Details are present in the *pSenseBuffer* payload. The content of *pRspBuffer* is undefined. Not all VSLs support this error condition.

Other error values indicate the content of *pRspBuffer* is undefined. In some cases, the *pSenseBuffer* can contain sense data.

The `HBA_SendReportLUNsV2()` function returns the following values:

`HBA_STATUS_OK`
The command has completed. Sense data must be verified to ensure that it does not contain a check-condition to determine success. If a check-condition is present, the content of *pRspBuffer* is undefined.

`HBA_STATUS_ERROR_ILLEGAL_WWN`
The port WWN *hbaPortWWN* is not a WWN contained by the HBA specified by *handle*.

`HBA_STATUS_ERROR_NOT_A_TARGET`
The identified remote Port does not have SCSI Target functionality.

`HBA_STATUS_ERROR_TARGET_BUSY`
The command cannot be sent due to a SCSI overlapped command condition.

`HBA_STATUS_ERROR`
An error occurred.

The `HBA_SendReadCapacity()` function returns the following values:

`HBA_STATUS_OK`
The command has completed. Success or failure should be determined by verifying that the sense data does not contain a check-condition. If a check-condition is present, the content of *pRspBuffer* is undefined.

`HBA_STATUS_SCSI_CHECK_CONDITION`
The HBA detected a check-condition state. Details are present in the *pSenseBuffer* payload. The content of *pRspBuffer* is undefined. Not all VSLs support this error condition.

Other error values indicate the content of *pRspBuffer* is undefined. In some cases, the *pSenseBuffer* can contain sense data.

The `HBA_ScsiReadCapacityV2()` function returns the following values:

`HBA_STATUS_OK`
The command has completed. Sense data must be verified to ensure that it does not contain a check-condition to determine success. If a check-condition is present, the content of *pRspBuffer* is undefined.

`HBA_STATUS_ERROR_ILLEGAL_WWN`
The port WWN *hbaPortWWN* is not a WWN contained by the HBA specified by *handle*.

`HBA_STATUS_ERROR_NOT_A_TARGET`
The identified remote Port does not have SCSI Target functionality.

`HBA_STATUS_ERROR_TARGET_BUSY`
The command cannot be sent due to a SCSI overlapped command condition.

`HBA_STATUS_ERROR`
An error occurred.

Other error values indicate the content of *pRspBuffer* is undefined. In some cases, the *pSenseBuffer* can contain sense data.

Errors See `libhbaapi(3LIB)` for general error status values.

Examples **EXAMPLE 1** Send a SCSI inquiry to the given discovered Target port WWN.

The following example sends a SCSI inquiry to the given discovered Target port WWN.

```
memset(&inq, 0, sizeof (inq));
memset(&sense, 0, sizeof (sense));
if ((status = HBA_SendScsiInquiry(handle,
        discPortAttrs.PortWWN, 0, 0, 0, &inq,
        sizeof (inq), &sense, sizeof (sense))) !=
        HBA_STATUS_OK) {
    fprintf(stderr, "Unable to send SCSI "
        "inquiry, reason %d\n", status);
    continue;
}
printf("    Vendor: %.*s\n", 8, inq.inq_vid);
printf("    Model: %.*s\n", 16, inq.inq_pid);
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also `HBA_OpenAdapter(3HBAAPI)`, `libhbaapi(3LIB)`, `attributes(5)`

T11 FC-MI Specification

Bugs The `HBA_SendScsiInquiry()`, `HBA_SendReportLUNs()`, and `HBA_SendReadCapacity()` functions do not take a *portindex* to define through which port of a multi-ported HBA the command should be sent. The behavior on multi-ported HBAs is vendor-specific and can result in the command being sent through the first HBA port, the first HBA port the given *PortWWN* is connected to, or other vendor-specific behavior. SNIA version 2 defines `HBA_ScsiInquiryV2()`, `HBA_ScsiReportLUNs()`, and `HBA_ScsiReadCapacity()` to take a Port WWN as an argument. This fixes the bug with multi-ported HBAs in `HBA_ScsiInquiry()`, `HBA_SendReportLUNs()`, and `HBA_SendReadCapacity()`.

Name HBA_SetRNIDMgmtInfo, HBA_GetRNIDMgmtInfo, HBA_SendRNID, HBA_SendRNIDV2 – access Fibre Channel Request Node Identification Data (RNID)

Synopsis

```
cc [ flag... ] file... -lHBAAPI [ library... ]
#include <hbaapi.h>

HBA_STATUS HBA_SetRNIDMgmtInfo(HBA_HANDLE handle,
     HBA_MGMTINFO *pInfo);

HBA_STATUS HBA_GetRNIDMgmtInfo(HBA_HANDLE handle,
     HBA_MGMTINFO *pInfo);

HBA_STATUS HBA_SendRNID(HBA_HANDLE handle, HBA_WWN wwn,
     HBA_WWNTYPE wwntype, void *pRspBuffer,
     HBA_UINT32 *RspBufferSize);

HBA_STATUS HBA_SendRNIDV2(HBA_HANDLE handle, HBA_WWN hbaPortWWN,
     HBA_WWN destWWN, HBA_UINT32 destFCID,
     HBA_UINT32 NodeIdDataFormat, void *pRspBuffer,
     HBA_UINT32 *RspBufferSize);
```

Parameters

handle
an open handle returned from HBA_OpenAdapter(3HBAAPI)

pInfo

HBA_SetRNIDMgmtInfo()
a pointer to a HBA_MGMTINFO structure containing the new RNID

HBA_GetRNIDMgmtInfo()
a pointer to a HBA_MGMTINFO structure into which the RNID is copied

wwn
the discovered port WWN to which the request is sent

wwntype
deprecated

hbaPortWWN
the Port WWN of the local HBA through which to send the ELS

destWWN
the Port WWN of the remote Target to which the ELS is sent

destFCID
If *destFCID* is non-zero, *destFCID* is the address identifier of the remote target to which the ELS is sent. If *destFCID* is 0, *destFCID* is ignored.

NodeIdDataFormat
the Node Identification Data Fromat value as defined in FC-FS

pRspBuffer
A pointer to a buffer into which the RNID response is copied . The data will be in Big Endian format.

RspBufferSize
A pointer to the size of the buffer. On completion it will contain the size of the actual response payload copied into the buffer.

Description These functions access Fibre Channel Request Node Identification Data (RNID) as defined in the T11 specification FC-FS.

The `HBA_SetRNIDMgmtInfo()` function sets the RNID returned from by HBA.

The `HBA_GetRNIDMgmtInfo()` function retrieves the stored RNID from the HBA.

The `HBA_SendRNID()` function sends an RNID request to a discovered port. The Node Identification Data format is always set to 0xDF for General Topology Discovery Format as defined in the T11 specification FC-FS.

The `HBA_SendRNIDV2()` function sends an RNID request to a discovered port requesting a specified Node Identification Data format.

Return Values Upon successful completion, `HBA_SetRNIDMgmtInfo()` returns `HBA_STATUS_OK` and sets the RNID.

Upon successful completion, `HBA_GetRNIDMgmtInfo()` returns `HBA_STATUS_OK`. Otherwise, an error value is returned and the content of *pInfo* is undefined.

Upon successful completion, `HBA_SendRNID()` returns `HBA_STATUS_OK`. Otherwise, an error value is returned and the content of *pRspBuffer* is undefined.

The `HBA_SendRNIDV2()` returns the following values:

`HBA_STATUS_OK`
The RNID ELS has been successfully returned.

`HBA_STATUS_ERROR_ELS_REJECT`
The RNID ELS was rejected by the HBA Port.

`HBA_STATUS_ERROR_ILLEGAL_WWN`
The value of *hbaPortWWN* is not a valid port WWN on the specified HBA.

`HBA_STATUS_ERROR_ILLEGAL_FCID`
The *destWWN*/*destFCID* pair conflicts with a discovered Port Name/address identifier pair known by the HBA.

`HBA_STATUS_ERROR_ILLEGAL_FCID`
The N_Port WWN in the RNID response does not match *destWWN*.

`HBA_STATUS_ERROR`
An error occurred.

Errors See attributes(5) for general error status values.

Attributes See libhbaapi(3LIB) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed Standard: FC-HBA Version 4 (API version 2)
MT-Level	Safe
Standard	FC-MI 1.92 (API version 1)

See Also HBA_OpenAdapter(3HBAAPI), libhbaapi(3LIB), attributes(5)

T11 FC-MI Specification

Bugs The HBA_SetRNIDMgmtInfo() and HBA_GetRNIDMgmtInfo() functions do not take a *portindex* to define to which port of a multi-ported HBA the command should apply. The behavior on multi-ported HBAs is vendor-specific and can result in all ports being set to the same value.

The HBA_SetRNIDMgmtInfo() and HBA_GetRNIDMgmtInfo() functions allow only 0xDF (General Topology Discovery Format).

The HBA_SendRNID() function does not take a *portindex* to define through which port of a multi-ported HBA to send the command. The behavior on multi-ported HBAs is vendor-specific and can result in the command being sent through the first port.

The HBA_SendRNID() function does not take an argument to specify the Node Identification Data Format. It always assumes that 0xDF (General Topology Discovery Format) is desired. SNIA version 2 defines HBA_SendRNIDV2() to take a Port WWN and a Node Data Format. This fixes the bugs with multi-ported HBAs of allowing only0xDF (General Topology Discovery Format) in HBA_SendRNID().

Name hypot, hypotf, hypotl – Euclidean distance function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double hypot(double x, double y);

float hypotf(float x, float y);

long double hypotl(long double x, long double y);
```

Description These functions compute the length of the square root of $x^2 + y^2$ without undue overflow or underflow.

Return Values Upon successful completion, these functions return the length of the hypotenuse of a right angled triangle with sides of length x^2 and y^2.

If the correct value would cause overflow, a range error occurs and `hypot()`, `hypotf()`, and `hypotl()` return the value of the macro `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.

If *x* or *y* is ±Inf, +Inf is returned even if one of *x* or *y* is NaN.

If *x* or *y* is NaN and the other is not ±Inf, a NaN is returned.

Errors These functions will fail if:

Range Error — The result overflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the overflow floating-point exception is raised.

Usage `hypot`(*x*,*y*), `hypot`(*y*,*x*), and `hypot`(*x*, −*y*) are equivalent.

`hypot`(*x*, ±0) is equivalent to `fabs`(*x*).

These functions takes precautions against underflow and overflow during intermediate steps of the computation.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Standard	See standards(5).

See Also fabs(3M), feclearexcept(3M), fetestexcept(3M), isnan(3M), math.h(3HEAD), sqrt(3M), attributes(5), standards(5)

Name ilogb, ilogbf, ilogbl – return an unbiased exponent

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int ilogb(double x);

int ilogbf(float x);

int ilogbl(long double x);

cc [ flag... ] file... -lm [ library... ]
#include <math.h>

int ilogb(double x);

int ilogbf(float x);

int ilogbl(long double x);
```

Description These functions return the exponent part of their argument *x*. Formally, the return value is the integral part of $\log_r |x|$ as a signed integral value, for non-zero *x*, where *r* is the radix of the machine's floating point arithmetic, which is the value of `FLT_RADIX` defined in `<float.h>`.

Return Values Upon successful completion, these functions return the exponent part of *x* as a signed integer value. They are equivalent to calling the corresponding `logb(3M)` function and casting the returned value to type `int`.

If *x* is 0, the value `FP_ILOGB0` is returned. For SUSv3–conforming applications compiled with the `c99` compiler driver (see `standards(5)`), a domain error occurs.

If *x* is ±Inf, the value `INT_MAX` is returned. For SUSv3–conforming applications compiled with the `c99` compiler driver, a domain error occurs.

If *x* is NaN, the value `FP_ILOGBNAN` is returned. For SUSv3–conforming applications compiled with the `c99` compiler driver, a domain error occurs.

Errors These functions will fail if:

Domain Error — The *x* argument is zero, NaN, or ±Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception is raised.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), limits.h(3HEAD), logb(3M), math.h(3HEAD), scalb(3M), attributes(5), standards(5)

Name isencrypt – determine whether a buffer of characters is encrypted

Synopsis

```
cc  [flag]... [file]... -lgen [library]...

#include<libgen.h>

int isencrypt(const char *fbuf, size_t ninbuf);
```

Description `isencrypt()` uses heuristics to determine whether a buffer of characters is encrypted. It requires two arguments: a pointer to an array of characters and the number of characters in the buffer.

`isencrypt()` assumes that the file is not encrypted if all the characters in the first block are ASCII characters. If there are non-ASCII characters in the first *ninbuf* characters, and if the `setlocale()` `LC_CTYPE` category is set to `C` or `ascii`, `isencrypt()` assumes that the buffer is encrypted

If the `LC_CTYPE` category is set to a value other than `C` or `ascii`, then `isencrypt()` uses a combination of heuristics to determine if the buffer is encrypted. If *ninbuf* has at least 64 characters, a chi-square test is used to determine if the bytes in the buffer have a uniform distribution; if it does, then `isencrypt()` assumes the buffer is encrypted. If the buffer has less than 64 characters, a check is made for null characters and a terminating new-line to determine whether the buffer is encrypted.

Return Values If the buffer is encrypted, `1` is returned; otherwise, zero is returned.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also `setlocale(3C)`, `attributes(5)`

Notes When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

Name isfinite – test for finite value

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int isfinite(real-floating x);
```

Description The `isfinite()` macro determines whether its argument has a finite value (zero, subnormal, or normal, and not infinite or NaN). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

Return Values The `isfinite()` macro returns a non-zero value if and only if its argument has a finite value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `fpclassify(3M)`, `isinf(3M)`, `isnan(3M)`, `isnormal(3M)`, `math.h(3HEAD)`, `signbit(3M)`, `attributes(5)`, `standards(5)`

Name isgreater – test if x greater than y

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int isgreater(real-floating x, real-floating y);
```

Description The `isgreater()` macro determines whether its first argument is greater than its second argument. The value of `isgreater`(*x*, *y*) is equal to (*x*) > (*y*); however, unlike (*x*) > (*y*), `isgreater`(*x*, *y*) does not raise the invalid floating-point exception when *x* and *y* are unordered.

Return Values Upon successful completion, the `isgreater()` macro returns the value of (*x*) > (*y*).

If x or y is NaN, 0 is returned.

Errors No errors are defined.

Usage The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators can raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for quiet NaNs without suffering the invalid floating-point exception. In the `SYNOPSIS` section, `real-floating` indicates that the argument is an expression of `real-floating` type.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `isgreaterequal(3M)`, `isless(3M)`, `islessequal(3M)`, `islessgreater(3M)`, `isunordered(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name isgreaterequal – test if x greater than or equal to y

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int isgreaterequal(real-floating x, real-floating y);
```

Description The `isgreaterequal()` macro determines whether its first argument is greater than or equal to its second argument. The value of `isgreaterequal`(x, y) is equal to $(x) \geq (y)$; however, unlike $(x) \geq (y)$, `isgreaterequal`(x, y) does not raise the invalid floating-point exception when x and y are unordered.

Return Values Upon successful completion, the `isgreaterequal()` macro returns the value of $(x) \geq (y)$.

If x or y is NaN, 0 is returned.

Errors No errors are defined.

Usage The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators can raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for quiet NaNs without suffering the invalid floating-point exception. In the `SYNOPSIS` section, `real-floating` indicates that the argument is an expression of `real-floating` type.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `isgreater(3M)`, `isless(3M)`, `islessequal(3M)`, `islessgreater(3M)`, `isunordered(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name isinf – test for infinity

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int isinf(real-floating x);
```

Description The `isinf()` macro determines whether its argument value is an infinity (positive or negative). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

Return Values The `isinf()` macro returns a non-zero value if and only if its argument has an infinite value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `fpclassify(3M)`, `isfinite(3M)`, `isnan(3M)`, `isnormal(3M)`, `math.h(3HEAD)`, `signbit(3M)`, `attributes(5)`, `standards(5)`

Name isless – test if x is less than y

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int isless(real-floating x, real-floating y);
```

Description The `isless()` macro determines whether its first argument is less than its second argument. The value of `isless`(x, y) is equal to (x) < (y); however, unlike (x) < (y), `isless`(x, y) does not raise the invalid floating-point exception when x and y are unordered.

Return Values Upon successful completion, the `isless()` macro returns the value of (x) < (y).

If x or y is NaN, 0 is returned.

Errors No errors are defined.

Usage The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators can raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for quiet NaNs without suffering the invalid floating-point exception. In the `SYNOPSIS` section, `real-floating` indicates that the argument is an expression of `real-floating` type.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `isgreater(3M)`, `isgreaterequal(3M)`, `islessequal(3M)`, `islessgreater(3M)`, `isunordered(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name islessequal – test if x is less than or equal to y

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int islessequal(real-floating x, real-floating y);
```

Description The `islessequal()` macro determines whether its first argument is less than or equal to its second argument. The value of `islessequal`(x, y) is equal to $(x) \leq (y)$; however, unlike $(x) \leq (y)$, `islessequal`(x, y) does not raise the invalid floating-point exception when x and y are unordered.

Return Values Upon successful completion, the `islessequal()` macro returns the value of $(x) \leq (y)$.

If x or y is NaN, 0 is returned.

Errors No errors are defined.

Usage The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators can raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for quiet NaNs without suffering the invalid floating-point exception. In the `SYNOPSIS` section, `real-floating` indicates that the argument is an expression of `real-floating` type.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `isgreater(3M)`, `isgreaterequal(3M)`, `isless(3M)`, `islessgreater(3M)`, `isunordered(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name islessgreater – test if x is less than or greater than y

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int islessgreater(real-floating x, real-floating y);
```

Description The `islessgreater()` macro determines whether its first argument is less than or greater than its second argument. The `islessgreater`(*x*, *y*) macro is similar to (*x*) < (*y*) || (*x*) > (*y*); however, `islessgreater`(*x*, *y*) does not raise the invalid floating-point exception when x and y are unordered (nor does it evaluate *x* and *y* twice).

Return Values Upon successful completion, the `islessgreater()` macro returns the value of (*x*) < (*y*) || (*x*) > (*y*).

If *x* or *y* is NaN, 0 is returned.

Errors No errors are defined.

Usage The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators can raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for quiet NaNs without suffering the invalid floating-point exception. In the `SYNOPSIS` section, `real-floating` indicates that the argument is an expression of `real-floating` type.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `isgreater(3M)`, `isgreaterequal(3M)`, `isless(3M)`, `islessequal(3M)`, `isunordered(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name isnan – test for NaN

Synopsis

```
cc [ flag... ] file... -lm [ library... ]
#include <math.h>

int isnan(double x);

c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int isnan(real–floating x);
```

Description In C90 mode, the `isnan()` function tests whether *x* is NaN.

In C99 mode, the `isnan()` macro determines whether its argument value is NaN. First, an argument represented in a format wider than its semantic type is converted to its semantic type. The determination is then based on the type of the argument.

Return Values Both the `isnan()` function and macro return non-zero if and only if *x* is NaN.

Errors No errors are defined.

Warnings In C99 mode, the practice of explicitly supplying a prototype for `isnan()` after the line

```
#include <math.h>
```

is obsolete and will no longer work.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `fpclassify(3M)`, `isfinite(3M)`, `isinf(3M)`, `isnormal(3M)`, `math.h(3HEAD)`, `signbit(3M)`, `attributes(5)`, `standards(5)`

Name isnormal – test for a normal value

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int isnormal(real-floating x);
```

Description The `isnormal()` macro determines whether its argument value is normal (neither zero, subnormal, infinite, nor NaN). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

Return Values The `isnormal()` macro returns a non-zero value if and only if its argument has a normal value.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `fpclassify(3M)`, `isfinite(3M)`, `isinf(3M)`, `isnan(3M)`, `math.h(3HEAD)`, `signbit(3M)`, `attributes(5)`, `standards(5)`

Name isunordered – test if arguments are unordered

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int isunordered(real-floating x, real-floating y);
```

Description The `isunordered()` macro determines whether its arguments are unordered.

Return Values Upon successful completion, the `isunordered()` macro returns 1 if its arguments are unordered and 0 otherwise.

Errors No errors are defined.

Usage The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators can raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for quiet NaNs without suffering the invalid floating-point exception. In the `SYNOPSIS` section, `real-floating` indicates that the argument shall be an expression of `real-floating` type.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `isgreater(3M)`, `isgreaterequal(3M)`, `isless(3M)`, `islessequal(3M)`, `islessgreater(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name it_config_load, it_config_commit, it_config_setprop, it_config_free – set and retrieve configuration data for the iSCSI Target Port Provider

Synopsis

```
cc [ flag... ] file... -liscsit [ library... ]
#include <libiscsit.h>

int it_config_load(it_config_t **cfg);

int it_config_commit(it_config_t *cfg);

int it_config_setprop(it_config_t *cfg, nvlist_t *proplist,
     nvlist_t **errlist);

void it_config_free(it_config_t *cfg);
```

Parameters

cfg a pointer to the iSCSI configuration structure

proplist a pointer to an `nvlist_t` containing the global properties to be set

errlist an optional pointer to an `nvlist_t` that will be used to store specific errors (if any) when validating global properties

Description The `it_config_load()` function allocates and creates an `it_config_t` structure representing the current iSCSI configuration. This structure is compiled using the "provider" data returned by `stmfGetProviderData(3STMF)`. If there is no provider data associated with iscsit, the `it_config_t` structure is set to a default configuration.

The `it_config_commit()` function informs the iscsit service that the configuration has changed and commits the new configuration to the persistent store by calling `stmfSetProviderData(3STMF)`. This function can be called multiple times during a configuration sequence, if necessary.

The `it_config_setprop()` function validates the provided property list and sets the global properties for iSCSI Target. If *errlist* is not `NULL`, this function returns detailed errors for each property that failed. The format for *errorlist* is *key* = *property*, *value* = *error* string.

The `it_config_free()` function frees resources associated with the `it_config_t` structure.

Global nvlist properties are as follows:

nvlist Key	Type	Valid Values
`alias`	`string`	any string
`auth`	`string`	`radius`, `chap`, or `none`
`isns`	`boolean`	`B_TRUE`, `B_FALSE`

nvlist Key	Type	Valid Values
isnsserver	string array	Array of portal specifications of the form IPaddress:port. Port is optional; if not specified, the default iSNS port number of 3205 will be used. IPv6 addresses should be enclosed in square brackets '[' ']'. If "none" is specified, all defined iSNS servers will be removed from the configuration.
radiusserver	string	IPaddress:port specification as described for 'isnsserver'
radiussecret	string	string of at least 12 characters but not more than 255 characters. secret will be base64 encoded when stored.

Return Values The `it_config_load()`, `it_config_commit()`, and `it_config_setprop()` functions return 0 on success and an error value on failure.

Errors The `it_config_load()`, `it_config_commit()`, and `it_config_setprop()` functions will fail if:

`EINVAL` A parameter or property is invalid.

`ENOMEM` Resources could not be allocated.

The `it_config_commit()` function will also fail if:

`STMF_ERROR_SERVICE_DATA_VERSION`
The configuration was updated by another client. See `stmfSetProviderData(3STMF)`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `it_ini_create(3ISCSIT)`, `it_portal_create(3ISCSIT)`, `it_tgt_create(3ISCSIT)`, `it_tpg_create(3ISCSIT)`, `libiscsit(3LIB)`, `libnvpair(3LIB)`, `libstmf(3LIB)`, `stmfGetProviderData(3STMF)`, `stmfSetProviderData(3STMF)`, `attributes(5)`

Name it_ini_create, it_ini_setprop, it_ini_delete, it_ini_free – create, modify and delete iSCSI Initiator Contexts

Synopsis

```
cc [ flag... ] file... -liscsit [ library... ]
#include <libiscsit.h>

int it_ini_create(it_config_t *cfg, it_ini_t **ini,
     char *ini_node_name);

int it_ini_setprop(it_ini_t *ini, nvlist_t *proplist,
     nvlist_t **errlist););

void it_ini_delete(it_config_t *cfg, it_ini_t *ini);

void it_ini_free(it_ini_t *ini);
```

Parameters

cfg	a pointer t the iSCSI configuration structure
ini	a pointer to the `it_ini_t` structure representing the initiator context
ini_node_name	the iSCSI node name of the remote initiator
proplist	a pointer to an `nvlist_t` containing the initiator properties to be set
errlist	an optional pointer to an `nvlist_t` that will be used to store specific errors (if any) when validating initiator properties

Description The `it_ini_create()` function adds an initiator context to the global configuration.

The `it_ini_setprop()` function validates the provided property list and sets the properties for the specified initiator. If *errlist* is not `NULL`, this function returns detailed errors for each property that failed. The format for *errlist* is *key* = *property*, *value* = *error* string.

The `it_ini_delete()` function removes the specified initiator context from the global configuration.

The `it_ini_free()` function deallocates resources of an `it_ini_t` structure. If *ini*→*next* is not `NULL`, this function frees all members of the list.

Configuration changes as a result of these functions are not instantiated until the modified configuration is committed by calling `it_config_commit(3ISCSIT)`.

Initiator nvlist properties are as follows:

nvlist Key	Type	Valid Values
`chapuser`	`string`	any string, or `none` to remove
`chapsecret`	`string`	string of at least 12 characters but not more than 255 characters. secret will be base64 encoded when stored.

Return Values The `it_ini_create()`, `it_ini_setprop()`, and `it_ini_delete()` functions return 0 on success and an error value on failure.

Errors The `it_ini_create()`, `it_ini_setprop()`, and `it_ini_delete()` functions will fail if:

`EEXIST` The requested initiator context is already configured.

`EINVAL` A parameter or property is invalid.

`ENOMEM` Resources could not be allocated.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `it_config_commit(3ISCSIT)`, `it_portal_create(3ISCSIT)`, `it_tgt_create(3ISCSIT)`, `it_tpg_create(3ISCSIT)`, `libiscsit(3LIB)`, `libnvpair(3LIB)`, `libstmf(3LIB)`, `stmfGetProviderData(3STMF)`, `stmfSetProviderData(3STMF)`, `attributes(5)`

Name it_portal_create, it_portal_delete – create and delete iSCSI portals

Synopsis

```
cc [ flag... ] file... -liscsit [ library... ]
#include <libiscsit.h>

int it_portal_create(it_config_t *cfg, it_tpg_t *tpg,
     it_portal_t **portal, char *portal_ip_port);

void it_portal_delete(it_config_t *cfg, it_tpg_t *tpg,
     it_portal_t *portal);
```

Parameters

cfg	a pointer to the iSCSI configuration structure
tpg	a pointer to the `it_tpg_t` structure representing the target portal group
portal	a pointer to the `it_portal_t` structure representing the portal
portal_ip_port	a string containing an appropriatedly formatted IP address:port. Both IPv4 and IPv6 addresses are permitted. IPv6 addresses should be enclosed in square brackets ('[', ']').

Description The `it_portal_create()` function adds an `it_portal_t` structure representing a new portal to the specified target portal group. A portal may belong to one and only one target portal group.

The `it_portal_delete()` function removes the specified portal from the specified target portal group.

Configuration changes as a result of these functions are not instantiated until the modified configuration is committed by calling `it_config_commit(3ISCSIT)`.

Return Values The `it_portal_create()` function returns 0 on success and an error value on failure.

Errors The `it_portal_create()` function will fail if:

`EEXIST`	The portal was already configured for another portal group.
`EINVAL`	A parameter is invalid.
`ENOMEM`	Resources could not be allocated.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also it_ini_create(3ISCSIT), it_tgt_create(3ISCSIT), it_tpg_create(3ISCSIT), libiscsit(3LIB), it_config_commit(3ISCSIT), libiscsit(3LIB), libnvpair(3LIB), libstmf(3LIB), attributes(5)

Name it_tgt_create, it_tgt_setprop, it_tgt_delete, it_tpgt_create, it_tpgt_delete, it_tgt_free, it_tpgt_free – create, modify and delete iSCSI Targets

Synopsis

```
cc [ flag... ] file... -liscsit [ library... ]
#include <libiscsit.h>

int it_tgt_create(it_config_t **cfg, it_tgt_t **tgt,
     char *tgt_name);

int it_tgt_setprop(it_config_t *cfg it_tgt_t *tgt,
     nvlist_t *proplist, nvlist_t **errlist);

int it_tgt_delete(it_config_t *cfg, it_tgt_t *tgt,
     boolean_t force);

int it_tpgt_create(it_config_t *cfg, it_tgt_t *tgt,
     it_tpgt_t **tpgt, char *tpg_name, uint16_t tpgt_tag);

void it_tpgt_delete(it_config_t *cfg, it_tgt_t *tgt,
     it_tpgt_t *tpgt);

void it_tgt_free(it_tgt_t *tgt);

void it_tpgt_free(it_tpgt_t *tpgt);
```

Parameters

cfg	a pointer to the iSCSI configuration structure
tgt	a pointer to an iSCSI target structure
tgt_name	the target node name for the target to be created. The name must be in either IQN or EUI format. If this value is `NULL`, a node name will be generated automatically in IQN format.
proplist	a pointer to an `nvlist_t` containing the target properties to be set
errlist	an optional pointer to an `nvlist_t` that will be used to store specific errors (if any) when validating target properties
force	a boolean value indicating if the target should be set to offline before removing it from the configuration. If not specified, the operation will fail if the target is determined to be online
tpgt	a pointer to a target portal group tag structure
tpgt_name	the name of the target portal group to be associated with this target portal group tag
tpgt_tag	a 16-bit numerical identifier for this target portal group tag. Valid values are 2 through 65535. If tpgt_tag is '0', `it_tpgt_create()` will assign an appropriate tag number. If *tpgt_tag* is != 0, and the requested tag number is unavailable, another value will be chosen.

Description

The `it_tgt_create()` function allocates and creates an `it_tgt_t` structure representing a new iSCSI target node. If *tgt_name* is `NULL`, then a unique target node name will be generated automatically. Otherwise, the value of *tgt_name* will be used as the target node name. The new `it_tgt_t` structure is added to the target list (*cfg_tgt_list*) in the configuration structure.

The `it_tgt_setprop()` function validates the provided property list and sets the properties for the specified target. If *errlist* is not `NULL`, this function returns detailed errors for each property that failed. The format for *errlist* is *key* = *property*, *value* = *error* string.

The `it_tgt_delete()` function removes the target represented by *tgt* from the configuration. The *tgt* argument is an existing `it_tgt_t` structure within the configuration *cfg*.

The `it_tpgt_create()` function allocates and creates an `it_tpgt_t` structure representing a new iSCSI target portal group tag. The new `it_tpgt_t` structure is added to the target *tpgt* list (*tgt_tpgt_list*) in the `it_tgt_t` structure.

The `it_tpgt_delete()` function removes the target portal group tag represented by *tpgt*, from the configuration. The *tpgt* argument is an existing `is_tpgt_t` structure within the target *tgt*.

The `it_tgt_free()` function frees an `it_tgt_t` structure. If *tgt→next* is not `NULL`, this function frees all structures in the list.

The `it_tpgt_free()` function deallocates resources of an `it_tpgt_t` structure. If *tpgt->next* is not `NULL`, this function frees all members of the list.

Configuration changes as a result of these functions are not instantiated until the modified configuration is committed by calling `it_config_commit(3ISCSIT)`.

Target nvlist properties are as follows:

nvlist Key	**Type**	**Valid Values**
`targetchapuser`	`string`	any string, or `none` to remove
`targetchapsecret`	`string`	string of at least 12 characters but not more than 255 characters. secret will be base64 encoded when stored.
`alias`	`string`	any string or `none` to remove
`auth`	`string`	`radius`, `chap`, or `none`

Return Values

The `it_tgt_create()`, `it_tgt_setprop()`, `it_tgt_delete()`, `it_tpgt_create()`, and `it_tpgt_delete()` functions return 0 on success and an error value on failure.

Errors

The `it_tgt_create()`, `it_tgt_setprop()`, `it_tgt_delete()`, `it_tpgt_create()`, and `it_tpgt_delete()` functions will fail if:

E2BIG	All tag numbers are already in use.
EBUSY	The target is online.
EEXIST	The requested target node name is already configured.
EINVAL	A parameter or property is invalid.
ENOMEM	Resources could not be allocated.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also it_config_commit(3ISCSIT), it_ini_create(3ISCSIT), it_portal_create(3ISCSIT), it_tpg_create(3ISCSIT), libiscsit(3LIB), libnvpair(3LIB), libstmf(3LIB), attributes(5)

Name it_tpg_create, it_tpg_delete, it_tpg_free – create and delete iSCSI target portal groups

Synopsis

```
cc [ flag... ] file... -liscsit [ library... ]
#include <libiscsit.h>

int it_tpg_create(it_config_t *cfg, it_tpg_t **tpg,
     char *tpg_name, char *portal_ip_port);

int it_tpg_delete(it_config_t *cfg, it_tpg_t *tpg,
     boolean_t force););

void it_tpg_free(it_tpg_t *tpg);
```

Parameters

cfg	a pointer t the iSCSI configuration structure
tpg	a pointer to the `it_tpg_t` structure representing the target portal group
tpg_name	an identifier for the target portal group
portal_ip_port	a string containing an appropriately formatted IP `address:port`. Both IPv4 and IPv6 addresses are permitted. This value becomes the first portal in the target portal group. Applications can add additional values using `it_portal_create(3ISCSIT)` before committing the target portal group. IPv6 addresses should be enclosed in square brackets ('[', ']').
force	boolean value indicating if the target portal group should be removed even if it is associated with one or more targets. If not `B_TRUE`, the operation will fail if the target product group is associated with a target.

Description The `it_tpg_create()` function allocates and creates an `it_tpg_t` structure representing a new iSCSI target portal group. The new `it_tpg_t` structure is added to the global *tpg* list (*cfg_tgt_list*) in the `it_config_t` structure.

The `it_tpg_delete()` function deletes the target portal group represented by *tpg*, where *tpg* is an existing `it_tpg_t` structure within the global configuration *cfg*.

The `it_tpg_free()` function deallocates resources associated with an `it_tpg_t` structure. If *tpg->next* is not `NULL`, this function frees all members of the list.

Configuration changes as a result of these functions are not instantiated until the modified configuration is committed by calling `it_config_commit(3ISCSIT)`.

Return Values The `it_tpg_create()` and `it_tpg_delete()` functions return 0 on success and an error value on failure.

Errors The `it_tpg_create()` and `it_tpg_delete()` functions will fail if:

`EBUSY`	The portal group is associated with one or more targets.
`EEXIST`	The portal was already configured for another portal group associated with this target.

EINVAL A parameter is invalid.

ENOMEM Resources could not be allocated.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also it_config_commit(3ISCSIT), it_ini_create(3ISCSIT), it_portal_create(3ISCSIT), it_tgt_create(3ISCSIT), libiscsit(3LIB), libnvpair(3LIB), libstmf(3LIB), attributes(5)

Name j0, j0f, j0l, j1, j1f, j1l, jn, jnf, jnl – Bessel functions of the first kind

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double j0(double x);

float j0f(float x);

long double j0l(long double x);

double j1(double x);

float j1f(float x);

long double j1l(long double x);

double jn(int n, double x);

float jnf(int n, float x);

long double jnl(int n, long double x);
```

Description These functions compute Bessel functions of *x* of the first kind of orders 0, 1 and *n* respectively.

Return Values Upon successful completion, these functions return the relevant Bessel value of *x* of the first kind.

If *x* is NaN, a NaN is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See below.

For j0(), j1(), and jn(), see standards(5).

See Also isnan(3M), y0(3M), math.h(3HEAD), attributes(5), standards(5)

Name kstat – kernel statistics facility

Description The `kstat` facility is a general-purpose mechanism for providing kernel statistics to users.

The kstat model The kernel maintains a linked list of statistics structures, or kstats. Each kstat has a common header section and a type-specific data section. The header section is defined by the `kstat_t` structure:

kstat header

```
typedef   int kid_t;    /* unique kstat id */

typedef struct kstat {
   /*
    * Fields relevant to both kernel and user
    */
   hrtime_t      ks_crtime;              /* creation time */
   struct kstat  *ks_next;               /* kstat chain linkage */
   kid_t         ks_kid;                 /* unique kstat ID */
   char          ks_module[KSTAT_STRLEN]; /* module name */
   uchar_t       ks_resv;                /* reserved */
   int           ks_instance;            /* module's instance */
   char          ks_name[KSTAT_STRLEN];  /* kstat name */
   uchar_t       ks_type;                /* kstat data type */
   char          ks_class[KSTAT_STRLEN]; /* kstat class */
   uchar_t       ks_flags;               /* kstat flags */
   void          *ks_data;               /* kstat type-specific
                                            data */
   uint_t        ks_ndata;               /* # of data records */
   size_t        ks_data_size;           /* size of kstat data
                                            section */
   hrtime_t      ks_snaptime;            /* time of last data
                                            snapshot */

   /*
    * Fields relevant to kernel only
    */
   int(*ks_update)(struct kstat *, int);
   void  *ks_private;
   int(*ks_snapshot)(struct kstat *, void *, int);
   void  *ks_lock;
} kstat_t;
```

The fields that are of significance to the user are:

`ks_crtime` The time the kstat was created. This allows you to compute the rates of various counters since the kstat was created; "rate since boot" is replaced by the more general concept of "rate since kstat creation". All times associated with kstats (such as creation time, last snapshot time, `kstat_timer_t` and `kstat_io_t` timestamps, and the like) are 64-bit nanosecond values. The accuracy of kstat timestamps is machine

dependent, but the precision (units) is the same across all platforms. See `gethrtime(3C)` for general information about high-resolution timestamps.

`ks_next` kstats are stored as a linked list, or chain. `ks_next` points to the next kstat in the chain.

`ks_kid` A unique identifier for the kstat.

`ks_module`, `ks_instance` contain the name and instance of the module that created the kstat. In cases where there can only be one instance, `ks_instance` is 0.

`ks_name` gives a meaningful name to a kstat. The full kstat namespace is <`ks_module`,`ks_instance`,`ks_name`>, so the name only need be unique within a module.

`ks_type` The type of data in this kstat. kstat data types are discussed below.

`ks_class` Each kstat can be characterized as belonging to some broad class of statistics, such as disk, tape, net, vm, and streams. This field can be used as a filter to extract related kstats. The following values are currently in use: `disk`, `tape`, `controller`, `net`, `rpc`, `vm`, `kvm`, `hat`, `streams`, `kmem`, `kmem_cache`, `kstat`, and `misc`. (The kstat class encompasses things like *kstat_types*.)

`ks_data`, `ks_ndata`, `ks_data_size` `ks_data` is a pointer to the kstat's data section. The type of data stored there depends on `ks_type`. `ks_ndata` indicates the number of data records. Only some kstat types support multiple data records. Currently, `KSTAT_TYPE_RAW`, `KSTAT_TYPE_NAMED` and `KSTAT_TYPE_TIMER` kstats support multiple data records. `KSTAT_TYPE_INTR` and `KSTAT_TYPE_IO` kstats support only one data record. `ks_data_size` is the total size of the data section, in bytes.

`ks_snaptime` The timestamp for the last data snapshot. This allows you to compute activity rates:

```
rate = (new_count - old_count) / (new_snaptime - old_snaptime);
```

kstat data types

The following types of kstats are currently available:

```
#define KSTAT_TYPE_RAW    0   /* can be anything */
#define KSTAT_TYPE_NAMED  1   /* name/value pairs */
#define KSTAT_TYPE_INTR   2   /* interrupt statistics */
#define KSTAT_TYPE_IO     3   /* I/O statistics */
#define KSTAT_TYPE_TIMER  4   /* event timers */
```

To get a list of all kstat types currently supported in the system, tools can read out the standard system kstat *kstat_types* (full name spec is <*"unix", 0, "kstat_types"*>). This is a `KSTAT_TYPE_NAMED` kstat in which the *name* field describes the type of kstat, and the *value* field is the kstat type number (for example, `KSTAT_TYPE_IO` is type 3 -- see above).

Raw kstat `KSTAT_TYPE_RAW` raw data

The "raw" kstat type is just treated as an array of bytes. This is generally used to export well-known structures, like *sysinfo*.

Name=value kstat `KSTAT_TYPE_NAMED` A list of arbitrary *name=value* statistics.

```
typedef struct kstat_named {
   char    name[KSTAT_STRLEN];    /* name of counter */
   uchar_t data_type;             /* data type */
   union {
            charc[16];            /* enough for 128-bit ints */
            struct {
               union {
                   char *ptr;    /* NULL-terminated string */
               } addr;
               uint32_t len;     /* length of string */
            } str;
            int32_t   i32;
            uint32_t  ui32;
            int64_t   i64;
            uint64_t  ui64;

  /* These structure members are obsolete */

            int32_t   l;
            uint32_t  ul;
            int64_t   ll;
            uint64_t  ull;
         } value;               /* value of counter */
} kstat_named_t;

/* The following types are Committed

KSTAT_DATA_CHAR
KSTAT_DATA_INT32
KSTAT_DATA_LONG
KSTAT_DATA_STRING
KSTAT_DATA_UINT32
KSTAT_DATA_ULONG
KSTAT_DATA_INT64
KSTAT_DATA_UINT64
```

```
/* The following types are Obsolete */

KSTAT_DATA_LONGLONG
KSTAT_DATA_ULONGLONG
KSTAT_DATA_FLOAT
KSTAT_DATA_DOUBLE
```

Some devices need to publish strings that exceed the maximum value for KSTAT_DATA_CHAR in length; KSTAT_DATA_STRING is a data type that allows arbitrary-length strings to be associated with a named kstat. The macros below are the supported means to read the pointer to the string and its length.

```
#define KSTAT_NAMED_STR_PTR(knptr) ((knptr)->value.str.addr.ptr)
#define KSTAT_NAMED_STR_BUFLEN(knptr) ((knptr)->value.str.len)
```

KSTAT_NAMED_STR_BUFLEN() returns the number of bytes required to store the string pointed to by KSTAT_NAMED_STR_PTR(); that is, strlen(KSTAT_NAMED_STR_PTR()) + 1.

Interrupt kstat KSTAT_TYPE_INTR Interrupt statistics.

An interrupt is a hard interrupt (sourced from the hardware device itself), a soft interrupt (induced by the system via the use of some system interrupt source), a watchdog interrupt (induced by a periodic timer call), spurious (an interrupt entry point was entered but there was no interrupt to service), or multiple service (an interrupt was detected and serviced just prior to returning from any of the other types).

```
#define KSTAT_INTR_HARD      0
#define KSTAT_INTR_SOFT      1
#define KSTAT_INTR_WATCHDOG  2
#define KSTAT_INTR_SPURIOUS  3
#define KSTAT_INTR_MULTSVC   4
#define KSTAT_NUM_INTRS      5

typedef struct kstat_intr {
   uint_t intrs[KSTAT_NUM_INTRS]; /* interrupt counters */
} kstat_intr_t;
```

Event timer kstat KSTAT_TYPE_TIMER Event timer statistics.

These provide basic counting and timing information for any type of event.

```
typedef struct kstat_timer {
   char         name[KSTAT_STRLEN]; /* event name */
   uchar_t      resv;               /* reserved */
   u_longlong_t num_events;         /* number of events */
   hrtime_t     elapsed_time;       /* cumulative elapsed time */
   hrtime_t     min_time;           /* shortest event duration */
   hrtime_t     max_time;           /* longest event duration */
   hrtime_t     start_time;         /* previous event start time */
```

```
        hrtime_t     stop_time;          /* previous event stop time */
} kstat_timer_t;
```

I/O kstat KSTAT_TYPE_IO I/O statistics.

```
typedef struct kstat_io {
/*
 * Basic counters.
 */
u_longlong_t     nread;      /* number of bytes read */
u_longlong_t     nwritten;   /* number of bytes written */
uint_t           reads;      /* number of read operations */
uint_t           writes;     /* number of write operations */
/*
* Accumulated time and queue length statistics.
*
* Time statistics are kept as a running sum of "active" time.
* Queue length statistics are kept as a running sum of the
* product of queue length and elapsed time at that length --
* that is, a Riemann sum for queue length integrated against time.
*
*              ^
*              |                                _________
*              8                               | i4    |
*              |                               |       |
*       Queue  6                               |       |
*       Length |       _________               |       |
*              4       | i2    |_______|       |       |
*              |       |       i3              |       |
*              2_______|                       |       |
*              |    i1                         |       |
*              |_______________________________|
*              Time->  t1      t2      t3      t4
*
* At each change of state (entry or exit from the queue),
* we add the elapsed time (since the previous state change)
* to the active time if the queue length was non-zero during
* that interval; and we add the product of the elapsed time
* times the queue length to the running length*time sum.
*
* This method is generalizable to measuring residency
* in any defined system: instead of queue lengths, think
* of "outstanding RPC calls to server X".
*
* A large number of I/O subsystems have at least two basic
* "lists" of transactions they manage: one for transactions
* that have been accepted for processing but for which processing
* has yet to begin, and one for transactions which are actively
```

```
* being processed (but not done). For this reason, two cumulative
* time statistics are defined here: pre-service (wait) time,
* and service (run) time.
*
* The units of cumulative busy time are accumulated nanoseconds.
* The units of cumulative length*time products are elapsed time
* times queue length.
*/
hrtime_t   wtime;            /* cumulative wait (pre-service) time */
hrtime_t   wlentime;         /* cumulative wait length*time product*/
hrtime_t   wlastupdate;      /* last time wait queue changed */
hrtime_t   rtime;            /* cumulative run (service) time */
hrtime_t   rlentime;         /* cumulative run length*time product */
hrtime_t   rlastupdate;      /* last time run queue changed */
uint_t     wcnt;             /* count of elements in wait state */
uint_t     rcnt;             /* count of elements in run state */
} kstat_io_t;
```

Using libkstat

The kstat library, `libkstat`, defines the user interface (API) to the system's kstat facility.

You begin by opening libkstat with `kstat_open(3KSTAT)`, which returns a pointer to a fully initialized kstat control structure. This is your ticket to subsequent libkstat operations:

```
typedef struct kstat_ctl {
   kid_t     kc_chain_id;    /* current kstat chain ID */
   kstat_t   *kc_chain;      /* pointer to kstat chain */
   int       kc_kd;          /* /dev/kstat descriptor */
} kstat_ctl_t;
```

Only the first two fields, `kc_chain_id` and `kc_chain`, are of interest to `libkstat` clients. (*kc_kd* is the descriptor for `/dev/kstat`, the kernel statistics driver. libkstat functions are built on top of `/dev/kstat` `ioctl(2)` primitives. Direct interaction with `/dev/kstat` is strongly discouraged, since it is *not* a public interface.)

`kc_chain` points to your copy of the kstat chain. You typically walk the chain to find and process a certain kind of kstat. For example, to display all I/O kstats:

```
kstat_ctl_t    *kc;
kstat_t        *ksp;
kstat_io_t     kio;

kc = kstat_open();
for (ksp = kc->kc_chain; ksp != NULL; ksp = ksp->ks_next) {
        if (ksp->ks_type == KSTAT_TYPE_IO) {
              kstat_read(kc, ksp, &kio);
                my_io_display(kio);
        }
}
```

`kc_chain_id` is the kstat chain ID, or KCID, of your copy of the kstat chain. See `kstat_chain_update(3KSTAT)` for an explanation of KCIDs.

Files

`/dev/kstat`	kernel statistics driver
`/usr/include/kstat.h`	header
`/usr/include/sys/kstat.h`	header

See Also `ioctl(2)`, `gethrtime(3C)`, `getloadavg(3C)`, `kstat_chain_update(3KSTAT)`, `kstat_close(3KSTAT)`, `kstat_data_lookup(3KSTAT)`, `kstat_lookup(3KSTAT)`, `kstat_open(3KSTAT)`, `kstat_read(3KSTAT)`, `kstat_write(3KSTAT)`, `attributes(5)`

Name kstat_chain_update – update the kstat header chain

Synopsis

```
cc [ flag... ] file... -lkstat [ library...]
#include <kstat.h>

kid_t kstat_chain_update(kstat_ctl_t *kc);
```

Description The `kstat_chain_update()` function brings the user's kstat header chain in sync with that of the kernel. The kstat chain is a linked list of kstat headers (`kstat_t`'s) pointed to by *kc*->`kc_chain`, which is initialized by `kstat_open(3KSTAT)`. This chain constitutes a list of all kstats currently in the system.

During normal operation, the kernel creates new kstats and delete old ones as various device instances are added and removed, thereby causing the user's copy of the kstat chain to become out of date. The `kstat_chain_update()` function detects this condition by comparing the kernel's current kstat chain ID (KCID), which is incremented every time the kstat chain changes, to the user's KCID, *kc*->`kc_chain_id`. If the KCIDs match, `kstat_chain_update()` does nothing. Otherwise, it deletes any invalid kstat headers from the user's kstat chain, adds any new ones, and sets *kc*->`kc_chain_id` to the new KCID. All other kstat headers in the user's kstat chain are unmodified.

Return Values Upon successful completion, `kstat_chain_update()` returns the new KCID if the kstat chain has changed and 0 if it has not changed. Otherwise, it returns −1 and sets `errno` to indicate the error.

Errors The `kstat_chain_update()` function will fail if:

`EAGAIN` The kstat was temporarily unavailable for reading or writing.

`ENOMEM` Insufficient storage space is available.

`ENXIO` The given kstat could not be located for reading.

`EOVERFLOW` The data for the given kstat was too large to be stored in the structure.

Files `/dev/kstat` kernel statistics driver

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe with exceptions

The `kstat_chain_update()` function is MT-Safe with the exception that only one thread may actively use a `kstat_ctl_t *` value at any time. Synchronization is left to the application.

See Also `kstat(3KSTAT)`, `kstat_lookup(3KSTAT)`, `kstat_open(3KSTAT)`, `kstat_read(3KSTAT)`, `attributes(5)`

Name kstat_lookup, kstat_data_lookup – find a kstat by name

Synopsis

```
cc [ flag... ] file... -lkstat [ library...]
#include <kstat.h>

kstat_t *kstat_lookup(kstat_ctl_t *kc, char *ks_module, int ks_instance,
     char *ks_name);

void *kstat_data_lookup(kstat_t *ksp, char *name);
```

Description The `kstat_lookup()` function traverses the kstat chain, *kc*->`kc_chain`, searching for a kstat with the same *ks_module*, *ks_instance*, and *ks_name* fields; this triplet uniquely identifies a kstat. If *ks_module* is `NULL`, *ks_instance* is -1, or *ks_name* is `NULL`, those fields will be ignored in the search. For example, `kstat_lookup(kc, NULL, -1, "foo")` will find the first kstat with name “foo”.

The `kstat_data_lookup()` function searches the kstat's data section for the record with the specified *name*. This operation is valid only for those kstat types that have named data records: `KSTAT_TYPE_NAMED` and `KSTAT_TYPE_TIMER`.

Return Values The `kstat_lookup()` function returns a pointer to the requested kstat if it is found. Otherwise it returns `NULL` and sets `errno` to indicate the error.

The `kstat_data_lookup()` function returns a pointer to the requested data record if it is found. Otherwise it returns `NULL` and sets `errno` to indicate the error .

Errors The `kstat_lookup()` and `kstat_data_lookup()` functions will fail if:

`EINVAL` An attempt was made to look up data for a kstat that was not of type `KSTAT_TYPE_NAMED` or `KSTAT_TYPE_TIMER`.

`ENOENT` The requested kstat could not be found.

Files `/dev/kstat` kernel statistics driver

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe with exceptions

The `kstat_lookup()` function is MT-Safe with the exception that only one thread may actively use a `kstat_ctl_t` * value at any time. Synchronization is left to the application.

See Also kstat(3KSTAT), kstat_chain_update(3KSTAT), kstat_open(3KSTAT), kstat_read(3KSTAT), attributes(5)

Name kstat_open, kstat_close – initialize kernel statistics facility

Synopsis

```
cc[ flag... ] file... -lkstat [ library...]
#include <kstat.h>

kstat_ctl_t *kstat_open(void);

int kstat_close(kstat_ctl_t *kc);
```

Description The `kstat_open()` function initializes a kstat control structure that provides access to the kernel statistics library. It returns a pointer to this structure, which must be supplied as the *kc* argument in subsequent `libkstat` function calls.

The `kstat_close()` function frees all resources that were associated with *kc*. This is performed automatically on `exit(2)` and `execve(2)`.

Return Values Upon successful completion, `kstat_open()` returns a pointer to a kstat control structure. Otherwise, it returns `NULL`, no resources are allocated, and `errno` is set to indicate the error.

Upon successful completion, `kstat_close()` returns 0. Otherwise, −1 is returned and `errno` is set to indicate the error.

Errors The `kstat_open()` function will fail if:

`ENOMEM` Insufficient storage space is available.

`EAGAIN` The kstat was temporarily unavailable for reading or writing.

`ENXIO` The given kstat could not be located for reading.

`EOVERFLOW` The data for the given kstat was too large to be stored in the structure.

The `kstat_open()` function can also return the error values for `open(2)`.

The `kstat_close()` function can also return the error values for `close(2)`.

Files `/dev/kstat` kernel statistics driver

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	See below.

The `kstat_open()` function is Safe. The `kstat_close()` function is MT-Safe with the exception that only one thread may actively use a `kstat_ctl_t *` value at any time. Synchronization is left to the application.

See Also close(2), execve(2), open(2), exit(2), kstat(3KSTAT), kstat_chain_update(3KSTAT), kstat_lookup(3KSTAT), kstat_read(3KSTAT), attributes(5)

Name kstat_read, kstat_write – read or write kstat data

Synopsis

```
cc [ flag... ] file... -lkstat [ library... ]
#include <kstat.h>

kid_t kstat_read(kstat_ctl_t *kc, kstat_t *ksp, void *buf);

kid_t kstat_write(kstat_ctl_t *kc, kstat_t *ksp, void *buf);
```

Description The `kstat_read()` function gets data from the kernel for the kstat pointed to by *ksp*. The *ksp*->`ks_data` field is automatically allocated (or reallocated) to be large enough to hold all of the data. The *ksp*->`ks_ndata` field is set to the number of data fields, *ksp*->`ks_data_size` is set to the total size of the data, and *ksp*->`ks_snaptime` is set to the high-resolution time at which the data snapshot was taken. If *buf* is non-null, the data is copied from *ksp*->`ks_data` to *buf*.

The `kstat_write()` function writes data from *buf*, or from *ksp*->`ks_data` if *buf* is `NULL`, to the corresponding kstat in the kernel. Only the superuser can use `kstat_write()`.

Return Values Upon successful completion, `kstat_read()` and `kstat_write()` return the current kstat chain ID (KCID). Otherwise, they return −1 and set `errno` to indicate the error.

Errors The `kstat_read()` and `kstat_write()` functions will fail if:

`EACCES` An attempt was made to write to a non-writable kstat.

`EAGAIN` The kstat was temporarily unavailable for reading or writing.

`EINVAL` An attempt was made to write data to a kstat, but the number of elements or the data size does not match.

`ENOMEM` Insufficient storage space is available.

`ENXIO` The given kstat could not be located for reading or writing.

`EOVERFLOW` The data for the given kstat was too large to be stored in the structure.

`EPERM` An attempt was made to write to a kstat, but {`PRIV_SYS_CONFIG`} was not asserted in the effective privilege set.

Files `/dev/kstat` kernel statistics driver

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe with exceptions

The `kstat_read()` function is MT-Safe with the exception that only one thread may actively use a `kstat_ctl_t` * value at any time. Synchronization is left to the application.

See Also kstat(3KSTAT), kstat_chain_update(3KSTAT), kstat_lookup(3KSTAT), kstat_open(3KSTAT), attributes(5), privileges(5)

Name kvm_getu, kvm_getcmd – get the u-area or invocation arguments for a process

Synopsis

```
cc [ flag... ] file... -lkvm [ library...]
#include <kvm.h>
#include <sys/param.h>
#include <sys/user.h>
#include <sys/proc.h>

struct user *kvm_getu(kvm_t *kd, struct proc *proc);

int kvm_getcmd(kvm_t *kd, struct proc *proc, struct user *u, char ***arg,
     char ***env);
```

Description The `kvm_getu()` function reads the u-area of the process specified by *proc* to an area of static storage associated with *kd* and returns a pointer to it. Subsequent calls to `kvm_getu()` will overwrite this static area.

The *kd* argument is a pointer to a kernel descriptor returned by `kvm_open(3KVM)`. The *proc* argument is a pointer to a copy in the current process's address space of a *proc* structure, obtained, for instance, by a prior `kvm_nextproc(3KVM)` call.

The `kvm_getcmd()` function constructs a list of string pointers that represent the command arguments and environment that were used to initiate the process specified by *proc*.

The *kd* argument is a pointer to a kernel descriptor returned by `kvm_open(3KVM)`. The *u* argument is a pointer to a copy in the current process's address space of a `user` structure, obtained, for instance, by a prior `kvm_getu()` call. If *arg* is not `NULL`, the command line arguments are formed into a null-terminated array of string pointers. The address of the first such pointer is returned in *arg*. If *env* is not `NULL`, the environment is formed into a null-terminated array of string pointers. The address of the first of these is returned in *env*.

The pointers returned in *arg* and *env* refer to data allocated by `malloc()` and should be freed by a call to `free()` when no longer needed. See `malloc(3C)`. Both the string pointers and the strings themselves are deallocated when freed.

Since the environment and command line arguments might have been modified by the user process, there is no guarantee that it will be possible to reconstruct the original command at all. The `kvm_getcmd()` function will make the best attempt possible, returning −1 if the user process data is unrecognizable.

Return Values On success, `kvm_getu()` returns a pointer to a copy of the u-area of the process specified by *proc*. On failure, it returns `NULL`.

The `kvm_getcmd()` function returns 0 on success and −1 on failure. If −1 is returned, the caller still has the option of using the command line fragment that is stored in the u-area.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Unsafe

See Also kvm_nextproc(3KVM), kvm_open(3KVM), kvm_kread(3KVM), malloc(3C), libkvm(3LIB), attributes(5)

Notes On systems that support both 32-bit and 64-bit processes, the 64-bit implementation of libkvm ensures that the *arg* and env pointer arrays for kvm_getcmd() are translated to the same form as if they were 64-bit processes. Applications that wish to access the raw 32-bit stack directly can use kvm_uread(). See kvm_read(3KVM).

Name kvm_kread, kvm_kwrite, kvm_uread, kvm_uwrite – copy data to or from a kernel image or running system

Synopsis

```
cc [ flag... ] file... -lkvm [ library...]
#include <kvm.h>

ssize_t kvm_kread(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes);

ssize_t kvm_kwrite(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes);

ssize_t kvm_uread(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes);

ssize_t kvm_uwrite(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes);
```

Description The `kvm_kread()` function transfers data from the kernel address space to the address space of the process. *nbytes* bytes of data are copied from the kernel virtual address given by *addr* to the buffer pointed to by *buf*.

The `kvm_kwrite()` function is like `kvm_kread()`, except that the direction of the transfer is reversed. To use this function, the `kvm_open(3KVM)` call that returned *kd* must have specified write access.

The `kvm_uread()` function transfers data from the address space of the processes specified in the most recent `kvm_getu(3KVM)` call. *nbytes* bytes of data are copied from the user virtual address given by *addr* to the buffer pointed to by *buf*.

The `kvm_uwrite()` function is like `kvm_uread()`, except that the direction of the transfer is reversed. To use this function, the `kvm_open(3KVM)` call that returned *kd* must have specified write access. The address is resolved in the address space of the process specified in the most recent `kvm_getu(3KVM)` call.

Return Values On success, these functions return the number of bytes actually transferred. On failure, they return −1.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Unsafe

See Also `kvm_getu(3KVM)`, `kvm_nlist(3KVM)`, `kvm_open(3KVM)`, `attributes(5)`

Name kvm_nextproc, kvm_getproc, kvm_setproc – read system process structures

Synopsis

```
cc [ flag... ] file... -lkvm [ library...]
#include <kvm.h>
#include <sys/param.h>
#include <sys/time.h>
#include <sys/proc.h>

struct proc *kvm_nextproc(kvm_t *kd);

int kvm_setproc(kvm_t *kd);

struct proc *kvm_getproc(kvm_t *kd, pid_t pid);
```

Description The `kvm_nextproc()` function reads sequentially all of the system process structures from the kernel identified by *kd* (see `kvm_open(3KVM)`). Each call to `kvm_nextproc()` returns a pointer to the static memory area that contains a copy of the next valid process table entry. There is no guarantee that the data will remain valid across calls to `kvm_nextproc()`, `kvm_setproc()`, or `kvm_getproc()`. If the process structure must be saved, it should be copied to non-volatile storage.

For performance reasons, many implementations will cache a set of system process structures. Since the system state is liable to change between calls to `kvm_nextproc()`, and since the cache may contain obsolete information, there is no guarantee that every process structure returned refers to an active process, nor is it certain that all processes will be reported.

The `kvm_setproc()` function rewinds the process list, enabling `kvm_nextproc()` to rescan from the beginning of the system process table. This function will always flush the process structure cache, allowing an application to re-scan the process table of a running system.

The `kvm_getproc()` function locates the `proc` structure of the process specified by *pid* and returns a pointer to it. Although this function does not interact with the process table pointer manipulated by `kvm_nextproc()`, the restrictions regarding the validity of the data still apply.

Return Values On success, `kvm_nextproc()` returns a pointer to a copy of the next valid process table entry. On failure, it returns `NULL`.

On success, `kvm_getproc()` returns a pointer to the *proc* structure of the process specified by *pid*. On failure, it returns `NULL`.

The `kvm_setproc()` function returns 0 on success and –1 on failure.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Unsafe

See Also kvm_getu(3KVM), kvm_open(3KVM), kvm_kread(3KVM), attributes(5)

Name kvm_nlist – get entries from kernel symbol table

Synopsis

```
cc [ flag... ] file... -lkvm [ library...]
#include <kvm.h>
#include <nlist.h>

int kvm_nlist(kvm_t *kd, struct nlist *nl);
```

Description The `kvm_nlist()` function examines the symbol table from the kernel image identified by *kd* (see `kvm_open(3KVM)`) and selectively extracts a list of values and puts them in the array of `nlist` structures pointed to by *nl*. The name list pointed to by *nl* consists of an array of structures containing names, types and values. The `n_name` field of each such structure is taken to be a pointer to a character string representing a symbol name. The list is terminated by an entry with a null pointer (or a pointer to a null string) in the `n_name` field. For each entry in *nl*, if the named symbol is present in the kernel symbol table, its value and type are placed in the `n_value` and `n_type` fields. If a symbol cannot be located, the corresponding `n_type` field of *nl* is set to 0.

Return Values The `kvm_nlist()` functions returns the value of `nlist(3ELF)`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Unsafe

See Also `kvm_open(3KVM)`, `kvm_kread(3KVM)`, `nlist(3ELF)`, `attributes(5)`

Notes Although the `libkvm` API is Committed, the symbol names and data values that can be accessed through this set of interfaces are Private and are subject to ongoing change.

Name kvm_open, kvm_close – specify a kernel to examine

Synopsis

```
cc [ flag... ] file... -lkvm [ library...]
#include <kvm.h>
#include <fcntl.h>

kvm_t *kvm_open(char *namelist, char *corefile, char *swapfile, int flag,
     char *errstr);

int kvm_close(kvm_t *kd);
```

Description The `kvm_open()` function initializes a set of file descriptors to be used in subsequent calls to kernel virtual memory (VM) routines. It returns a pointer to a kernel identifier that must be used as the *kd* argument in subsequent kernel VM function calls.

The *namelist* argument specifies an unstripped executable file whose symbol table will be used to locate various offsets in *corefile*. If *namelist* is `NULL`, the symbol table of the currently running kernel is used to determine offsets in the core image. In this case, it is up to the implementation to select an appropriate way to resolve symbolic references, for instance, using `/dev/ksyms` as a default *namelist* file.

The *corefile* argument specifies a file that contains an image of physical memory, for instance, a kernel crash dump file (see `savecore(1M)`) or the special device `/dev/mem`. If *corefile* is `NULL`, the currently running kernel is accessed, using `/dev/mem` and `/dev/kmem`.

The *swapfile* argument specifies a file that represents the swap device. If both *corefile* and *swapfile* are `NULL`, the swap device of the currently running kernel is accessed. Otherwise, if *swapfile* is `NULL`, `kvm_open()` may succeed but subsequent `kvm_getu(3KVM)` function calls may fail if the desired information is swapped out.

The *flag* function is used to specify read or write access for *corefile* and may have one of the following values:

`O_RDONLY` open for reading

`O_RDWR` open for reading and writing

The *errstr* argument is used to control error reporting. If it is a null pointer, no error messages will be printed. If it is non-null, it is assumed to be the address of a string that will be used to prefix error messages generated by `kvm_open`. Errors are printed to `stderr`. A useful value to supply for *errstr* would be *argv*[0]. This has the effect of printing the process name in front of any error messages.

Applications using `libkvm` are dependent on the underlying data model of the kernel image, that is, whether it is a 32–bit or 64–bit kernel.

The data model of these applications must match the data model of the kernel in order to correctly interpret the size and offsets of kernel data structures. For example, a 32–bit application that uses the 32–bit version of the `libkvm` interfaces will fail to open a 64–bit

kernel image. Similarly, a 64–bit application that uses the 64–bit version of the `libkvm` interfaces will fail to open a 32–bit kernel image.

The `kvm_close()` function closes all file descriptors that were associated with *kd*. These files are also closed on `exit(2)` and `execve( )` (see `exec(2)`). `kvm_close()` also resets the `proc` pointer associated with `kvm_nextproc(3KVM)` and flushes any cached kernel data.

Return Values The `kvm_open()` function returns a non-null value suitable for use with subsequent kernel VM function calls. On failure, it returns `NULL` and no files are opened.

The `kvm_close()` function returns 0 on success and –1 on failure.

Files `/dev/kmem`

`/dev/ksyms`

`/dev/mem`

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Unsafe

See Also `savecore(1M)`, `exec(2)`, `exit(2)`, `pathconf(2)`, `getloadavg(3C)`, `kstat(3KSTAT)`, `kvm_getu(3KVM)`, `kvm_nextproc(3KVM)`, `kvm_nlist(3KVM)`, `kvm_kread(3KVM)`, `libkvm(3LIB)`,`sysconf(3C)`, `proc(4)`, `attributes(5)`, `lfcompile(5)`

Notes Kernel core dumps should be examined on the platform on which they were created. While a 32-bit application running on a 64-bit kernel can examine a 32-bit core dump, a 64-bit application running on a 64-bit kernel cannot examine a kernel core dump from the 32-bit system.

On 32-bit systems, applications that use `libkvm` to access the running kernel must be 32-bit applications. On systems that support both 32-bit and 64-bit applications, applications that use the `libkvm` interfaces to access the running kernel must themselves be 64-bit applications.

Although the `libkvm` API is Committed, the symbol names and data values that can be accessed through this set of interfaces are Private and are subject to ongoing change.

Applications using `libkvm` are likely to be platform- and release-dependent.

Most of the traditional uses of `libkvm` have been superseded by more stable interfaces that allow the same information to be extracted more efficiently, yet independent of the kernel data model. For examples, see `sysconf(3C)`, `proc(4)`, `kstat(3KSTAT)`, `getloadavg(3C)`, and `pathconf(2)`.

Name kvm_read, kvm_write – copy data to or from a kernel image or running system

Synopsis

```
cc [ flag... ] file... -lkvm [ library...]
#include <kvm.h>

ssize_t kvm_read(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes);

ssize_t kvm_write(kvm_t *kd, uintptr_t addr, void *buf, size_t nbytes);
```

Description The `kvm_read()` function transfers data from the kernel image specified by *kd* (see `kvm_open(3KVM)`) to the address space of the process. *nbytes* bytes of data are copied from the kernel virtual address given by *addr* to the buffer pointed to by *buf*.

The `kvm_write()` function is like `kvm_read()`, except that the direction of data transfer is reversed. To use this function, the `kvm_open(3KVM)` call that returned *kd* must have specified write access. If a user virtual address is given, it is resolved in the address space of the process specified in the most recent `kvm_getu(3KVM)` call.

Usage The `kvm_read()` and `kvm_write()` functions are obsolete and might be removed in a future release. The functions described on the `kvm_kread(3KVM)` manual page should be used instead.

Return Values On success, these functions return the number of bytes actually transferred. On failure, they return −1.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Obsolete
MT-Level	Unsafe

See Also `kvm_getu(3KVM)`, `kvm_kread(3KVM)kvm_nlist(3KVM)`, `kvm_open(3KVM)`, `attributes(5)`

Name ldexp, ldexpf, ldexpl – load exponent of a floating point number

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double ldexp(double x, int exp);

float ldexpf(float x, int exp);

long double ldexpl(long double x, int exp);
```

Description These functions computes the quantity x * 2^{exp}.

Return Values Upon successful completion, these functions return *x* multiplied by 2 raised to the power *exp*.

If these functions would cause overflow, a range error occurs and `ldexp()`, `ldexpf()`, and `ldexpl()` return ±`HUGE_VAL`, ±`HUGE_VALF`, and ±`HUGE_VALL` (according to the sign of *x*), respectively.

If *x* is NaN, a NaN is returned.

If *x* is ±0 or ±Inf, *x* is returned.

If *exp* is 0, *x* is returned.

Errors These functions will fail if:

Range Error — The result overflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the overflow floating-point exception is raised.

The `ldexp()` function sets `errno` to `ERANGE` if the result overflows.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `ldexp()`. On return, if `errno` is non-zero, an error has occurred. The `ldexpf()` and `ldexpl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Standard	See standards(5).

See Also frexp(3M), isnan(3M), modf(3M), attributes(5), standards(5)

Name lgamma, lgammaf, lgammal, lgamma_r, lgammaf_r, lgammal_r, gamma, gammaf, gammal, gamma_r, gammaf_r, gammal_r – log gamma function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

extern int signgam;

double lgamma(double x);

float lgammaf(float x);

long double lgammal(long double x);

double gamma(double x);

float gammaf(float x);

long double gammal(long double x);

double lgamma_r(double x, int *signgamp);

float lgammaf_r(float x, int *signgamp);

long double lgammal_r(long double x, int *signgamp);

double gamma_r(double x, int *signgamp);

float gammaf_r(float x, int *signgamp);

long double gammal_r(long double x, int *signgamp);
```

Description These functions return

$$\ln |\Gamma(x)|$$

where

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\,dt$$

for $x > 0$ and

$$\Gamma(x) = \pi/(\Gamma(1-x)\sin(\pi x))$$

for $x < 1$.

These functions use the external integer `signgam` to return the sign of `|~(x)` while `lgamma_r()` and `gamma_r()` use the user-allocated space addressed by `signgamp`.

Return Values Upon successful completion, these functions return the logarithmic gamma of *x*.

If *x* is a non-positive integer, a pole error occurs and these functions return `+HUGE_VAL`, `+HUGE_VALF`, and `+HUGE_VALL`, respectively.

If *x* is NaN, a NaN is returned.

If *x* is 1 or 2, +0 shall be returned.

If *x* is ±Inf, +Inf is returned.

Errors These functions will fail if:

Pole Error — The *x* argument is a negative integer or 0.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the divide-by-zero floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

In the case of `lgamma()`, do not use the expression `signgam*exp(lgamma(x))` to compute

$$g := \Gamma(x)$$

Instead compute `lgamma()` first:

```
lg = lgamma(x); g = signgam*exp(lg);
```

only after `lgamma()` has returned can `signgam` be correct. Note that `|~(x)` must overflow when *x* is large enough, underflow when $-x$ is large enough, and generate a division by 0 exception at the singularities *x* a nonpositive integer.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	See below.

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Standard	See below.

The `lgamma()`, `lgammaf()`, `lgammal()`, `gamma()`, `gammaf()`, and `gammal()` functions are Unsafe in multithreaded applications. The `lgamma_r()`, `lgammaf_r()`, `lgammal_r()`, `gamma_r()`, `gammaf_r()`, and `gammal_r()` functions are MT-Safe and should be used instead.

For `lgamma()`, `lgammaf()`, `lgammal()`, and `gamma()`, see `standards(5)`.

See Also `exp(3M)`, `feclearexcept(3M)`, `fetestexcept(3M)`, `isnan(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Notes When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

Name lgrp_affinity_get, lgrp_affinity_set – get of set lgroup affinity

Synopsis

```
cc [ flag ... ] file... -llgrp [ library ... ]
#include <sys/lgrp_user.h>

lgrp_affinity_t lgrp_affinity_get(idtype_t idtype, id_t id,
     lgrp_id_t lgrp);

int lgrp_affinity_set(idtype_t idtype, id_t id, lgrp_id_t lgrp,
     lgrp_affinity_t affinity);
```

Description The `lgrp_affinity_get()` function returns the affinity that the LWP or set of LWPs specified by the *idtype* and *id* arguments have for the given lgroup.

The `lgrp_affinity_set()` function sets the affinity that the LWP or set of LWPs specified by *idtype* and *id* have for the given lgroup. The lgroup affinity can be set to `LGRP_AFF_STRONG`, `LGRP_AFF_WEAK`, or `LGRP_AFF_NONE`.

If the *idtype* is `P_PID`, the affinity is retrieved for one of the LWPs in the process or set for all the LWPs of the process with process ID (PID) *id*. The affinity is retrieved or set for the LWP of the current process with LWP ID *id* if *idtype* is `P_LWPID`. If *id* is `P_MYID`, then the current LWP or process is specified.

The operating system uses the lgroup affinities as advice on where to run a thread and allocate its memory and factors this advice in with other constraints. Processor binding and processor sets can restrict which lgroups a thread can run on, but do not change the lgroup affinities.

Each thread can have an affinity for an lgroup in the system such that the thread will tend to be scheduled to run on that lgroup and allocate memory from there whenever possible. If the thread has affinity for more than one lgroup, the operating system will try to run the thread and allocate its memory on the lgroup for which it has the strongest affinity, then the next strongest, and so on up through some small, system-dependent number of these lgroup affinities. When multiple lgroups have the same affinity, the order of preference among them is unspecified and up to the operating system to choose. The lgroup with the strongest affinity that the thread can run on is known as its "home lgroup" (see `lgrp_home(3LGRP)`) and is usually the operating system's first choice of where to run the thread and allocate its memory.

There are different levels of affinity that can be specified by a thread for a particuliar lgroup. The levels of affinity are the following from strongest to weakest:

```
LGRP_AFF_STRONG         /* strong affinity */
LGRP_AFF_WEAK           /* weak affinity */
LGRP_AFF_NONE           /* no affinity */
```

The `LGRP_AFF_STRONG` affinity serves as a hint to the operating system that the calling thread has a strong affinity for the given lgroup. If this is the thread's home lgroup, the operating system will avoid rehoming it to another lgroup if possible. However, dynamic reconfiguration, processor offlining, processor binding, and processor set binding and

manipulation are examples of events that can cause the operating system to change the thread's home lgroup for which it has a strong affinity.

The LGRP_AFF_WEAK affinity is a hint to the operating system that the calling thread has a weak affinity for the given lgroup. If a thread has a weak affinity for its home lgroup, the operating system interpets this to mean that thread does not mind whether it is rehomed, unlike LGRP_AFF_STRONG. Load balancing, dynamic reconfiguration, processor binding, or processor set binding and manipulation are examples of events that can cause the operating system to change a thread's home lgroup for which it has a weak affinity.

The LGRP_AFF_NONE affinity signifies no affinity and can be used to remove a thread's affinity for a particuliar lgroup. Initially, each thread has no affinity to any lgroup. If a thread has no lgroup affinities set, the operating system chooses a home lgroup for the thread with no affinity set.

Return Values Upon successful completion, `lgrp_affinity_get()` returns the affinity for the given lgroup.

Upon successful completion, `lgrp_affinity_set()` return 0.

Otherwise, both functions return −1 and set `errno` to indicate the error.

Errors The `lgrp_affinity_get()` and `lgrp_affinity_set()` functions will fail if:

`EINVAL` The specified lgroup, affinity, or ID type is not valid.

`EPERM` The effective user of the calling process does not have appropriate privileges, and its real or effective user ID does not match the real or effective user ID of one of the LWPs.

`ESRCH` The specified lgroup or LWP(s) was not found.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_home(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_children – get children of given lgroup

Synopsis

```
cc [ flag... ] file... -llgrp [ library... ]
#include <sys/lgrp_user.h>

int lgrp_children(lgrp_cookie_t cookie, lgrp_id_t parent,
     lgrp_id_t *lgrp_array, uint_t lgrp_array_size);
```

Description The `lgrp_children()` function takes a *cookie* representing a snapshot of the lgroup hierarchy retrieved from `lgrp_init(3LGRP)` and returns the number of lgroups that are children of the specified lgroup. If the *lgrp_array* and *lgrp_array_size* arguments are non-null, the array is filled with as many of the children lgroup IDs as will fit, given the size of the array.

Return Values – returns the number of child lgroup IDs. Otherwise, it returns –1 and sets `errno` to indicate the error.

Errors The `lgrp_children()` function will fail if:

`EINVAL` The specified lgroup ID is not valid or the cookie is invalid.

`ESRCH` The specified lgroup ID was not found.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_init(3LGRP)`, `lgrp_nlgrps(3LGRP)`, `lgrp_parents(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_cookie_stale – determine whether snapshot of lgroup hierarchy is stale

Synopsis

```
cc [ flag ... ] file... -llgrp [ library ... ]
#include <sys/lgrp_user.h>

int lgrp_cookie_stale(lgrp_cookie_t cookie);
```

Description The `lgrp_cookie_stale()` function takes a *cookie* representing the snapshot of the lgroup hierarchy obtained from `lgrp_init(3LGRP)` and returns whether it is stale. The snapshot can become out-of-date for a number of reasons depending on its view. If the snapshot was taken with `LGRP_VIEW_OS`, changes in the lgroup hierarchy from dynamic reconfiguration, CPU on/offline, or other conditions can cause the snapshot to become out-of-date. A snapshot taken with `LGRP_VIEW_CALLER` can be affected by the caller's processor set binding and changes in its processor set itself, as well as changes in the lgroup hierarchy.

If the snapshot needs to be updated, `lgrp_fini(3LGRP)` should be called with the old cookie and `lgrp_init()` should be called to obtain a new snapshot.

Return Values Upon successful completion, `lgrp_cookie_stale()` returns whether the cookie is stale. Otherwise, it returns −1 and sets `errno` to indicate the error.

Errors The `lgrp_cookie_stale()` function will fail if:

`EINVAL` The cookie is not valid.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_init(3LGRP)`, `lgrp_fini(3LGRP)`, `lgrp_view(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_cpus – get CPU IDs contained in specified lgroup

Synopsis

```
cc [ flag... ] file... -llgrp [ library... ]
#include <sys/lgrp_user.h>

int lgrp_cpus(lgrp_cookie_t cookie, lgrp_id_t lgrp,
     processorid_t *cpuids, uint_t count, int content);
```

Description The `lgrp_cpus()` function takes a *cookie* representing a snapshot of the lgroup hierarchy obtained from `lgrp_init(3LGRP)` and returns the number of CPUs in the lgroup specified by *lgrp*. If both the *cpuids*[] argument is non-null and the count is non-zero, `lgrp_cpus()` stores up to the specified count of CPU IDs into the *cpuids*[] array.

The *content* argument should be set to one of the following values to specify whether the direct contents or everything in this lgroup should be returned:

```
LGRP_CONTENT_ALL        /* everything in this lgroup */
LGRP_CONTENT_DIRECT     /* directly contained in lgroup */
LGRP_CONTENT_HIERARCHY  /* everything within this hierarchy (for
                           compatibility only, use LGRP_CONTENT_ALL) */
```

The `LGRP_CONTENT_HIERARCHY` value can still be used, but is being replaced by `LGRP_CONTENT_ALL`.

Return Values Upon successful completion, the number of CPUs in the given lgroup is returned. Otherwise, −1 is returned and `errno` is set to indicate the error.

Errors The `lgrp_cpus()` function will fail if:

`EINVAL` The specified cookie, lgroup ID, or one of the flags is not valid.

`ESRCH` The specified lgroup ID was not found.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_init(3LGRP)`, `lgrp_mem_size(3LGRP)`, `lgrp_resources(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_fini – finished using lgroup interface

Synopsis

```
cc [ flag... ] file... -llgrp [ library... ]
#include <sys/lgrp_user.h>

int lgrp_fini(lgrp_cookie_t cookie);
```

Description The `lgrp_fini()` function takes a *cookie*, frees the snapshot of the lgroup hierarchy created by `lgrp_init(3LGRP)`, and cleans up anything else set up by `lgrp_init()`. After this function is called, any memory allocated and returned by the lgroup interface might no longer be valid and should not be used.

Return Values Upon successful completion, 0 is returned. Otherwise, −1 is returned and `errno` is set to indicate the error.

Errors The `lgrp_fini()` function will fail if:

`EINVAL` The *cookie* is not valid.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_init(3LGRP)`, `lgrp_cookie_stale(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_home – get home lgroup

Synopsis

```
cc [ flag... ] file... -llgrp [ library... ]
#include <sys/lgrp_user.h>

lgrp_id_t lgrp_home(idtype_t idtype, id_t id);
```

Description The `lgrp_home()` function returns the ID of the home lgroup for the given process or thread. A thread can have an affinity for an lgroup in the system such that the thread will tend to be scheduled to run on that lgroup and allocate memory from there whenever possible. The lgroup with the strongest affinity that the thread can run on is known as the "home lgroup" of the thread. If the thread has no affinity for any lgroup that it can run on, the operating system will choose a home for it.

The *idtype* argument should be `P_PID` to specify a process and the *id* argument should be its process ID. Otherwise, the *idtype* argument should be `P_LWPID` to specify a thread and the *id* argument should be its LWP ID. The value `P_MYID` can be used for the *id* argument to specify the current process or thread.

Return Values Upon successful completion, `lgrp_home()` returns the ID of the home lgroup of the specified process or thread. Otherwise, −1 is returned and `errno` is set to indicate the error.

Errors The `lgrp_home()` function will fail if:

`EINVAL` The ID type is not valid.

`EPERM` The effective user of the calling process does not have appropriate privileges, and its real or effective user ID does not match the real or effective user ID of one of the threads.

`ESRCH` The specified process or thread was not found.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_affinity_get(3LGRP)`, `lgrp_init(3LGRP)`, `attributes(5)`

Name lgrp_init – initialize lgroup interface

Synopsis

```
cc [ flag... ] file... -llgrp [ library... ]
#include <sys/lgrp_user.h>

lgrp_cookie_t lgrp_init(lgrp_view_t view);
```

Description The `lgrp_init()` function initializes the lgroup interface and takes a snapshot of the lgroup hierarchy with the given *view*. If the given *view* is `LGRP_VIEW_CALLER`, the snapshot contains only the resources that are available to the caller (for example, with respect to processor sets). When the *view* is `LGRP_VIEW_OS`, the snapshot contains what is available to the operating system.

Given the *view*, `lgrp_init()` returns a cookie representing this snapshot of the lgroup hierarchy. This cookie should be used with other routines in the lgroup interface needing the lgroup hierarchy. The `lgrp_fini(3LGRP)` function should be called with the cookie when it is no longer needed.

The lgroup hiearchy represents the latency topology of the machine. The hierarchy is simplified to be a tree and can be used to find the nearest resources.

The lgroup hiearchy consists of a root lgroup, which is the maximum bounding locality group of the system, contains all the CPU and memory resources of the machine, and may contain other locality groups that contain CPUs and memory within a smaller locality. The leaf lgroups contain resources within the smallest latency.

The resources of a given lgroup come directly from the lgroup itself or from leaf lgroups contained within the lgroup. Leaf lgroups directly contain their own resources and do not encapsulate any other lgroups.

The lgroup hierarchy can be used to find the nearest resources. From a given lgroup, the closest resources can be found in the lgroup itself. After that, the next nearest resources can be found in its parent lgroup, and so on until the root lgroup is reached where all the resources of the machine are located.

Return Values Upon successful completion, `lgrp_init()` returns a cookie. Otherwise it returns `LGRP_COOKIE_NONE` and sets `errno` to indicate the error.

Errors The `lgrp_init()` function will fail if:

`EINVAL` The view is not valid.

`ENOMEM` There was not enough memory to allocate the snapshot of the lgroup hierarchy.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also lgrp_children(3LGRP), lgrp_cookie_stale(3LGRP), lgrp_cpus(3LGRP), lgrp_fini(3LGRP), lgrp_mem_size(3LGRP), lgrp_nlgrps(3LGRP), lgrp_parents(3LGRP), lgrp_resources(3LGRP), lgrp_root(3LGRP), lgrp_view(3LGRP), liblgrp(3LIB), attributes(5)

Name lgrp_latency, lgrp_latency_cookie – get latency between two lgroups

Synopsis

```
cc [ flag... ] file... -llgrp [ library... ]
#include <sys/lgrp_user.h>

int lgrp_latency_cookie(lgrp_cookie_t cookie, lgrp_id_t from,
     lgrp_id_t to, lgrp_lat_between_t between);

int lgrp_latency(lgrp_id_t from, lgrp_id_t to);
```

Description The `lgrp_latency_cookie()` function takes a cookie representing a snapshot of the lgroup hierarchy obtained from `lgrp_init(3LGRP)` and returns the latency value between a hardware resource in the *from* lgroup to a hardware resource in the *to* lgroup. If *from* is the same lgroup as *to*, the latency value within that lgroup is returned.

The *between* argument should be set to the following value to specify between which hardware resources the latency should be measured:

```
LGRP_LAT_CPU_TO_MEM     /* latency from CPU to memory */
```

The latency value is defined by the operating system and is platform-specific. It can be used only for relative comparison of lgroups on the running system. It does not necessarily represent the actual latency between hardware devices, and it might not be applicable across platforms.

The `lgrp_latency()` function is similiar to the `lgrp_latency_cookie()` function, but returns the latency between the given lgroups at the given instant in time. Since lgroups can be freed and reallocated, this function might not be able to provide a consistent answer across calls. For that reason, the `lgrp_latency_cookie()` function should be used in its place.

Return Values Upon successful completion, the latency value is returned. Otherwise −1 is returned and `errno` is set to indicate the error.

Errors The `lgrp_latency_cookie()` and `lgrp_latency()` functions will fail if:

`EINVAL` The specified cookie, lgroup ID, or value given for the *between* argument is not valid.

`ESRCH` The specified lgroup ID was not found, the *from* lgroup does not contain any CPUs, or the *to* lgroup does not have any memory.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_init`(3LGRP), `lgrp_parents`(3LGRP), `lgrp_children`(3LGRP), `liblgrp`(3LIB), `attributes`(5)

Name lgrp_mem_size – return the memory size of the given lgroup

Synopsis

```
cc [ flag... ] file... -llgrp [ library... ]
#include <sys/lgrp_user.h>

lgrp_mem_size_t lgrp_mem_size(lgrp_cookie_t cookie, lgrp_id_t lgrp,
     int type, int content);
```

Description The `lgrp_mem_size()` function takes a *cookie* representing a snapshot of the lgroup hierarchy. The *cookie* was obtained by calling `lgrp_init(3LGRP)`. The `lgrp_mem_size()` function returns the memory size of the given lgroup in bytes. The *type* argument should be set to one of the following values:

```
LGRP_MEM_SZ_FREE                /* free memory */
LGRP_MEM_SZ_INSTALLED           /* installed memory */
```

The *content* argument should be set to one of the following values to specify whether the direct contents or everything in this lgroup should be returned:

```
LGRP_CONTENT_ALL        /* everything in this lgroup */
LGRP_CONTENT_DIRECT     /* directly contained in lgroup */
LGRP_CONTENT_HIERARCHY  /* everything within this hierarchy (for */
                           compatibility only, use LGRP_CONTENT_ALL) */
```

The `LGRP_CONTENT_HIERARCHY` value can still be used, but is being replaced by `LGRP_CONTENT_ALL`.

The total sizes include all the memory in the lgroup including its children, while the others reflect only the memory contained directly in the given lgroup.

Return Values Upon successful completion, the size in bytes is returned. Otherwise, −1 is returned and `errno` is set to indicate the error.

Errors The `lgrp_mem_size()` function will fail if:

`EINVAL` The specified cookie, lgroup ID, or one of the flags is not valid.

`ESRCH` The specified lgroup ID was not found.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also lgrp_init(3LGRP), lgrp_cpus(3LGRP), lgrp_resources(3LGRP), liblgrp(3LIB), attributes(5)

Name lgrp_nlgrps – get number of lgroups

Synopsis

```
cc [ flag ... ] file... -llgrp [ library ... ]
#include <sys/lgrp_user.h>

int lgrp_nlgrps(lgrp_cookie_t cookie);
```

Description The `lgrp_nlgrps()` function takes a *cookie* representing a snapshot of the lgroup hierarchy obtained from `lgrp_init(3LGRP)`. It returns the number of lgroups in the hierarchy where the number is always at least one.

Return Values Upon successful completion, `lgrp_nlgrps()` returns the number of lgroups in the system. Otherwise, it returns −1 and sets `errno` to indicate the error.

Errors The `lgrp_nlgrps()` function will fail if:

`EINVAL` The *cookie* is not valid.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_children(3LGRP)`, `lgrp_init(3LGRP)`, `lgrp_parents(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_parents – get parents of given lgroup

Synopsis

```
cc [ flag ... ] file... -llgrp [ library ... ]
#include <sys/lgrp_user.h>

int lgrp_parents(lgrp_cookie_t cookie, lgrp_id_t child,
     lgrp_id_t *lgrp_array, uint_t lgrp_array_size);
```

Description The `lgrp_parents()` function takes a *cookie* representing a snapshot of the lgroup hierarchy obtained from `lgrp_init(3LGRP)` and returns the number of parent lgroups of the specified lgroup. If *lgrp_array* is non-null and the *lgrp_array_size* is non-zero, the array is filled with as many of the parent lgroup IDs as will fit given the size of the array. For the root lgroup, the number of parents returned is 0 and the *lgrp_array* argument is not filled in.

Return Values Upon successful completion, `lgrp_parents()` returns the number of parent lgroup IDs. Otherwise, –1 is returned and `errno` is set to indicate the error.

Errors The `lgrp_parents()` function will fail if:

`EINVAL` The specified cookie or lgroup ID is not valid.

`ESRCH` The specified lgroup ID was not found.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_children(3LGRP)`, `lgrp_init(3LGRP)`, `lgrp_nlgrps(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_resources – get lgroup resources of given lgroup

Synopsis

```
cc [ flag ... ] file... -llgrp [ library ... ]
#include <sys/lgrp_user.h>

int lgrp_resources(lgrp_cookie_t cookie, lgrp_id_t lgrp,
     lgrp_id_t *lgrpids, uint_t count, lgrp_rsrc_t type);
```

Description The `lgrp_resources()` function takes a cookie representing a snapshot of the lgroup hierarchy obtained from `lgrp_init(3LGRP)` and returns the number of resources in the lgroup specified by *lgrp*. The resources are represented by a set of lgroups in which each lgroup directly contains CPU and/or memory resources.

The *type* argument should be set to one of the following values to specify whether the CPU or memory resources should be returned:

```
LGRP_RSRC_CPU      /* CPU resources */
LGRP_RSRC_MEM      /* Memory resources */
```

If the *lgrpids*[] argument is non-null and the *count* argument is non-zero, `lgrp_resources()` stores up to the specified count of lgroup IDs into the *lgrpids*[] array.

Return Values Upon successful completion, `lgrp_resources()` returns the number of lgroup resources. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors The `lgrp_resources()` function will fail if:

`EINVAL` The specified cookie, lgroup ID, or type is not valid.

`ESRCH` The specified lgroup ID was not found.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_children(3LGRP)`, `lgrp_init(3LGRP)`, `lgrp_parents(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_root – return root lgroup ID

Synopsis

```
cc [ flag ... ] file... -llgrp [ library ... ]
#include <sys/lgrp_user.h>

lgrp_id_t lgrp_root(lgrp_cookie_t cookie);
```

Description The `lgrp_root()` function returns the root lgroup ID.

Return Values Upon successful completion, `lgrp_root()` returns the lgroup ID of the root lgroup. Otherwise, it returns −1 and sets `errno` to indicate the error.

Errors The `lgrp_root()` function will fail if:

`EINVAL` The *cookie* is not valid.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_children(3LGRP)`, `lgrp_init(3LGRP)`, `lgrp_nlgrps(3LGRP)`, `lgrp_parents(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_version – coordinate library and application versions

Synopsis

```
cc [ flag ... ] file... -llgrp [ library ... ]
#include <sys/lgrp_user.h>

int lgrp_version(const int version);
```

Description The `lgrp_version()` function takes an interface version number, *version*, as an argument and returns an lgroup interface version. The *version* argument should be the value of `LGRP_VER_CURRENT` bound to the application when it was compiled or `LGRP_VER_NONE` to find out the current lgroup interface version on the running system.

Return Values If *version* is still supported by the implementation, then `lgrp_version()` returns the requested version. If `LGRP_VER_NONE` is returned, the implementation cannot support the requested version. The application should be recompiled and might require further changes.

If *version* is `LGRP_VER_NONE`, `lgrp_version()` returns the current version of the library.

Examples **EXAMPLE 1** Test whether the version of the interface used by the caller is supported.

The following example tests whether the version of the interface used by the caller is supported:

```
#include <sys/lgrp_user.h>

if (lgrp_version(LGRP_VER_CURRENT) != LGRP_VER_CURRENT) {
    fprintf(stderr, "Built with unsupported lgroup interface %d\n",
        LGRP_VER_CURRENT);
    exit (1);
}
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_init(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name lgrp_view – get view of lgroup hierarchy

Synopsis

```
cc [ flag... ] file... -llgrp [ library... ]
#include <sys/lgrp_user.h>

lgrp_view_t lgrp_view(lgrp_cookie_t cookie);
```

Description The `lgrp_view()` function takes a *cookie* representing the snapshot of the lgroup hierarchy obtained from `lgrp_init(3LGRP)` and returns the snapshot's view of the lgroup hierarchy.

If the given view is `LGRP_VIEW_CALLER`, the snapshot contains only the resources that are available to the caller (such as those with respect to processor sets). When the view is `LGRP_VIEW_OS`, the snapshot contains what is available to the operating system.

Return Values Upon succesful completion, `lgrp_view()` returns the view for the snapshot of the lgroup hierarchy represented by the given cookie. Otherwise, −1 is returned and `errno` is set to indicate the error.

Errors The `lgrp_view()` function will fail if:

`EINVAL` The *cookie* is not valid.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also `lgrp_cookie_stale(3LGRP)`, `lgrp_fini(3LGRP)`, `lgrp_init(3LGRP)`, `liblgrp(3LIB)`, `attributes(5)`

Name llrint, llrintf, llrintl – round to nearest integer value using current rounding direction

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

long long llrint(double x);

long long llrintf(float x);

long long llrintl(long double x);
```

Description These functions round their argument to the nearest integer value, rounding according to the current rounding direction.

Return Values Upon successful completion, these functions return the rounded integer value.

If *x* is NaN, a domain error occurs and an unspecified value is returned.

If *x* is +Inf, a domain error occurs and an unspecified value is returned.

If *x* is -Inf, a domain error occurs and an unspecified value is returned.

If the correct value is positive and too large to represent as a `long long`, a domain error occurs and an unspecified value is returned.

If the correct value is negative and too large to represent as a `long long`, a domain error occurs and an unspecified value is returned.

Errors These functions will fail if:

Domain Error — The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception will be raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

These functions provide floating-to-integer conversions. They round according to the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and the invalid floating-point exception is raised. When they raise no other floating-point exception and the result differs from the argument, they raise the inexact floating-point exception.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), lrint(3M), math.h(3HEAD), attributes(5), standards(5)

Name llround, llroundf, llroundl – round to nearest integer value

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

long long llround(double x);

long long llroundf(float x);

long long llroundl(long double x);
```

Description These functions rounds their argument to the nearest integer value, rounding halfway cases away from 0 regardless of the current rounding direction.

Return Values Upon successful completion, these functions return the rounded integer value.

If *x* is NaN, a domain error occurs and an unspecified value is returned.

If *x* is +Inf, a domain error occurs and an unspecified value is returned.

If *x* is -Inf, a domain error occurs and an unspecified value is returned.

If the correct value is positive and too large to represent as a `long long`, a domain error occurs and an unspecified value is returned.

If the correct value is negative and too large to represent as a `long long`, a domain error occurs and an unspecified value is returned.

Errors These functions will fail if:

Domain Error — The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception will be raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

These functions differ from the `llrint(3M)` functions in that the default rounding direction for the `llround()` functions round halfway cases away from 0 and need not raise the inexact floating-point exception for non-integer arguments that round to within the range of the return type.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), llrint(3M), lrint(3M), lround(3M), math.h(3HEAD), attributes(5), standards(5)

Name log10, log10f, log10l – base 10 logarithm function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double log10(double x);

float log10f(float x);

long double log10l(long double x);
```

Description These functions compute the base 10 logarithm of x, $\log_{10}(x)$.

Return Values Upon successful completion, `log10()` returns the base 10 logarithm of x.

If x is ±0, a pole error occurs and `log10()`, `log10f()`, and `log10l()` return −`HUGE_VAL`, −`HUGE_VALF`, and −`HUGE_VALL`, respectively.

For finite values of x that are less than 0, or if x is −Inf, a domain error occurs and a NaN is returned.

If x is NaN, a NaN is returned.

If x is 1, +0 is returned.

If x is +Inf, x is returned.

For exceptional cases, matherr(3M) tabulates the values to be returned by `log10()` as specified by SVID3 and XPG3.

Errors These functions will fail if:

Domain Error — The finite value of x is negative, or x is -Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

The `log10()` function sets `errno` to `EDOM` if the value of x is negative.

Pole Error — The value of x is 0.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the divide-by-zero floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `log10()`. On return, if `errno` is non-zero, an error has occurred. The `log10f()` and `log10l()` functions do not set `errno`.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), isnan(3M), log(3M), math.h(3HEAD), matherr(3M), pow(3M), attributes(5), standards(5)

Name log1p, log1pf, log1pl – compute natural logarithm

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double log1p(double x);

float log1pf(float x);

long double log1pl(long double x);
```

Description These functions compute $\log_e(1.0 + x)$.

Return Values Upon successful completion, these functions return the natural logarithm of $1.0 + x$.

If x is −1, a pole error occurs and `log1p()`, `log1pf()`, and `log1pl()` return −`HUGE_VAL`, −`HUGE_VALF`, and −`HUGE_VALL`, respectively.

For finite values of x that are less than −1, or if x is −Inf, a domain error occurs and a NaN is returned.

If x is NaN, a NaN is returned.

If x is ±0 or +Inf, x is returned.

For exceptional cases, matherr(3M) tabulates the values to be returned by `log1p()` as specified by SVID3 and XPG3.

Errors These functions will fail if:

Domain Error — The finite value of x is less than −1, or x is -Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

The `log1p()` function sets `errno` to `EDOM` if the value of x is less than −1.

Pole Error — The value of x is −1.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the divide-by-zero floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `log1p()`. On return, if `errno` is non-zero, an error has occurred. The `log1pf()` and `log1pl()` functions do not set `errno`.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), log(3M), math.h(3HEAD), matherr(3M), attributes(5), standards(5)

Name log2, log2f, log2l – compute base 2 logarithm functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double log2(double x);

float log2f(float x);

long double log2l(long double x);
```

Description These functions compute the base 2 logarithm of their argument *x*, `log2`(*x*).

Return Values Upon successful completion, these functions return the base 2 logarithm of *x*.

If *x* is ±0, a pole error occurs and `log2()`, `log2f()`, and `log2l()` return −`HUGE_VAL`, −`HUGE_VALF`, and −`HUGE_VALL`, respectively.

For finite values of *x* that are less than 0, or if *x* is −Inf a domain error occurs and a NaN is returned.

If *x* is NaN, a NaN is returned.

If *x* is 1, +0 is returned.

If *x* is +Inf, *x* is returned.

Errors These functions will fail if:

Domain Error — The finite value of *x* is less than 0, or *x* is −Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception is raised.

Pole Error — The value of *x* is 0.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the divide-by-zero floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept`(`FE_ALL_EXCEPT`) before calling these functions. On return, if `fetestexcept`(`FE_INVALID` | `FE_DIVBYZERO` | `FE_OVERFLOW` | `FE_UNDERFLOW`) is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), log(3M), math.h(3HEAD), attributes(5), standards(5)

Name log, logf, logl – natural logarithm function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double log(double x);

float logf(float x);

long double logl(long double x);
```

Description These functions compute the natural logarithm of their argument x, $\log_e(x)$.

Return Values Upon successful completion, `log()` returns the natural logarithm of x.

If x is ±0, a pole error occurs and `log()`, `logf()`, and `logl()` return −`HUGE_VAL`, −`HUGE_VALF`, and −`HUGE_VALL`, respectively.

For finite values of x that are less than 0, or if x is −Inf, a domain error occurs and a NaN is returned.

If x is NaN, a NaN is returned.

If x is 1, +0 is returned.

If x is +Inf, x is returned.

For exceptional cases, `matherr(3M)` tabulates the values to be returned by `log()` as specified by SVID3 and XPG3.

Errors These functions will fail if:

Domain Error — The finite value of x is negative, or x is -Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

The `log()` function sets `errno` to `EDOM` if the value of x is negative.

Pole Error — The value of x is 0.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the divide-by-zero floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `log()`. On return, if `errno` is non-zero, an error has occurred. The `logf()` and `logl()` functions do not set `errno`.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also exp(3M), feclearexcept(3M), fetestexcept(3M), isnan(3M), log10(3M), log1p(3M), math.h(3HEAD), matherr(3M), attributes(5), standards(5)

Name logb, logbf, logbl – radix-independent exponent

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double logb(double x);

float logbf(float x);

long double logbl(long double x);

cc [ flag... ] file... -lm [ library... ]
#include <math.h>

double logb(double x);

float logbf(float x);

long double logbl(long double x);
```

Description These functions compute the exponent of x, which is the integral part of $\log_r |x|$, as a signed floating point value, for non-zero x, where r is the radix of the machine's floating-point arithmetic, which is the value of `FLT_RADIX` defined in the <`float.h`> header.

Return Values Upon successful completion, these functions return the exponent of x.

If x is subnormal:

- For SUSv3–conforming applications compiled with the `c99` compiler driver (see `standards(5)`), the exponent of x as if x were normalized is returned.
- Otherwise, if compiled with the `cc` compiler driver, −1022, −126, and −16382 are returned for `logb()`, `logbf()`, and `logbl()`, respectively.

If x is ±0, a pole error occurs and `logb()`, `logbf()`, and `logbl()` return −`HUGE_VAL`, −`HUGE_VALF`, and −`HUGE_VALL`, respectively.

If x is NaN, a NaN is returned.

If x is ±Inf, +Inf is returned.

Errors These functions will fail if:

Pole Error — The value of x is ±0.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the divide-by-zero floating-point exception is raised.

The `logb()` function sets `errno` to `EDOM` if the value of x is 0.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `logb()`. On return, if `errno` is non-zero, an error has occurred. The `logbf()` and `logbl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `feclearexcept(3M)`, `fetestexcept(3M)`, `ilogb(3M)`, `math.h(3HEAD)`, `matherr(3M)`, `scalb(3M)`, `attributes(5)`, `standards(5)`

Name lrint, lrintf, lrintl – round to nearest integer value using current rounding direction

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

long lrint(double x);

long lrintf(float x);

long lrintl(long double x);
```

Description These functions round their argument to the nearest integer value, rounding according to the current rounding direction.

Return Values Upon successful completion, these functions return the rounded integer value.

If *x* is NaN, a domain error occurs and an unspecified value is returned.

If *x* is +Inf, a domain error occurs and an unspecified value is returned.

If *x* is –Inf, a domain error occurs and an unspecified value is returned.

If the correct value is positive and too large to represent as a `long`, a domain error occurs and an unspecified value is returned.

If the correct value is negative and too large to represent as a `long`, a domain error occurs and an unspecified value is returned.

Errors These functions will fail if:

Domain Error — The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also feclearexcept(3M), fetestexcept(3M), llrint(3M), math.h(3HEAD), attributes(5), standards(5)

Name lround, lroundf, lroundl – round to nearest integer value

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

long lround(double x);

long lroundf(float x);

long lroundl(long double x);
```

Description These functions round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

Return Values Upon successful completion, these functions return the rounded integer value.

If *x* is NaN, a domain error occurs and an unspecified value is returned.

If *x* is +Inf, a domain error occurs and an unspecified value is returned.

If *x* is –Inf, a domain error occurs and an unspecified value is returned.

If the correct value is positive and too large to represent as a `long`, a domain error occurs and an unspecified value is returned.

If the correct value is negative and too large to represent as a `long`, a domain error occurs and an unspecified value is returned.

Errors These functions will fail if:

Domain Error — The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also feclearexcept(3M), fetestexcept(3M), llround(3M), math.h(3HEAD), attributes(5), standards(5)

Name maillock, mailunlock, touchlock – functions to manage lockfile(s) for user's mailbox

Synopsis

```
cc [ flag ... ] file ... -lmail [ library ... ]
#include <maillock.h>

int maillock(const char *user, int retrycnt);

void mailunlock(void);

void touchlock(void);
```

Description The `maillock()` function attempts to create a lockfile for the user's mailfile. If a lockfile already exists, and it has not been modified in the last 5 minutes, `maillock()` will remove the lockfile and set its own lockfile.

It is crucial that programs locking mail files refresh their locks at least every three minutes to maintain the lock. Refresh the lockfile by calling the `touchlock()` function with no arguments.

The algorithm used to determine the age of the lockfile takes into account clock drift between machines using a network file system. A zero is written into the lockfile so that the lock will be respected by systems running the standard version of System V.

If the lockfile has been modified in the last 5 minutes the process will sleep until the lock is available. The sleep algorithm is to sleep for 5 seconds times the attempt number. That is, the first sleep will be for 5 seconds, the next sleep will be for 10 seconds, etc. until the number of attempts reaches *retrycnt*.

When the lockfile is no longer needed, it should be removed by calling `mailunlock()`.

The *user* argument is the login name of the user for whose mailbox the lockfile will be created. `maillock()` assumes that user's mailfiles are in the "standard" place as defined in `<maillock.h>`.

Return Values Upon successful completion, `.maillock()` returns `0`. Otherwise it returns `−1`.

Files

`/var/mail/*`	user mailbox files
`/var/mail/*.lock`	user mailbox lockfiles

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

See Also libmail(3LIB),attributes(5)

Notes The `mailunlock()` function will only remove the lockfile created from the most previous call to `maillock()`. Calling `maillock()` for different users without intervening calls to `mailunlock()` will cause the initially created lockfile(s) to remain, potentially blocking subsequent message delivery until the current process finally terminates.

Name matherr – math library exception-handling function

Synopsis

```
#include <math.h>

int matherr(struct exception *exc);
```

Description The System V Interface Definition, Third Edition (SVID3) specifies that certain `libm` functions call `matherr()` when exceptions are detected. Users may define their own mechanisms for handling exceptions, by including a function named `matherr()` in their programs. The `matherr()` function is of the form described above. When an exception occurs, a pointer to the exception structure *exc* will be passed to the user-supplied `matherr()` function. This structure, which is defined in the `<math.h>` header file, is as follows:

```
struct exception {
    int type;
    char *name;
    double arg1, arg2, retval;
};
```

The `type` member is an integer describing the type of exception that has occurred, from the following list of constants (defined in the header file):

`DOMAIN`	argument domain exception
`SING`	argument singularity
`OVERFLOW`	overflow range exception
`UNDERFLOW`	underflow range exception
`TLOSS`	total loss of significance
`PLOSS`	partial loss of significance

Both `TLOSS` and `PLOSS` reflect limitations of particular algorithms for trigonometric functions that suffer abrupt declines in accuracy at definite boundaries. Since the implementation does not suffer such abrupt declines, `PLOSS` is never signaled. `TLOSS` is signaled for Bessel functions *only* to satisfy SVID3 requirements.

The `name` member points to a string containing the name of the function that incurred the exception. The `arg1` and `arg2` members are the arguments with which the function was invoked. `retval` is set to the default value that will be returned by the function unless the user's `matherr()` sets it to a different value.

If the user's `matherr()` function returns non-zero, no exception message will be printed and `errno` is not set.

Svid3 Standard Conformance When an application is built as a SVID3 conforming application (see `standards(5)`), if `matherr()` is not supplied by the user, the default matherr exception-handling mechanisms, summarized in the table below, are invoked upon exception:

`DOMAIN` 0.0 is usually returned, `errno` is set to `EDOM` and a message is usually printed on standard error.

`SING` The largest finite single-precision number, `HUGE` of appropriate sign, is returned, `errno` is set to `EDOM`, and a message is printed on standard error.

`OVERFLOW` The largest finite single-precision number, `HUGE` of appropriate sign, is usually returned and `errno` is set to `ERANGE`.

`UNDERFLOW` 0.0 is returned and `errno` is set to `ERANGE`.

`TLOSS` 0.0 is returned, `errno` is set to `ERANGE`, and a message is printed on standard error.

In general, `errno` is not a reliable error indicator because it can be unexpectedly set by a function in a handler for an asynchronous signal.

SVID3 ERROR HANDLING PROCEDURES (compile with cc \-Xt)

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
errno	EDOM	EDOM	ERANGE	ERANGE	ERANGE
IEEE Exception	Invalid Operation	Division by Zero	Overflow	Underflow	–
fp_exception_type	fp_invalid	fp_division	fp_overflow	fp_underflow	–
ACOS, ASIN (|x| > 1):	Md, 0.0	–	–	–	–
ACOSH (x < 1), ATANH (|x| > 1):	NaN	–	–	–	–
ATAN2 (0,0):	Md, 0.0	–	–	–	–
COSH, SINH:	–	–	±HUGE	–	–
EXP:	–	–	+HUGE	0.0	–
FMOD (x,0):	x	–	–	–	–
HYPOT:	–	–	+HUGE	–	–
J0, J1, JN (|x| > X_TLOSS):	–	–	–	–	Mt, 0.0
LGAMMA:					
usual cases	–	–	+HUGE	–	–
(x = 0 or –integer)	–	Ms, +HUGE	–	–	–
LOG, LOG10:					
(x < 0)	Md, –HUGE	–	–	–	–

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
(x = 0)	–	Ms, –HUGE	–	–	–
POW:					
usual cases	–	–	±HUGE	±0.0	–
(x < 0) ** (y not an integer)	Md, 0.0	–	–	–	–
0 ** 0	Md, 0.0	–	–	–	–
0 ** (y < 0)	Md, 0.0	–	–	–	
REMAINDER (x,0):	NaN	–	–	–	–
SCALB:	–	–	±HUGE_VAL	±0.0	–
SQRT (x < 0):	Md, 0.0	–	–	–	–
Y0, Y1, YN:					
(x < 0)	Md, –HUGE	–	–	–	–
(x = 0)	–	Md, –HUGE	–	–	–
(x > X_TLOSS)	–	–	–	–	Mt, 0.0

Abbreviations

Md	Message is printed (DOMAIN error).
Ms	Message is printed (SING error).
Mt	Message is printed (TLOSS error).
NaN	IEEE NaN result and invalid operation exception.
HUGE	Maximum finite single-precision floating-point number.
HUGE_VAL	IEEE ∞ result and division-by-zero exception.
X_TLOSS	The value X_TLOSS is defined in <values.h>.

The interaction of IEEE arithmetic and `matherr()` is not defined when executing under IEEE rounding modes other than the default round to nearest: `matherr()` is not always called on overflow or underflow and can return results that differ from those in this table.

X/OPEN Common Application Environment (CAE) Specifications Conformance

The X/Open System Interfaces and Headers (XSH) Issue 3 and later revisions of that specification no longer sanctions the use of the `matherr` interface. The following table summarizes the values returned in the exceptional cases. In general, XSH dictates that as long as one of the input argument(s) is a NaN, NaN is returned. In particular, `pow(NaN,0)` = NaN.

CAE SPECIFICATION ERROR HANDLING PROCEDURES (compile with cc -Xa)

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS
errno	EDOM	EDOM	ERANGE	ERANGE	ERANGE
ACOS, ASIN (|x| > 1):	0.0	–	–	–	–
ATAN2 (0,0):	0.0	–	–	–	–
COSH, SINH:	–	–	{±HUGE_VAL}	–	–
EXP:	–	–	{+HUGE_VAL}	{0.0}	–
FMOD (x,0):	{NaN}	–	–	–	–
HYPOT:	–	–	{+HUGE_VAL}	–	–
J0, J1, JN (|x| > X_TLOSS):	–	–	–	–	{0.0}
LGAMMA:					
usual cases	–	–	{+HUGE_VAL}	–	–
(x = 0 or –integer)	–	+HUGE_VAL	–	–	–
LOG, LOG10:					
(x < 0)	-HUGE_VAL	–	–	–	–
(x = 0)	–	-HUGE_VAL	–	–	–
POW:					
usual cases	–	–	±HUGE_VAL	±0.0	–
(x < 0) ** (y not an integer)	0.0	–	–	–	–
0 ** 0	{1.0}	–	–	–	–
0 ** (y < 0)	{-HUGE_VAL}	–	–	–	–
SQRT (x < 0):	0.0	–	–	–	–
Y0, Y1, YN:					
(x < 0)	{-HUGE_VAL}	–	–	–	–
(x = 0)	–	{-HUGE_VAL}	–	–	–
(x > X_TLOSS)	–	–	–	–	0.0

Abbreviations

{...}	`errno` is not to be relied upon in all braced cases.
NaN	IEEE NaN result and invalid operation exception.
HUGE_VAL	IEEE ∞ result and division-by-zero exception.
X_TLOSS	The value X_TLOSS is defined in <`values.h`>.

Ansi/ISO-C Standard Conformance

The ANSI/ISO-C standard covers a small subset of the CAE specification.

The following table summarizes the values returned in the exceptional cases.

ANSI/ISO-C ERROR HANDLING PROCEDURES (compile with cc -Xc)

<math.h> type	DOMAIN	SING	OVERFLOW	UNDERFLOW
`errno`	EDOM	EDOM	ERANGE	ERANGE
ACOS, ASIN (|x| > 1):	0.0	–	–	–
ATAN2 (0,0):	0.0	–	–	–
EXP:	–	–	+HUGE_VAL	0.0
FMOD (x,0):	NaN	–	–	–
LOG, LOG10:				
(x < 0)	`-HUGE_VAL`	–	–	–
(x = 0)	–	`-HUGE_VAL`	–	–
POW:				
usual cases	–	–	±HUGE_VAL	±0.0
(x < 0) ** (y not an integer)	0.0	–	–	–
0 ** (y < 0)	`-HUGE_VAL`	–	–	–
SQRT (x < 0):	0.0	–	–	–

ABBREVIATIONS

NaN	IEEE NaN result and invalid operation exception.
HUGE_VAL	IEEE ∞ result and division-by-zero.

Examples

EXAMPLE 1 Example of `matherr()` function

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

int
matherr(struct exception *x) {
```

EXAMPLE 1 Example of `matherr()` function *(Continued)*

```
    switch (x->type) {
        case DOMAIN:
            /* change sqrt to return sqrt(-arg1), not NaN */
   if (!strcmp(x->name, "sqrt")) {
    x->retval = sqrt(-x->arg1);
    return (0); /* print message and set errno */
   } /* FALLTHRU */
  case SING:
   /* all other domain or sing exceptions, print message and */
   /* abort */
   fprintf(stderr, "domain exception in %s\n", x->name);
   abort( );
   break;
  }
  return (0); /* all other exceptions, execute default procedure */
 }
```

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also

`attributes(5)`, `standards(5)`

Name m_create_layout – initialize a layout object

Synopsis

```
cc [ flag... ] file... –llayout [ library... ]
#include <sys/layout.h>

LayoutObject m_create_layout(const AttrObject attrobj,
      const char*modifier);
```

Description The `m_create_layout()` function creates a `LayoutObject` associated with the locale identified by *attrobj*.

The `LayoutObject` is an opaque object containing all the data and methods necessary to perform the layout operations on context-dependent or directional characters of the locale identified by the *attrobj*. The memory for the `LayoutObject` is allocated by `m_create_layout()`. The `LayoutObject` created has default layout values. If the *modifier* argument is not `NULL`, the layout values specified by the *modifier* overwrite the default layout values associated with the locale. Internal states maintained by the layout transformation function across transformations are set to their initial values.

The *attrobj* argument is or may be an amalgam of many opaque objects. A locale object is just one example of the type of object that can be attached to an attribute object. The *attrobj* argument specifies a name that is usually associated with a locale category. If *attrobj* is `NULL`, the created `LayoutObject` is associated with the current locale as set by the `setlocale(3C)` function.

The *modifier* argument announces a set of layout values when the `LayoutObject` is created.

Return Values Upon successful completion, the `m_create_layout()` function returns a `LayoutObject` for use in subsequent calls to `m_*_layout()` functions. Otherwise the `m_create_layout()` function returns (`LayoutObject`) 0 and sets `errno` to indicate the error.

Errors The `m_create_layout()` function may fail if:

`EBADF` The attribute object is invalid or the locale asssociated with the attribute object is not available.

`EINVAL` The *modifier* string has a syntax error or it contains unknown layout values.

`EMFILE` There are `{OPEN_MAX}` file descriptors currently open in the calling process.

`ENOMEM` Insufficient storage space is available.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Standard	See standards(5).

See Also setlocale(3C), attributes(5), standards(5)

Name m_destroy_layout – destroy a layout object

Synopsis

```
cc [ flag... ] file... –llayout [ library... ]
#include <sys/layout.h>

int m_destroy_layout(const LayoutObject layoutobject);
```

Description The `m_destroy_layout()` function destroys a `LayoutObject` by deallocating the layout object and all the associated resources previously allocated by the `m_create_layout(3LAYOUT)` function.

Return Values Upon successful completion, `0` is returned. Otherwise `–1` is returned and `errno` is set to indicate the error.

Errors The `m_destroy_layout()` function may fail if:

`EBADF` The attribute object is erroneous.

`EFAULT` Errors occurred while processing the request.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `m_create_layout(3LAYOUT)`, `attributes(5)`, `standards(5)`

Name m_getvalues_layout – query layout values of a LayoutObject

Synopsis

```
cc [ flag... ] file... -llayout [ library... ]
#include <sys/layout.h>

int m_getvalues_layout(const LayoutObject
     layout_object, LayoutValues values, int *index_returned);
```

Description The `m_getvalues_layout()` function queries the current setting of layout values within a `LayoutObject`.

The *layout_object* argument specifies a `LayoutObject` returned by the `m_create_layout(3LAYOUT)` function.

The *values* argument specifies the list of layout values that are to be queried. Each value element of a `LayoutValueRec` must point to a location where the layout value is stored. That is, if the layout value is of type `T`, the argument must be of type `T*`. The values are queried from the `LayoutObject` and represent its current state.

It is the user's responsibility to manage the space allocation for the layout values queried. If the layout value name has `QueryValueSize` OR-ed to it, instead of the value of the layout value, only its size is returned. The caller can use this option to determine the amount of memory needed to be allocated for the layout values queried.

Return Values Upon successful completion, the `m_getvalues_layout()` function returns `0`. If any value cannot be queried, the index of the value causing the error is returned in *index_returned*, `-1` is returned and `errno` is set to indicate the error.

Errors The `m_getvalues_layout()` function may fail if:

`EINVAL` The layout value specified by *index_returned* is unknown, its value is invalid, or the *layout_object* argument is invalid. In the case of an invalid *layout_object* argument, the value returned in *index_returned* is `-1`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `m_create_layout(3LAYOUT)`, `attributes(5)`, `standards(5)`

Name mkdirp, rmdirp – create or remove directories in a path

Synopsis

```
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

int mkdirp(const char *path, mode_t mode);

int rmdirp(char *dir, char *dir1);
```

Description The `mkdirp()` function creates all the missing directories in *path* with *mode*. See `chmod(2)` for the values of *mode*.

The `rmdirp()` function removes directories in path *dir*. This removal begins at the end of the path and moves backward toward the root as far as possible. If an error occurs, the remaining path is stored in *dir1*.

Return Values If *path* already exists or if a needed directory cannot be created, `mkdirp()` returns `−1` and sets `errno` to one of the error values listed for `mkdir(2)`. It returns zero if all the directories are created.

The `rmdirp()` function returns `0` if it is able to remove every directory in the path. It returns `−2` if a "`.`" or "`..`" is in the path and `−3` if an attempt is made to remove the current directory. Otherwise it returns `−1`.

Examples **EXAMPLE 1** Example of creating scratch directories.

The following example creates scratch directories.

```
/* create scratch directories */
if(mkdirp("/tmp/sub1/sub2/sub3", 0755) == −1) {
    fprintf(stderr, "cannot create directory");
    exit(1);
}
chdir("/tmp/sub1/sub2/sub3");
.
.
.
/* cleanup */
chdir("/tmp");
rmdirp("sub1/sub2/sub3");
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also chmod(2), mkdir(2), rmdir(2), malloc(3C), attributes(5)

Notes The `mkdirp()` function uses malloc(3C) to allocate temporary space for the string.

Name modf, modff, modfl – decompose floating-point number

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double modf(double x, double *iptr);

float modff(float x, float *iptr);

long double modfl(long double x, long double *iptr);
```

Description These functions break the argument *x* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a `double` for the `modf()` function, a float for the `modff()` function, or a long double for the`modfl()` function in the object pointed to by *iptr*.

Return Values Upon successful completion, these functions return the signed fractional part of *x*.

If *x* is NaN, a NaN is returned and **iptr* is set to NaN.

If *x* is ±Inf, ±0 is returned and **iptr* is set to ±Inf.

Errors No errors are defined.

Usage These functions compute the function result and **iptr* such that:

```
a = modf(x, &iptr) ;
x == a+*iptr ;
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `frexp(3M)`, `isnan(3M)`, `ldexp(3M)`, `attributes(5)`, `standards(5)`

Name mp, mp_madd, mp_msub, mp_mult, mp_mdiv, mp_mcmp, mp_min, mp_mout, mp_pow, mp_gcd, mp_rpow, mp_msqrt, mp_sdiv, mp_itom, mp_xtom, mp_mtox, mp_mfree – multiple precision integer arithmetic

Synopsis

```
cc [ flag... ] file... -lmp [ library... ]
#include <mp.h>

void mp_madd(MINT *a, MINT *b, MINT *c);

void mp_msub(MINT *a, MINT *b, MINT *c);

void mp_mult(MINT *a, MINT *b, MINT *c);

void mp_mdiv(MINT *a, MINT *b, MINT *q, MINT *r);

int mp_mcmp(MINT *a, MINT *b);

int mp_min(MINT *a);

void mp_mout(MINT *a);

void mp_pow(MINT *a, MINT *b, MINT *c, MINT *d);

void mp_gcd(MINT *a, MINT *b, MINT *c);

void mp_rpow(MINT *a, short n, MINT *b);

int mp_msqrt(MINT *a, MINT *b, MINT *r);

void mp_sdiv(MINT *a, short n, MINT *q, short *r);

MINT * mp_itom(short n);

MINT * mp_xtom(char *a);

char * mp_mtox(MINT *a);

void mp_mfree(MINT *a);
```

Description These functions perform arithmetic on integers of arbitrary length. The integers are stored using the defined type `MINT`. Pointers to a `MINT` should be initialized using the function `mp_itom(`*n*`)`, which sets the initial value to *n*. Alternatively, `mp_xtom(`*a*`)` may be used to initialize a `MINT` from a string of hexadecimal digits. `mp_mfree(`*a*`)` may be used to release the storage allocated by the `mp_itom(`*a*`)` and `mp_xtom(`*a*`)` routines.

The `mp_madd(`*a*,*b*,*c*`)`, `mp_msub(`*a*,*b*,*c*`)` and `mp_mult(`*a*,*b*,*c*`)` functions assign to their third arguments the sum, difference, and product, respectively, of their first two arguments. The `mp_mdiv(`*a*,*b*,*q*,*r*`)` function assigns the quotient and remainder, respectively, to its third and fourth arguments. The `mp_sdiv(`*a*,*n*,*q*,*r*`)` function is similar to `mp_mdiv(`*a*,*b*,*q*,*r*`)` except that the divisor is an ordinary integer. The `mp_msqrt(`*a*,*b*,*r*`)` function produces the square root and remainder of its first argument. The `mp_mcmp(`*a*,*b*`)` function compares the values of its arguments and returns `0` if the two values are equal, a value greater than `0` if the first argument is greater than the second, and a value less than `0` if the second argument is greater than the first. The `mp_rpow(`*a*,*n*,*b*`)` function raises *a* to the *n*th power and assigns this value to *b*. The `mp_pow(`*a*,*b* ,*c*,*d*`)` function raises *a* to the *b*th power, reduces the result `modulo` *c* and assigns this

value to *d*. The `mp_min(`*a*`)` and `mp_mout(`*a*`)` functions perform decimal input and output. The `mp_gcd(`*a*,*b*,*c*`)` function finds the greatest common divisor of the first two arguments, returning it in the third argument. The `mp_mtox(` *a*`)` function provides the inverse of `mp_xtom(a)`. To release the storage allocated by `mp_mtox(`*a*`)` use `free()` (see `malloc(3C)`).

Use the `-lmp` loader option to obtain access to these functions.

Files `/usr/lib/libmp.so` shared object

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

See Also `exp(3M)`, `malloc(3C)`, `libmp(3LIB)`, `attributes(5)`

Diagnostics Illegal operations and running out of memory produce messages and core images.

Warnings The function `pow()` exists in both `libmp` and `libm` with widely differing semantics. This is the reason `libmp.so.2` exists. `libmp.so.1` exists solely for reasons of backward compatibility, and should not be used otherwise. Use the `mp_*( )` functions instead. See `libmp(3LIB)`.

Name MP_AssignLogicalUnitToTPG – assign a multipath logical unit to a target port group

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_AssignLogicalUnitToTPG(MP_OID tpgOid, MP_OID luOid);
```

Parameters

tpgOid An object ID that has type `MP_TARGET_PORT_GROUP`. The target port group currently in active access state that the administrator would like the LU assigned to.

luOid An object ID that has type `MP_MULTIPATH_LOGICAL_UNIT`.

Description The `MP_AssignLogicalUnitToTPG()` function assigns a multipath logical unit to a target port group.

Calling this function is valid only if the field `supportsLuAssignment` in the data structure `TARGET_PORT_GROUP_PROPERTIES` is true. This capability is not defined in SCSI standards. In some cases, devices support this capability through non-SCSI interfaces (such as SMI-S or SNMP). This method is only used when devices support this capability through vendor-specific means.

At any given time, each LU will typically be associated with two target port groups, one in active state and one in standby state. The result of this API will be that the LU associations change to a different pair of target port groups. The caller should specify the object ID of the desired target port group in active access state.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE` The *tpgOid* or *luOid* parameter does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_INVALID_PARAMETER` The *tpgOid* parameter has a type subfield other than `MP_OBJECT_TYPE_TARGET_PORT_GROUP` or *luOid* has a type subfield other than `MP_OBJECT_TYPE_MULTIPATH_LU`.

`MP_STATUS_OBJECT_NOT_FOUND` The *tpgOid* or *luOid* owner ID or object sequence number is invalid.

`MP_STATUS_UNSUPPORTED` The API is not supported.

`MP_STATUS_SUCCESS` The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), MP_GetAssociatedTPGOidList(3MPAPI), MP_GetMPLuOidListFromTPG(3MPAPI), attributes(5)

Name MP_CancelOverridePath – cancel a path override

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_CancelOverridePath(MP_OID logicalUnitOid);
```

Parameters *logicalUnitOid* An object ID that has type `MP_MULTIPATH_LOGICAL_UNIT`.

Description The `MP_CancelOverridePath()` function cancels a path override and re-enables load balancing.

Calling this function is valid only if the field `canOverridePaths` in data structure `MP_PLUGIN_PROPERTIES` is true.

The previous load balance configuration and preferences in effect before the path was overridden are restored.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *logicalUnitOid* parameter does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *logicalUnitOid* parameter has a type subfield other than `MP_MULTIPATH_LOGICAL_UNIT`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *logicalUnitOid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_UNSUPPORTED`	The API is not supported.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_SetOverridePath(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_CompareOIDs – compare two object IDs

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_CompareOIDs(MP_OID oid1, MP_OID oid2);
```

Parameters

oid1 An object ID that has type MP_OIDs for two objects to compare.

oid2 An object ID that has type MP_OIDs for two objects to compare.

Description The `MP_CompareOIDs()` function compares two object IDs (OIDs) for equality to see whether they refer to the same object. The fields in the two object IDs are compared field-by-field for equality.

Return Values

`MP_STATUS_FAILED` The object IDs do not compare.

`MP_STATUS_SUCCESS` The two object IDs refer to the same object.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_DeregisterForObjectPropertyChanges – deregister a previously registered client function

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_DeregisterForObjectPropertyChanges(
     MP_OBJECT_PROPERTY_FN pClientFn, MP_OBJECT_TYPE objectType,
     MP_OID pluginOid);
```

Parameters

pClientFn A pointer to an object ID that has type `MP_OBJECT_PROPERTY_FN` function defined by the client that was previously registered using the `MP_RegisterForObjectPropertyChanges(3MPAPI)` API. With a successful return this function will no longer be called to inform the client of object property changes.

objectType The type of object the client wants to deregister for property change callbacks.

pluginOid If this is a valid plugin object ID, then registration will be removed from that plugin. If this is zero, then registration is removed for all plugins.

Description The `MP_DeregisterForObjectPropertyChanges()` function deregisters a previously registered client function that is to be invoked whenever an object's property changes.

The function specified by *pClientFn* takes a single parameter of type `MP_OBJECT_PROPERTY_FN`.

The function specified by *pClientFn* will no longer be called whenever an object's property changes.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *pluginOid* parameter does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *pluginOid* parmeter is not zero and has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *pluginOid* owner ID or object sequence number is invalid.
`MP_STATUS_UNKNOWN_FN`	The *pClientFn* parmeter is not the same as the previously registered function.
`MP_STATUS_SUCCESS`	The *pClientFn* parmeter is deregistered successfully.
`MP_STATUS_FAILED`	The *pClientFn* parmeter deregistration is not possible.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_RegisterForObjectPropertyChanges(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_DeregisterForObjectVisibilityChanges – deregister a client function

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_DeregisterForObjectVisibilityChanges(
     MP_OBJECT_VISIBILITY_FN pClientFn, MP_OBJECT_TYPE objectType,
     MP_OID pluginOid);
```

Parameters

pClientFn A pointer to an object ID that has type `MP_OBJECT_VISIBILITY_FN` function defined by the client that was previously registered using the `MP_RegisterForObjectVisibilityChanges(3MPAPI)` API. With a successful return this function will no longer be called to inform the client of object visibility changes.

objectType The type of object the client wishes to deregister for visibility change callbacks.

pluginOid If this is a valid plugin object ID, then registration will be removed from that plugin. If this is zero, then registration is removed for all plugins.

Description The `MP_DeregisterForObjectVisibilityChanges()` function deregistesr a client function to be called whenever a high level object appears or disappears.

The function specified by *pClientFn* takes a single parameter of type MP_OBJECT_VISIBILITY_FN.

The function specified by *pClientFn* will no longer be called whenever high level objects appear or disappear.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE` The *pluginOid* parameter does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_INVALID_PARAMETER` The *pluginOid* parameter is not zero or has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`.

`MP_STATUS_OBJECT_NOT_FOUND` The *pluginOid* owner ID or object sequence number is invalid.

`MP_STATUS_UNKNOWN_FN` The *pluginOid* parameter is not zero or has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`.

`MP_STATUS_SUCCESS` The *pClientFn* parameter is deregistered successfully.

`MP_STATUS_FAILED` The *pClientFn* parameter deregistration is not possible at this time.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), MP_RegisterForObjectVisibilityChanges(3MPAPI), attributes(5)

Multipath Management API Version 1.0

Name MP_DeregisterPlugin – deregister a plugin

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_DeregisterPlugin(MP_WCHAR *pPluginId);
```

Parameters

pPluginId A pointer to a Plugin ID previously registered using the `MP_RegisterPlugin(3MPAPI)` API.

Description The `MP_DeregisterPlugin()` function deregisters a plugin from the common library.

The plugin will no longer be invoked by the common library. This API does not dynamically remove the plugin from a running library instance. Instead, it prevents an application that is currently not using a plugin from accessing the plugin. This is generally the behavior expected from dynamically loaded modules.

Return Values

`MP_STATUS_INVALID_PARAMETER`	The *pPluginId* parameter is null or specifies a memory area that is not executable.
`MP_STATUS_UNKNOWN_FN`	The *pPluginId* parameter is not the same as a previously registered function.
`MP_STATUS_SUCCESS`	The *pPluginId* parameter is deregistered successfully.
`MP_STATUS_FAILED`	The *pPluginId*parameter deregistration is not possible at this time

Files `/etc/mpapi.conf` MPAPI library configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_RegisterPlugin(3MPAPI)`, `mpapi.conf(4)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_DisableAutoFailback – disable auto-failback

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_DisableAutoFailback(MP_OID oid);
```

Parameters *oid* The object ID of the plugin or the multipath logical unit.

Description The `MP_DisableAutoFailback()` function disables auto-failback for the specified plugin or multipath logical unit.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the AP I.
`MP_STATUS_INVALID_PARAMETER`	The *oid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN` or `MP_OBJECT_TYPE_MULTIPATH_LU`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_UNSUPPORTED`	The API is not supported.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_EnableAutoFailback(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_DisableAutoProbing – disable auto-probing

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_DisableAutoProbing(MP_OID oid);
```

Parameters *oid* The object ID of the plugin or the multipath logical unit.

Description The `MP_DisableAutoProbing()` function disables auto-probing for the specified plugin or multipath logical unit.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *oid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN` or `MP_OBJECT_TYPE_MULTIPATH_LU`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_UNSUPPORTED`	The API is not supported.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_EnableAutoProbing(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_DisablePath – disable a path

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_DisablePath(MP_OID oid);
```

Parameters *oid* The object ID of the path.

Description The `MP_DisablePath()` function disables a path. This API might cause failover in a logical unit with asymmetric access.

This API sets the disabled field of structure `MP_PATH_LOGICAL_UNIT_PROPERTIES` to true.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* parameter does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* parameter owner ID or object sequence number is invalid.
`MP_STATUS_INVALID_PARAMETER`	The *oid* parameter does not have a type subfield of `MP_OBJECT_TYPE_PATH_LU`.
`MP_STATUS_UNSUPPORTED`	The API is not supported.
`MP_STATUS_TRY_AGAIN`	The path cannot be disabled at this time.
`MP_STATUS_NOT_PERMITTED`	Disabling this path causes the logical unit to become unavailable. The plugin that administers the path might return this value or allow the last path to be disabled.
`MP_STATUS_SUCCESS`	The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_EnablePath(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_EnableAutoFailback – enable auto-failback

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_EnableAutoFailback(MP_OID oid);
```

Parameters *oid* The object ID of the plugin or multipath logical unit.

Description The `MP_EnableAutoFailback()` function enables auto-failback.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPEkkk`	The *oid* parameter does not specify any valid object type . This is most likely to happen if an uninitialized object ID is passed to the AP I.
`MP_STATUS_INVALID_PARAMETER`	The *oid* parameter has a type subfield other than `MP_OBJE CT_TYPE_PLUGIN` or `MP_OBJECT_TYPE_MULTIPATH_LU`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* parameter owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful
`MP_STATUS_UNSUPPORTED`	The API is not supported.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_DisableAutoFailback(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_EnableAutoProbing – enable auto-probing

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_EnableAutoProbing(MP_OID oid);
```

Parameters *oid* The object ID of the plugin or multipath logical unit.

Description The `MP_EnableAutoProbing()` function enables auto-probing.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* parameter does not specify any valid object type . This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *oid* parameter has a type subfield other than `MP_OBJE CT_TYPE_PLUGIN` or `MP_OBJECT_TYPE_MULTIPATH_LU`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* parameter owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_UNSUPPORTED`	The API is not supported.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_DisableAutoProbing(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_EnablePath – enable a path

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_EnablePath(MP_OID oid);
```

Parameters *oid* The object ID of the path.

Description The `MP_EnablePath()` function enables a path. This API might cause failover in a logical unit with asymmetric access.

This API sets the field disabled of structure `MP_PATH_LOGICAL_UNIT_PROPERTIES` to false.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *oid* has a type subfield other than `MP_OBJECT_TYPE_PATH_LU`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_UNSUPPORTED`	The API is not supported.
`MP_STATUS_TRY_AGAIN`	The path cannot be enabled at this time.
`MP_STATUS_SUCCESS`	The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_DisablePath(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_FreeOidList – free up memory

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_FreeOidList(MP_OID_LIST *pOidList);
```

Parameters

pOidList A pointer to an object ID list returned by an MP API. With a successful return, the allocated memory is freed.

The client will free all `MP_OID_LIST` structures returned by any API by using this function.

Description The `MP_FreeOidList()` function frees memory returned by an MP API.

Return Values

`MP_STATUS_INVALID_PARAMETER` The *pOidList* is null or specifies a memory area to which data cannot be written.

`MP_STATUS_SUCCESS` The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetAssociatedPathOidList – get a list of object IDs

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetAssociatedPathOidList(
     MP_OID oid, MP_OID MP_OID_LIST **ppList);
```

Parameters

oid The object ID of the multipath logical unit, initiator port, or target port.

ppList A pointer to a pointer to an object ID that has type `MP_OID_LIST` structure. With a successful return, this will contain a pointer to an object ID that has type `MP_OID_LIST` that contains the object IDs of all the paths associated with the specified (multipath) logical unit, initiator port, or target port *oid*.

Description The `MP_GetAssociatedPathOidList()` function gets a list of oid object IDs for all the path logica l units associated with the specified multipath logical unit , initiator port, or target port.

Returns a list of object IDs for all the path logical units associated with the specified mult ipath logical unit, initiator port, or target port.

When the caller is finished using the list it must free the memory used by the list by calling `MP_FreeOidList`.

Return Values

`MP_STATUS_INVALID_PARAMETER` The *ppList* is null or specifies a memory area to that the data cannot be written or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_MULTI PATH_LU`, `MP_OBJECT_TYPE_INITIATOR_PORT`, or `MP_OBJECT_TYPE_TARGET_PORT`.

`MP_STATUS_INVALID_OBJECT_TYPE` The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_OBJECT_NOT_FOUND` The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_SUCCESS` The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), MP_GetPathLogicalUnitProperties(3MPAPI), attributes(5)

Multipath Management API Version 1.0

Name MP_GetAssociatedPluginOid – get the object ID for the plugin

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetAssociatedPluginOid(MP_OID oid,
     MP_OID  *pPluginOID);
```

Parameters

oid The object ID of an object that has been received from a previous API call.

pPluginOID A pointer to an object ID that has type `MP_OID` structure allocated by the caller. With a successful return this will contain the object ID of the plugin associated with the object specified by the *oid*.

Description The `MP_GetAssociatedPluginOid()` function gets the object ID for the plugin associated with the specified object ID. The sequence number subfield of the *oid* is not validate since this API is implemented in the common library.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE` The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_INVALID_PARAMETER` The *pluginOid* is null or specifies a memory area to which data cannot be written.

`MP_STATUS_OBJECT_NOT_FOUND` The *oid* owner ID is invalid.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetAssociatedTPGOidList – get a list of the object IDs

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetAssociatedTPGOidList(MP_OID oid,
     MP_OID_LIST **ppList);
```

Parameters

oid The object ID of the multipath logical unit.

ppList A pointer to a pointer to an object ID that has type `MP_OID_LIST` structure. With a successful return, this will contain a pointer to an object ID that has type `MP_OID_LIST` that contains the object IDs of target port groups associated with the specified logical unit.

Description The `MP_GetAssociatedTPGOidList()` function gets a list of the object IDs containing the target port group associated with the specified multipath logical unit.

When the caller is finished using the list, it must free the memory used by the list by calling `MP_FreeOidList`.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *ppList* is null or specifies a memory area to which data cannot be written, or the *oid* has a type subfield other than `MP_OBJECT_TYPE_MULTIPATH_LU`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FAILED`	The target port group list for the specified object ID is not found.
`MP_STATUS_INSUFFICIENT_MEMORY`	A memory allocation failure occurred.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), MP_GetTargetPortGroupProperties(3MPAPI), attributes(5)

Multipath Management API Version 1.0

Name MP_GetDeviceProductOidList – get a list of the object IDs

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetDeviceProductOidList(MP_OID oid,
     MP_OID_LIST **ppList);
```

Parameters

oid The object ID of the plugin.

ppList A pointer to a pointer to an object ID that has type `MP_OID_LIST` structure. With a successful return, this will contain a pointer to an object ID that has type `MP_OID_LIST` that contains the object IDs of all the device product descriptors associated with the specified plugin.

Description The `MP_GetDeviceProductOidList()` function gets a list of the object IDs of all the device product properties associated with this plugin. When the caller is finished using the list, it must free the memory used by the list by calling `MP_FreeOidList`.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *ppList* is null or specifies a memory area to which data cannot be written because the *oid* has a type subfield other than`MP_OBJECT_TYPE_PLUGIN`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful
`MP_STATUS_FAILED`	The plugin for the specified object ID is not found.
`MP_STATUS_INSUFFICIENT_MEMORY`	A memory allocation failure occurred.
`MP_STATUS_UNSUPPORTED`	The API is not supported.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetDeviceProductProperties(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetDeviceProductProperties – get the properties of a specified device product

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetDeviceProductProperties(MP_OID oid,
     MP_DEVICE_PRODUCT_PROPERTIES *pProps);
```

Parameters

oid The object ID of the device product.

pProps A pointer to an object ID that has type `MP_DEVICE_PR ODUCT_PROPERTIES` structure allocated by the caller. With a successful return, this structure contains the properties of the device product specified by the *oid*.

Description The `MP_GetDeviceProductProperties()` function gets the properties of the specified device product.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_INVALID_PARAMETER`	Returned when *pProps* is null or specifies a memory area to which data cannot be written, or the *oid* has a type subfield other than `MP_OBJECT_TYPE_DEVICE_PRODUCT`.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FAILED`	The plugin for the specified *oid* is not found.
`MP_STATUS_UNSUPPORTED`	The implementation does not support the API.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetDeviceProductOidList(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetInitiatorPortOidList – gets a list of the object IDs

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

 MP_STATUS MP_GetInitiatorPortOidList(MP_OID oid,
     MP_OID_LIST **ppList);
```

Parameters

oid The object ID of the plugin.

ppList A pointer to a pointer to an object ID that has type `MP_OID_LIST` structure. With a successful return, this contains a pointer to an`MP_OID_LIST` that contains the object IDs of all the initiator ports associated with the specified plugin.

Description The `MP_GetInitiatorPortOidList()` function gets a list of the object IDs of all the initiator ports associated with this plugin. When the caller is finished using the list it must free the memory used by the list by calling `MP_FreeOidList`.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE` The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_INVALID_PARAMETER` The *pplist* is null or specifies a memory area to which data cannot be written, or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`.

`MP_STATUS_OBJECT_NOT_FOUND` The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_SUCCESS` The operation is successful.

`MP_STATUS_INSUFFICIENT_MEMORY` A memory allocation failure occurred.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetInitiatorPortProperties(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetInitiatorPortProperties – get initiator port properties

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetInitiatorPortProperties(MP_OID oid,
     MP_INITIATOR_PORT_PROPERTIES *pProps);
```

Parameters

oid The object ID of the Port.

pProps A pointer to an object ID that has type `MP_INITIATOR_PORT_PROPERTIES` structure allocated by the caller. With a successful return, this structure contains the properties of the port specified by the *oid* parameter.

Description The `MP_GetInitiatorPortProperties()` function gets the properties of the specified initiator port.

Return Values

`MP_STATUS_INVALID_PARAMETER`	The *pProps* is null or specifies a memory area to which data cannot be written, or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_INITIATOR_PORT`.
`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetInitiatorPortOidList(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetLibraryProperties – get MP library properties

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetLibraryProperties(MP_LIBRARY_PROPERTIES*pProps);
```

Parameters

pProps A pointer to an object ID that has type `MP_LIBRARY_PROPERTIES` structure allocated by the caller. With a successful return, this structure contains the properties of the MP library currently in use.

Description The `MP_GetLibraryProperties()` function gets the properties of the MP library currently in use.

Return Values

`MP_STATUS_INVALID_PARAMETER` The *pProps* is null or specifies a memory area that cannot be written to.

`MP_STATUS_SUCCESS` The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetMPLogicalUnitProperties – get logical unit properties

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetMPLogicalUnitProperties(MP_OID oid,
     MP_MULTIPATH_LOGICAL_UNIT_PROPERTIES *pProps);
```

Parameters

oid The object ID of the multipath logical unit.

pProps A pointer to an object ID that has type `MP_MULTIPATH_LOGICAL_UNIT_PROPERTIES` structure allocated by the caller. With a successful return, this structure contains the properties of the multipath logical unit specified by the object ID.

Description The `MP_GetMPLogicalUnitProperties()` function gets the properties of the specified logical unit.

Return Values

`MP_STATUS_INVALID_PARAMETER` The *pProps* is null or specifies a memory area to which data cannot be written, or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_MULTIPATH_LU`.

`MP_STATUS_INVALID_OBJECT_TYPE` The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized *oid* is passed to the API.

`MP_STATUS_OBJECT_NOT_FOUND` The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_SUCCESS` The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetMPLuOidListFromTPG(3MPAPI)`, `MP_GetMultipathLus(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetMPLuOidListFromTPG – return a list of object IDs

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetMPLuOidListFromTPG(MP_OID oid,
     MP_OID_LIST **ppList);
```

Parameters

oid The object ID of the target port group.

ppList A pointer to a pointer to an object ID that has type `MP_OID_LIST` structure. With a successful return, this contains a pointer to an object ID that has type `MP_OID_LIST` that contains the object IDs of all the (multipath) logical units associated with the specified target port group.

Description The `MP_GetMPLuOidListFromTPG()` function returns the list of object IDs for the multipath logical units associated with the specific target port group.

When the caller is finished using the list, it must free the memory used by the list by calling `MP_FreeOidList`.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *pplist* is null or specifies a memory area to which data cannot be written, or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_TARGET_PORT`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FAILED`	The multipath logical unit list for the specified target port group object ID is not found.
`MP_STATUS_INSUFFICIENT_MEMORY`	A memory allocation failure occurred.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), MP_GetMPLogicalUnitProperties(3MPAPI), attributes(5)

Multipath Management API Version 1.0

Name MP_GetMultipathLus – return a list of multipath logical units

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetMultipathLus(MP_OID oid, MP_OID_LIST **ppList);
```

Parameters

oid The object ID of the plugin or device product object.

ppList A pointer to a pointer to an object ID that has type `MP_OID_LIST` structure. With a successful return, this contains a pointer to an `MP_OID_LIST` that contains the object IDs of all the (multipath) logical units associated with the specified plugin object ID.

Description The `MP_GetMultipathLus()` function returns a list of multipath logical units associated to a plugin.

When the caller is finished using the list it must free the memory used by the list by calling `MP_FreeOidList`.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE` The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_INVALID_PARAMETER` The *ppList* is null or specifies a memory area that cannot be written, or when *oid* has a type subfield other than `MP_OBJECT_TYPE_DEVICE_PRODUCT` or `MP_OBJECT_TYPE_PLUG IN`.

`MP_STATUS_OBJECT_NOT_FOUND` The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_SUCCESS` The operation is successful.

`MP_STATUS_FAILED` The plugin for the specified object ID is not found.

`MP_STATUS_INSUFFICIENT_MEMORY` A memory allocation failure occurred.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), MP_GetMPLogicalUnitProperties(3MPAPI), attributes(5)

Multipath Management API Version 1.0

Name MP_GetObjectType – get an object type

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetObjectType(MP_OID oid, MP_OBJECT_TYPE *pObjectType);
```

Parameters

oid	The initialized object ID to have the type determined.
pObjectType	A pointer to an object ID that has type `MP_OBJECT_TYPE` variable allocated by the caller. With a successful return it contains the object type of *oid*.

Description The `MP_GetObjectType()` function gets the object type of an initialized object ID.

This API is provided so that clients can determine the type of object an object ID represents. This can be very useful for a client function that receives notifications.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to th e API.
`MP_STATUS_INVALID_PARAMETER`	The *pObjectType* is null or specifies a memory area to which data cannot be written.
`MP_STATUS_SUCCESS`	The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_RegisterForObjectVisibilityChanges(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetPathLogicalUnitProperties – get the specified path properties

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetPathLogicalUnitProperties(MP_OID oid,
    MP_PATH_LOGICAL_UNIT_PROPERTIES *pProps);
```

Parameters

oid The object ID of the logical unit path.

pProps A pointer to an object ID that has type `MP_PATH_LOGICAL_UNIT_PROPERTIES` structure allocated by the caller. With a successful return, this structure contains the properties of the path specified by *oid*.

Description The `MP_GetPathLogicalUnitProperties()` function gets the properties of the specified path.

Return Values

`MP_STATUS_INVALID_PARAMETER` The *pProps* is null or specifies a memory area to which data cannot be written, or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_PATH_LU`.

`MP_STATUS_INVALID_OBJECT_TYPE` The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_OBJECT_NOT_FOUND` The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_SUCCESS` The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetAssociatedPathOidList(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetPluginOidList – get a list of the object IDs

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetPluginOidList(MP_OID_LIST **ppList);
```

Parameters

ppList A pointer to a pointer to an object ID that has type `MP_OID_LIST`. With a successful return, this contains a pointer to an object ID that has type `MP_OID_LIST` that contains the object IDs of all of the plugins currently loaded by the library.

Description The `MP_GetPluginOidList()` function returns a list of the object IDs of all currently loaded plugins. The returned list is guaranteed to not contain any duplicate entries.

When the caller is finished using the list it must free the memory used by the list by calling `MP_FreeOidList(3MPAPI)`.

Return Values

`MP_STATUS_INVALID_PARAMETER`	The *ppList* is null or specifies a memory area to which data cannot be written.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FAILED`	The plugin for the specified object ID is not found.
`MP_STATUS_INSUFFICIENT_MEMORY`	A memory allocation failure occurred.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_FreeOidList(3MPAPI)`, `MP_GetPluginProperties(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetPluginProperties – get specified plugin properties

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetPluginProperties(MP_OID oid,
     MP_PLUGIN_PROPERTIES *pProps);
```

Parameters

oid The object ID of the plugin.

pProps A pointer to an object ID that has type `MP_PLUGIN_PR OPERTIES` structure allocated by the caller. With a successful return, this structure contains the properties of the plugin specified by *oid*.

Description The `MP_GetPluginProperties()` function gets the properties of the specified plugin.

Return Values

`MP_STATUS_INVALID_PARAMETER` The *pProps* is null or specifies a memory area to which data cannot be written, or the *oid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`.

`MP_STATUS_INVALID_OBJECT_TYPE` The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_OBJECT_NOT_FOUND` The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_SUCCESS` The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetProprietaryLoadBalanceProperties(3MPAPI)`, `MP_GetPluginOidList(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetProprietaryLoadBalanceOidList – get a list of object IDs

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetProprietaryLoadBalanceOidList(MP_OID oid
     MP_OID_LIST **ppList);
```

Parameters

oid The object ID of the plugin.

ppList A pointer to a pointer to an object ID that has type `MP_OID_LIST` structure. With a successful return, this contains a pointer to an object ID that has type `MP_OID_LIST` that contains the object IDs of all the proprietary load balance types associated with the specified plugin.

Description The `MP_GetProprietaryLoadBalanceOidList()` function returns a list of the object IDs of all the proprietary load balance algorithms associated with this plugin.

When the caller is finished using the list, it must free the memory used by the list by calling `MP_FreeOidList`.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *ppList* in null or specifies a memory area to which data cannot be written, or if the *oid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FAILED`	The plugin for the specified object ID is not found.
`MP_STATUS_INSUFFICIENT_MEMORY`	A memory allocation failure occurred.
`MP_STATUS_UNSUPPORTED`	The implementation does not support the API.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), MP_GetProprietaryLoadBalanceProperties(3MPAPI), attributes(5)

Multipath Management API Version 1.0

Name MP_GetProprietaryLoadBalanceProperties – get load balance properties

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetProprietaryLoadBalanceProperties(MP_OID oid,
     MP_PROPRIETARY_LOAD_BALANCE_PROPERTIES *pProps);
```

Parameters

oid The object ID of the proprietary load balance.

pProps A pointer to an object ID that has type `MP_PROPRIETARY_LOAD_BALANCE_PROPERTIES` structure allocated by the caller. With a successful return, this structure contains the properties of the proprietary load balance algorithm specified by the *oid*.

Description The `MP_GetProprietaryLoadBalanceProperties()` function returns the properties of the specified load balance.

Return Values

`MP_STATUS_INVALID_PARAMETER`
The *pObjectType* is null or specifies a memory area to which data cannot be written, or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_PROPRIETARY_LOAD_BALANCE`.

`MP_STATUS_INVALID_OBJECT_TYPE`
The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_OBJECT_NOT_FOUND`
The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_SUCCESS`
The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetProprietaryLoadBalanceOidList(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetTargetPortGroupProperties – return properties of the target port group

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetTargetPortGroupProperties(MP_OID oid,
     MP_TARGET_PORT_GROUP_PROPERTIES *pProps);
```

Parameters

oid The object ID of the target port group.

pProps A pointer to an object ID that has type `MP_TARGET_PORT_GROUP_PROPERTIES` structure allocated by the caller. With a successful return, this structure contains the properties of the target port group specified by the *oid*.

Description The `MP_GetTargetPortGroupProperties()` function returns the properties of the specified target port group.

Return Values

`MP_STATUS_INVALID_PARAMETER`
The *pProps* is null or specifies a memory area to which data cannot be written, or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_TARGET_PORT_GROUP`.

`MP_STATUS_INVALID_OBJECT_TYPE`
The *oid* does not specify a valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_OBJECT_NOT_FOUND`
The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_SUCCESS`
The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetAssociatedTPGOidList(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_GetTargetPortOidList – get a list of target port object IDs

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetTargetPortOidList(MP_OID oid,
     MP_OID_LIST **ppList);
```

Parameters

oid The object ID of the target port group.

ppList A pointer to a pointer to an object ID that has type `MP_OID_LIST` structure. With a successful return, this contains a pointer to an object ID that has type `MP_OID_LIST` that contains the object IDs of all the target ports associated with the specified target port group *oid*.

Description The `MP_GetTargetPortOidList()` function returns a list of the object IDs of the target ports in the specified target port group.

When the caller is finished using the list it must free the memory used by the list by calling `MP_FreeOidList`.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *ppList* is null or specifies a memory area to which data cannot be written, or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_TARGET_PORT`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FAILED`	The target port group for the specified object ID is not found.
`MP_STATUS_INSUFFICIENT_MEMORY`	A memory allocation failure occurred.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), MP_GetTargetPortProperties(3MPAPI), attributes(5)

Multipath Management API Version 1.0

Name MP_GetTargetPortProperties – get targrt port properties

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_GetTargetPortProperties(MP_OID oid,
     MP_TARGET_PORT_GROUP_PROPERTIES *pProps);
```

Parameters

oid The object ID of the target port group.

pProps A pointer to an object ID that has type `MP_TARGET_PORT_GROUP_PROPERTIES` structure allocated by the caller. With a successful return, this structure contains the properties of the target port group specified by the *oid*.

Description The `MP_GetTargetPortProperties()` function returns the properties of the specified target port.

Return Values

`MP_STATUS_INVALID_PARAMETER`	The *pProps* is null or specifies a memory area to which data cannot be written or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_TARGET_PORT`.
`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_GetTargetPortOidList(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_RegisterForObjectPropertyChanges – register a client function to be called

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_RegisterForObjectPropertyChanges(
     MP_OBJECT_PROPERTY_FN pClientFn, MP_OBJECT_TYPE objectType,
     void *pCallerData, MP_OID pluginOid);
```

Parameters

pClientFn	A pointer to an object ID that has type `MP_OBJECT_PROPERTY_FN` function defined by the client. With a successful return, this function is called to inform the client of objects that have had one or more properties changed.
objectType	The type of object the client wishes to register for property change callbacks.
pCallerData	A pointer that is passed to the callback routine with each event. This might be used by the caller to correlate the event to the source of the registration.
pluginOid	If this is a valid plugin object ID, then registration is limited to that plugin. If this is zero, then the registration is for all plugins.

Description The `MP_RegisterForObjectPropertyChanges()` function registers a client function to be called whenever the property of an object changes.

The function specified by *pClientFn* is called whenever the property of an object changes. For the purposes of this function, a property is defined to be a field in an object's property structure and the object's status. Therefore, the client function is not called if a statistic of the associated object changes. But, it is called when the status changes (e.g., from working to failed) or when a name or other field in a property structure changes.

It is not an error to re-register a client function. However, a client function has only one registration. The first call to deregister a client function will deregister it no matter how many calls to register the function have been made.

If multiple properties of an object change simultaneously, a client function can be called only once to be notified that all the changes have occurred.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *pluginOid* or *objectType* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_OBJECT_NOT_FOUND`	The *pluginOid* owner ID or object sequence number is invalid.
`MP_STATUS_INVALID_PARAMETER`	The *pCallerData* is null or if the *pluginOid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`, or when *objectType* is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.

`MP_STATUS_FN_REPLACED`	An existing client function is replaced with the one specified in *pClientFn*.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_DeregisterForObjectPropertyChanges(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_RegisterForObjectVisibilityChanges – register a client function to be called

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_RegisterForObjectVisibilityChanges(
     MP_OBJECT_PROPERTY_FN pClientFn, MP_OBJECT_TYPE objectType,
     void *pCallerData, MP_OID pluginOid);
```

Parameters

pClientFn	A pointer to an object ID that has type `MP_OBJECT_VISIBILITY_FN` function defined by the client. With a successful return, this function is called to inform the client of objects that have had one or more properties changed.
objectType	The type of object the client wishes to register for property change callbacks.
pCallerData	A pointer that is passed to the callback routine with each event. This might be used by the caller to correlate the event to the source of the registration.
pluginOid	If this is a valid plugin object ID, then registration is limited to that plugin. If this is zero, then the registration is for all plugins.

Description The `MP_RegisterForObjectVisibilityChanges()` function registers a client function to be called whenever the property of an object changes. The function specified by *pClientFn* is called whenever objects appear or disappear.

It is not an error to re-register a client function. However, a client function has only one registration. The first call to deregister a client function will deregister it no matter how many calls to register the function have been made.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *pluginOid* or *objectType* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_OBJECT_NOT_FOUND`	The *pluginOid* owner ID or object sequence number is invalid.
`MP_STATUS_INVALID_PARAMETER`	The *pCallerData* is null or if the *pluginOid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`, or when *objectType* is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FN_REPLACED`	An existing client function is replaced with the one specified in *pClientFn*.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `MP_DeregisterForObjectVisibilityChanges(3MPAPI)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_RegisterPlugin – register a plugin with the common library

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_RegisterPlugin(MP_WCHAR *pPluginId,
     MP_CHAR *pFileName);
```

Parameters

pPluginId A pointer to the key name shall be the reversed domain name of the vendor followed by a ".", followed by the vendor-specific name for the plugin that uniquely identifies it.

pFileName The full path name of the plugin library.

Description The `MP_RegisterPlugin()` function registers a plugin with the common library. The current implementation adds an entry to the `/etc/mpapi.conf` file.

Unlike some other APIs, this API is implemented entirely in the common library. It must be called before a plugin is invoked by the common library.

This API does not impact dynamically add or change plugins bound to a running library instance. Instead, it causes an application that is currently not using a plugin to access the specified plugin on future calls to the common library. This is generally the behavior expected from dynamically loaded modules.

This API is typically called by a plugin's installation software to inform the common library of the path for the plugin library.

It is not an error to re-register a plugin. However, a plugin has only one registration. The first call to deregister a plugin will deregister it no matter how many calls to register the plugin have been made.

A vendor may register multiple plugins by using separate plugin IDs and filenames.

Return Values

`MP_STATUS_INVALID_PARAMETER` The *pFileName* does not exist.

`MP_STATUS_SUCCESS` The operation is successful.

Files `/etc/mpapi.conf` MPAPI library configuration file

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), MP_DeregisterPlugin(3MPAPI), mpapi.conf(4), attributes(5)

Multipath Management API Version 1.0

Name MP_SetFailbackPollingRate – set the polling rates

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_SetFailbackPollingRate(MP_OID oid,
     MP_UINT32 pollingRate);
```

Parameters

oid An object ID of either the plugin or a multipath logical unit.

pollingRate The value to be set in `MP_PLUGIN_PROPERTIES` *currentFailbackPollingRate* or `MP_MULTIPATH_LOGICAL_UNIT_PROPERTIES` *failbackPollingRate*.

Description The `MP_SetFailbackPollingRate()` function sets the polling rates. Setting the *pollingRate* to zero disables polling.

If the object ID refers to a plugin, this sets the *currentFailbackPollingRate* property in the plugin properties. If the object ID refers to a multipath logical unit, this sets the *failbackPollingRate* property.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE` The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_INVALID_PARAMETER` One of the polling values is outside the range supported by the driver, or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN` or `MP_OBJECT_TYPE_MULTIPATH_LU`.

`MP_STATUS_OBJECT_NOT_FOUND` The *oid* ownerID or object sequence number is invalid.

`MP_STATUS_SUCCESS` The operation is successful.

`MP_STATUS_UNSUPPORTED` The implementation does not support the API.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_SetLogicalUnitLoadBalanceType – set a load balancing policy

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_SetLogicalUnitLoadBalanceType(MP_OID logicalUnitoid,
     MP_LOAD_BALANCE_TYPE  loadBalance);
```

Parameters

logicalUnitOid The object ID of the multipath logical unit.

loadBalance The desired load balance policy for the specified logical unit.

Description The `MP_SetLogicalUnitLoadBalanceType()` function sets the multipath logical unit's load balancing policy. The value must correspond to one of the supported values in `MP_PLUGIN_PROPERTIES.SupportedLogicalUnitLoadBalanceTypes`.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *logicalUnitOid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *loadBalance* is invalid or *logicalUnitOid* has a type subfield other than `MP_OBJECT_TYPE_MULTIPATH_LU`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *logicalUnitOid* owner ID or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FAILED`	The specified *loadBalance* type cannot be handled by the plugin. One possible reason for this is a request to set `MP_LOAD_BALANCE_TYPE_PRODUCT` when the specified logical unit has no corresponding `MP_DEVICE_PRODUCT_PROPERTIES` instance (i.e., the plugin does not have a product-specific load balance algorithm for the LU product).
`MP_STATUS_UNSUPPORTED`	The implementation does not support the API.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also libMPAPI(3LIB), attributes(5)

Multipath Management API Version 1.0

Name MP_SetOverridePath – manually override a logical unit path

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_SetOverridePath(MP_OID logicalUnitOid,
     MP_OID pathOid);
```

Parameters

logicalUnitOid The object ID of the multipath logical unit.

pathOid The object ID of the path logical unit.

Description The `MP_SetOverridePath()` function is used to manually override the path for a logical unit. The path is exclusively used to access the logical unit until cleared. Use `MP_CancelOverride` to clear the override.

This API allows the administrator to disable the dri ver's load balance algorithm and force all I/O operations to a specific path. The existing path weight configuration is maintained. If the administrator undoes the override (by calling `MP_CancelOverridePath`), the driver starts load balancing based on the weights of available paths (and target port group access state for asymmetric devices).

If the multipath logical unit is part of a target with asymmetrical access, executing this command could cause failover.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *logicalUnitOid* or *pathOid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *logicalUnitOid* has a type subfield other than `MP_OBJECT_TYPE_MULTIPATH_LU`, or if *pathOid* has an object type other than `MP_OBJECT_TYPE_PATH_LU`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *logicalUnitOid*, *pathOid* owner ID, or object sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_UNSUPPORTED`	The implementation does not support the API.
`MP_STATUS_PATH_NONOPERATIONAL`	The driver cannot communicate through selected path

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_SetPathWeight – set the weight of a path

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_SetPathWeight(MP_OID pathOid, MP_UINT32 weight);
```

Parameters

pathOid The object ID of the path logical unit.

weight A weight that will be assigned to the path logical unit.

Description The `MP_SetPathWeight()` function sets the weight to be assigned to a particular path.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *pathOid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_OBJECT_NOT_FOUND`	The *pathOid* ownerID or object sequence number is invalid.
`MP_STATUS_INVALID_PARAMETER`	The *pathOid* has a type subfield other than `MP_OBJECT_TYPE_PATH_LU`, or when the weight parameter is greater than the plugin's maximum weight property.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FAILED`	The operation failed.
`MP_STATUS_UNSUPPORTED`	The driver does not support setting path weight.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_SetPluginLoadBalanceType – set the plugin default load balance policy

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_SetPluginLoadBalanceType(MP_OID oid,
     MP_LOAD_BALANCE_TYPE loadBalance);
```

Parameters

oid The object ID of the plugin.

loadBalance The desired default load balance policy for the specified plugin.

Description The `MP_SetPluginLoadBalanceType()` function sets the default load balance policy for the plugin. The value must correspond to one of the supported values in `MP_PLUGIN_PROPERTIES.SupportedPluginLoadBalanceTypes`.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`	The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.
`MP_STATUS_INVALID_PARAMETER`	The *loadBalance* is invalid or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`.
`MP_STATUS_OBJECT_NOT_FOUND`	The *oid* ownerID or sequence number is invalid.
`MP_STATUS_SUCCESS`	The operation is successful.
`MP_STATUS_FAILED`	The specified *loadBalance* type cannot be handled by the plugin.
`MP_STATUS_UNSUPPORTED`	The implementation does not support the API.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_SetProbingPollingRate – set the polling rate

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_SetProbingPollingRate(MP_OID oid,
     MP_UINT32 pollingRate);
```

Parameters

oid An object ID of either the plugin or a multipath logical unit.

pollingRate The value to be set in `MP_PLUGIN_PROPERTIES` *currentProbingPollingRate* or `MP_MULTIPATH_LOGICAL_UNIT_PROPERTIES` *ProbingPollingRate*.

Description The `MP_SetProbingPollingRate()` function sets the polling rates. Setting the *pollingRate* to zero disables polling.

If the object ID refers to a plugin, this sets the *currentProbingPollingRate* property in the plugin properties. If the object ID refers to a multipath logical unit, this sets the *ProbingPollingRate* property.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE` The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_INVALID_PARAMETER` One of the polling values is outside the range supported by the driver or when the *oid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN` or `MP_OBJECT_TYPE_MULTIPATH_LU`.

`MP_STATUS_OBJECT_NOT_FOUND` The *oid* ownerID or sequence number is invalid.

`MP_STATUS_SUCCESS` The operation is successful.

`MP_STATUS_UNSUPPORTED` The implementation does not support the API.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_SetProprietaryProperties – set proprietary properties

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_SetProprietaryProperties(MP_OID oid,
     MP_UINT32 count, MP_PROPRIETARY_PROPERTY *pPropertyList);
```

Parameters

oid The object ID representing an object ID that has type `MP_LOAD_BALANCE_PROPIETARY_TYPE`, or `MP_PLUGIN_PROPERTIES`, or `MP_MULTIPATH_LOGICAL_UNIT_PROPERTIES` instance.

count The number of valid items in *pPropertyList*.

pPropertyList A pointer to an array of property name/value pairs. This array must contain the same number of elements as does *count*.

Description The `MP_SetProprietaryProperties()` function sets proprietary properties in supported object instances.

This API allows an application with a priori knowledge of proprietary plugin capabilities to set proprietary properties. The *pPropertyList* is a list of property name/value pairs. The property names shall be a subset of the proprietary property names listed in the referenced object ID.

Return Values

`MP_STATUS_INVALID_OBJECT_TYPE`
The *oid* does not specify a valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_OBJECT_NOT_FOUND`
The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_INVALID_PARAMETER`
The *pPropertyList* is null, or when one of the properties referenced in the list is not associated with the specified object ID, or the *oid* has a type subfield other than`MP_OBJECT_TYPE_PROPRIETARY_LOAD_BALANCE`, or `MP_OBJECT_TYPE_PLUGIN`, or `MP_OBJECT_TYPE_MULTIPATH_LU`.

`MP_STATUS_SUCCESS`
The operation is successful.

`MP_STATUS_UNSUPPORTED`
The API is not supported.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name MP_SetTPGAccess – set a target port group access state

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>

MP_STATUS MP_SetTPGAccess(MP_OID luOid, MP_UINT32 count,
     MP_TPG_STATE_PAIR *pTpgStateList);
```

Parameters

luOid	An object ID that has type `MP_MULTIPATH_LOGICAL_UNIT`.
count	The number of valid items in the *pTpgStateList*.
pTpgStateList	A pointer to an array of data structure `MP_TPG_STATE_PAIR`. This array must contain the same number of elements as *count*.

Description The `MP_SetTPGAccess()` function sets the access state for a list of target port groups. This allows a client to force a failover or failback to a desired set of target port groups. This is only valid for devices that support explicit access state manipulation (i.e., the field *explicitFailover* of data structure `MP_TARGET_PORT_GROUP_PROPERTIES` must be true).

This API provides the information needed to set up a `SCSI SET TARGET PORT GROUPS` command.

The plugin should not implement this API by directly calling the `SCSI SET TARGET PORT GROUPS` command. The plugin should use the MP drivers API (for example, `ioctl`) if available.

There are two reasons why this API is restricted to devices supporting explicit failover commands. Without an explicit command, the behavior of failback tends to be device-specific.

When the caller is finished using the list it must free the memory used by the list by calling `MP_FreeOidList`.

Return Values

`MP_STATUS_ACCESS_STATE_INVALID`
: The target device returns a status indicating the caller is attempting to establish an illegal combination of access states.

`MP_STATUS_FAILED`
: The underlying interface failed the command for some reason other than `MP_STATUS_ACCESS_STATE_INVALID`.

`MP_STATUS_INVALID_OBJECT_TYPE`
: The *luOid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_OBJECT_NOT_FOUND`
: The *luOid* owner ID or object sequence number is invalid.

`MP_STATUS_INVALID_PARAMETER`
The *pTpgStateList* is null, or when one of the TPGs referenced in the list is not associated with the specified MP logical unit, or the *luOid* has a type subfield other than `MP_OBJECT_TYPE_MULTIPATH_LU`.

`MP_STATUS_SUCCESS`
The operation is successful.

`MP_STATUS_UNSUPPORTED`
The API is not supported.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe
Standard	ANSI INCITS 412 Multipath Management API

See Also `libMPAPI(3LIB)`, `attributes(5)`

Multipath Management API Version 1.0

Name m_setvalues_layout – set layout values of a `LayoutObject`

Synopsis

```
cc [ flag... ] file... –llayout [ library... ]
#include <sys/layout.h>

int m_setvalues_layout(LayoutObject layout_object,
     const LayoutValues values, int *index_returned);
```

Description The `m_setvalues_layout()` function changes the layout values of a LayoutObject.

The *layout_object* argument specifies a `LayoutObject` returned by the `m_create_layout(3LAYOUT)` function.

The *values* argument specifies the list of layout values that are to be changed. The values are written into the `LayoutObject` and may affect the behavior of subsequent layout functions. Some layout values do alter internal states maintained by a `LayoutObject`.

The `m_setvalues_layout()` function can be implemented as a macro that evaluates the first argument twice.

Return Values Upon successful completion, the requested layout values are set and `0` is returned. Otherwise `–1` is returned and `errno` is set to indicate the error. If any value cannot be set, none of the layout values are changed and the (zero-based) index of the first value causing the error is returned in *index_returned*.

Errors The `m_setvalues_layout()` function may fail if:

`EINVAL` The layout value specified by *index_returned* is unknown, its value is invalid, or the *layout_object* argument is invalid.

`EMFILE` There are `{OPEN_MAX}` file descriptors currently open in the calling process.

Usage Do not use expressions with side effects such as auto-increment or auto-decrement within the first argument to the `m_setvalues_layout()` function.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `m_create_layout(3LAYOUT)`, `attributes(5)`, `standards(5)`

Name m_transform_layout – layout transformation

Synopsis

```
cc [ flag... ] file... –llayout [ library... ]
#include <sys/layout.h>

int m_transform_layout(LayoutObject layout_object,
     const char *InpBuf, const size_t ImpSize, const void *OutBuf,
     size_t *Outsize, size_t *InpToOut, size_t *OutToInp,
     unsigned char *Property, size_t *InpBufIndex);
```

Description The `m_transform_layout()` function performs layout transformations (reordering, shaping, cell determination) or provides additional information needed for layout transformation (such as the expected size of the transformed layout, the nesting level of different segments in the text and cross-references between the locations of the corresponding elements before and after the layout transformation). Both the input text and output text are character strings.

The `m_transform_layout()` function transforms the input text in *InpBuf* according to the current layout values in *layout_object*. Any layout value whose value type is `LayoutTextDescriptor` describes the attributes of the *InpBuf* and *OutBuf* arguments. If the attributes are the same for both *InpBuf* and *OutBuf*, a null transformation is performed with respect to that specific layout value.

The *InpBuf* argument specifies the source text to be processed. The *InpBuf* may not be `NULL`, unless there is a need to reset the internal state.

The *InpSize* argument is the number of bytes within *InpBuf* to be processed by the transformation. Its value will not change after return from the transformation. *InpSize* set to −1 indicates that the text in *InpBuf* is delimited by a null code element. If *InpSize* is not set to −1, it is possible to have some null elements in the input buffer. This might be used, for example, for a "one shot" transformation of several strings, separated by nulls.

Output of this function may be one or more of the following depending on the setting of the arguments:

OutBuf Any transformed data is stored in *OutBuf*, converted to `ShapeCharset`.

Outsize The number of bytes in *OutBuf*.

InpToOut A cross-reference from each *InpBuf* code element to the transformed data. The cross-reference relates to the data in *InpBuf* starting with the first element that *InpBufIndex* points to (and not necessarily starting from the beginning of the *InpBuf*).

OutToInp A cross-reference to each *InpBuf* code element from the transformed data. The cross-reference relates to the data in *InpBuf* starting with the first element that *InpBufIndex* points to (and not necessarily starting from the beginning of the *InpBuf*).

Property A weighted value that represents peculiar input string transformation properties with different connotations as explained below. If this argument is not a null pointer, it represents an array of values with the same number of elements as the source substring text before the transformation. Each byte will contain relevant "property" information of the corresponding element in *InpBuf* starting from the element pointed by *InpBufIndex*. The four rightmost bits of each "property" byte will contain information for bidirectional environments (when `ActiveDirectional` is True) and they will mean "`NestingLevels`." The possible value from 0 to 15 represents the nesting level of the corresponding element in the *InpBuf* starting from the element pointed by *InpBufIndex*. If `ActiveDirectional` is false the content of `NestingLevel` bits will be ignored. The leftmost bit of each "property" byte will contain a "new cell indicator" for composed character environments, and will have a value of either 1 (for an element in *InpBuf* that is transformed to the beginning of a new cell) or 0 (for the "zero-length" composing character elements, when these are grouped into the same presentation cell with a non-composing character). Here again, each element of "property" pertains to the elements in the *InpBuf* starting from the element pointed by *InpBufIndex*. (Remember that this is not necessarily the beginning of *InpBuf*). If none of the transformation properties is required, the argument *Property* can be `NULL`. The use of "property" can be enhanced in the future to pertain to other possible usage in other environments.

The *InpBufIndex* argument is an offset value to the location of the transformed text. When `m_transform_layout()` is called, *InpBufIndex* contains the offset to the element in *InpBuf* that will be transformed first. (Note that this is not necessarily the first element in *InpBuf*). At the return from the transformation, *InpBufIndex* contains the offset to the first element in the *InpBuf* that has not been transformed. If the entire substring has been transformed successfully, *InpBufIndex* will be incremented by the amount defined by *InpSize*.

Each of these output arguments may be `NULL` to specify that no output is desired for the specific argument, but at least one of them should be set to a non-null value to perform any significant work.

The layout object maintains a directional state that keeps track of directional changes, based on the last segment transformed. The directional state is maintained across calls to the layout transformation functions and allows stream data to be processed with the layout functions. The directional state is reset to its initial state whenever any of the layout values `TypeOfText`, `Orientation`, or `ImplicitAlg` is modified by means of a call to `m_setvalues_layout()`.

The *layout_object* argument specifies a `LayoutObject` returned by the `m_create_layout()` function.

The *OutBuf*argument contains the transformed data. This argument can be specified as a null pointer to indicate that no transformed data is required.

The encoding of the *OutBuf* argument depends on the `ShapeCharset` layout value defined in *layout_object*. If the `ActiveShapeEditing` layout value is not set (False), the encoding of *OutBuf* is guaranteed to be the same as the codeset of the locale associated with the `LayoutObject` defined by *layout_object*.

On input, the *OutSize* argument specifies the size of the output buffer in number of bytes. The output buffer should be large enough to contain the transformed result; otherwise, only a partial transformation is performed. If the `ActiveShapeEditing` layout value is set (True) the *OutBuf* should be allocated to contain at least the *InpSize* multiplied by `ShapeCharsetSize`.

On return, the *OutSize* argument is modified to the actual number of bytes placed in *OutBuf*.

When the *OutSize* argument is specified as zero, the function calculates the size of an output buffer large enough to contain the transformed text, and the result is returned in this field. The content of the buffers specified by *InpBuf* and *OutBuf*, and the value of *InpBufIndex*, remain unchanged. If *OutSize* = NULL, the `EINVAL` error condition should be returned.

If the *InpToOut* argument is not a null pointer, it points to an array of values with the same number of bytes in *InpBuf* starting with the one pointed by *InpBufIndex* and up to the end of the substring in the buffer. On output, the nth value in *InpToOut* corresponds to the nth byte in *InpBuf*. This value is the index (in units of bytes) in *OutBuf*that identifies the transformed `ShapeCharset` element of the nth byte in *InpBuf*. In the case of multibyte encoding, the index points (for each of the bytes of a code element in the *InpBuf*) to the first byte of the transformed code element in the *OutBuf*.

InpToOut may be specified as `NULL` if no index array from *InpBuf* to *OutBuf* is desired.

If the *OutToInp* argument is not a null pointer, it points to an array of values with the same number of bytes as contained in *OutBuf*. On output, the nth value in *OutToInp* corresponds to the nth byte in *OutBuf* This value is the index in *InpBuf*, starting with the byte pointed to by *InpBufIndex*, that identifies the logical code element of the nth byte in *OutBuf*. In the case of multibyte encoding, the index will point for each of the bytes of a transformed code element in the *OutBuf* to the first byte of the code element in the *InpBuf*.

OutToInp may be specified as `NULL` if no index array from *OutBuf* to *InpBuf* is desired.

To perform shaping of a text string without reordering of code elements, the *layout_object* should be set with input and output layout value *TypeOfText* set to `TEXT_VISUAL` and both in and out of `Orientation` set to the same value.

Return Values If successful, the `m_transform_layout()` function returns `0`. If unsuccessful, the returned value is `−1` and the `errno` is set to indicate the source of error. When the size of *OutBuf* is not large enough to contain the entire transformed text, the input text state at the end of the uncompleted transformation is saved internally and the error condition `E2BIG` is returned in `errno`.

Errors The `m_transform_layout()` function may fail if:

`E2BIG` The output buffer is full and the source text is not entirely processed.

`EBADF` The layout values are set to a meaningless combination or the layout object is not valid.

`EILSEQ` Transformation stopped due to an input code element that cannot be shaped or is invalid. The *InpBufIndex* argument is set to indicate the code element causing the error. The suspect code element is either a valid code element but cannot be shaped into the `ShapeCharset` layout value, or is an invalid code element not defined by the codeset of the locale of *layout_object*. The `mbtowc()` and `wctomb()` functions, when used in the same locale as the `LayoutObject`, can be used to determine if the code element is valid.

`EINVAL` Transformation stopped due to an incomplete composite sequence at the end of the input buffer, or *OutSize* contains `NULL`.

`ERANGE` More than 15 embedding levels are in source text or *InpBuf* contain unbalanced directional layout information (push/pop) or an incomplete composite sequence has been detected in the input buffer at the beginning of the string pointed to by *InpBufIndex*.

An incomplete composite sequence at the end of the input buffer is not always detectable. Sometimes, the fact that the sequence is incomplete will only be detected when additional character elements belonging to the composite sequence are found at the beginning of the next input buffer.

Usage A `LayoutObject` will have a meaningful combination of default layout values. Whoever chooses to change the default layout values is responsible for making sure that the combination of layout values is meaningful. Otherwise, the result of `m_transform_layout()` might be unpredictable or implementation-specific with `errno` set to `EBADF`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `attributes(5)`, `standards(5)`

Name m_wtransform_layout – layout transformation for wide character strings

Synopsis

```
cc [ flag... ] file... –llayout [ library... ]
#include <sys/layout.h>

int m_wtransform_layout(LayoutObject layout_object,
     const wchar_t *InpBuf, const size_t ImpSize, const void *OutBuf,
     size_t *Outsize, size_t *InpToOut, size_t *OutToInp,
     unsignedchar *Property, size_t *InpBufIndex);
```

Description The `m_wtransform_layout()` function performs layout transformations (reordering, shaping, cell determination) or provides additional information needed for layout transformation (such as the expected size of the transformed layout, the nesting level of different segments in the text and cross-references between the locations of the corresponding elements before and after the layout transformation). Both the input text and output text are wide character strings.

The `m_wtransform_layout()` function transforms the input text in *InpBuf* according to the current layout values in *layout_object*. Any layout value whose value type is `LayoutTextDescriptor` describes the attributes of the *InpBuf* and *OutBuf* arguments. If the attributes are the same for both *InpBuf* and *OutBuf*, a null transformation is performed with respect to that specific layout value.

The *InpBuf* argument specifies the source text to be processed. The *InpBuf* may not be `NULL`, unless there is a need to reset the internal state.

The *InpSize* argument is the number of bytes within *InpBuf* to be processed by the transformation. Its value will not change after return from the transformation. *InpSize* set to –1 indicates that the text in *InpBuf* is delimited by a null code element. If *InpSize* is not set to –1, it is possible to have some null elements in the input buffer. This might be used, for example, for a "one shot" transformation of several strings, separated by nulls.

Output of this function may be one or more of the following depending on the setting of the arguments:

OutBuf Any transformed data is stored in *OutBuf*, converted to `ShapeCharset`.

Outsize The number of wide characters in *OutBuf*.

InpToOut A cross-reference from each *InpBuf* code element to the transformed data. The cross-reference relates to the data in *InpBuf* starting with the first element that *InpBufIndex* points to (and not necessarily starting from the beginning of the *InpBuf*).

OutToInp A cross-reference to each *InpBuf* code element from the transformed data. The cross-reference relates to the data in *InpBuf* starting with the first element that *InpBufIndex* points to (and not necessarily starting from the beginning of the *InpBuf*).

Property A weighted value that represents peculiar input string transformation properties with different connotations as explained below. If this argument is not a nullpointer, it represents an array of values with the same number of elements as the source substring text before the transformation. Each byte will contain relevant "property" information of the corresponding element in *InpBuf* starting from the element pointed by *InpBufIndex*. The four rightmost bits of each "property" byte will contain information for bidirectional environments (when `ActiveDirectional` is True) and they will mean "`NestingLevels`." The possible value from 0 to 15 represents the nesting level of the corresponding element in the *InpBuf* starting from the element pointed by *InpBufIndex*. If `ActiveDirectional` is false the content of `NestingLevel` bits will be ignored. The leftmost bit of each "property" byte will contain a "new cell indicator" for composed character environments, and will have a value of either 1 (for an element in *InpBuf* that is transformed to the beginning of a new cell) or 0 (for the "zero-length" composing character elements, when these are grouped into the same presentation cell with a non- composing character). Here again, each element of "property" pertains to the elements in the *InpBuf* starting from the element pointed by *InpBufIndex*. (Remember that this is not necessarily the beginning of *InpBuf*). If none of the transformation properties is required, the argument *Property* can be `NULL`. The use of "property" can be enhanced in the future to pertain to other possible usage in other environments.

The *InpBufIndex* argument is an offset value to the location of the transformed text. When `m_wtransform_layout()` is called, *InpBufIndex* contains the offset to the element in *InpBuf* that will be transformed first. (Note that this is not necessarily the first element in *InpBuf*). At the return from the transformation, *InpBufIndex* contains the offset to the first element in the *InpBuf* that has not been transformed. If the entire substring has been transformed successfully, *InpBufIndex* will be incremented by the amount defined by *InpSize*.

Each of these output arguments may be null to specify that no output is desired for the specific argument, but at least one of them should be set to a non-null value to perform any significant work.

In addition to the possible outputs above, *layout_object* maintains a directional state across calls to the transform functions. The directional state is reset to its initial state whenever any of the layout values `TypeOfText`, `Orientation`, or `ImplicitAlg` is modified by means of a call to `m_setvalues_layout()`.

The *layout_object* argument specifies a `LayoutObject` returned by the `m_create_layout()` function.

The *OutBuf*argument contains the transformed data. This argument can be specified as a null pointer to indicate that no transformed data is required.

The encoding of the *OutBuf* argument depends on the `ShapeCharset` layout value defined in *layout_object*. If the `ActiveShapeEditing` layout value is not set (False), the encoding of *OutBuf* is guaranteed to be the same as the codeset of the locale associated with the `LayoutObject` defined by *layout_object*.

On input, the *OutSize* argument specifies the size of the output buffer in number of wide characters. The output buffer should be large enough to contain the transformed result; otherwise, only a partial transformation is performed. If the `ActiveShapeEditing` layout value is set (True) the *OutBuf* should be allocated to contain at least the *InpSize* multiplied by `ShapeCharsetSize`.

On return, the *OutSize* argument is modified to the actual number of code elements in *OutBuf*.

When the *OutSize* argument is specified as zero, the function calculates the size of an output buffer large enough to contain the transformed text, and the result is returned in this field. The content of the buffers specified by *InpBuf* and *OutBuf*, and the value of *InpBufIndex*, remain unchanged. If *OutSize* = `NULL`, the `EINVAL` error condition should be returned.

If the *InpToOut* argument is not a null pointer, it points to an array of values with the same number of wide characters in *InpBuf* starting with the one pointed by *InpBufIndex* and up to the end of the substring in the buffer. On output, the nth value in *InpToOut* corresponds to the nth byte in *InpBuf*. This value is the index (in units of wide characters) in *OutBuf*that identifies the transformed `ShapeCharset` element of the nth byte in *InpBuf*.

InpToOut may be specified as `NULL` if no index array from *InpBuf* to *OutBuf* is desired.

If the *OutToInp* argument is not a null pointer, it points to an array of values with the same number of wide characters as contained in *OutBuf*. On output, the nth value in *OutToInp* corresponds to the nth byte in *OutBuf*. This value is the index in *InpBuf*, starting with wide character byte pointed to by *InpBufIndex*, that identifies the logical code element of the nth wide character in *OutBuf*.

OutToInp may be specified as `NULL` if no index array from *OutBuf* to *InpBuf* is desired.

To perform shaping of a text string without reordering of code elements, the *layout_object* should be set with input and output layout value *TypeOfText* set to `TEXT_VISUAL` and both in and out of `Orientation` set to the same value.

Return Values If successful, the `m_wtransform_layout()` function returns `0`. If unsuccessful, the returned value is `−1` and the `errno` is set to indicate the source of error. When the size of *OutBuf* is not

large enough to contain the entire transformed text, the input text state at the end of the uncompleted transformation is saved internally and the error condition `E2BIG` is returned in `errno`.

Errors The `m_wtransform_layout()` function may fail if:

`E2BIG` The output buffer is full and the source text is not entirely processed.

`EBADF` The layout values are set to a meaningless combination or the layout object is not valid.

`EILSEQ` Transformation stopped due to an input code element that cannot be shaped or is invalid. The *InpBufIndex* argument is set to indicate the code element causing the error. The suspect code element is either a valid code element but cannot be shaped into the `ShapeCharset` layout value, or is an invalid code element not defined by the codeset of the locale of *layout_object*. The `mbtowc()` and `wctomb()` functions, when used in the same locale as the `LayoutObject`, can be used to determine if the code element is valid.

`EINVAL` Transformation stopped due to an incomplete composite sequence at the end of the input buffer, or *OutSize* contains `NULL`.

`ERANGE` More than 15 embedding levels are in source text or *InpBuf* contain unbalanced directional layout information (push/pop) or an incomplete composite sequence has been detected in the input buffer at the beginning of the string pointed to by *InpBufIndex*.

An incomplete composite sequence at the end of the input buffer is not always detectable. Sometimes the fact that the sequence is incomplete will only be detected when additional character elements belonging to the composite sequence are found at the beginning of the next input buffer.

Usage A `LayoutObject` will have a meaningful combination of default layout values. Whoever chooses to change the default layout values is responsible for making sure that the combination of layout values is meaningful. Otherwise, the result of `m_wtransform_layout()` might be unpredictable or implementation-specific with `errno` set to `EBADF`.

Examples **EXAMPLE 1** Shaping and reordering input string into output buffer

The following example illustrated what the different arguments of `m_wtransform_layout()` look like when a string in *InpBuf* is shaped and reordered into *OutBuf*. Upper-case letters in the example represent left-to-right letters while lower-case letters represent right-to-left letters. `xyz` represents the shapes of cde.

```
Position:                   0123456789
InpBuf:                     AB cde 12z

Position:                   0123456789
```

EXAMPLE 1 Shaping and reordering input string into output buffer *(Continued)*

```
OutBuf:                     AB 12 zyxZ

Position:                   0123456789
OutToInp:                   0127865439

Position:                   0123456789
Property.NestLevel:         0001111220
Property.CelBdry:           1111111111
```

The values (encoded in bianry) returned in the *Property* argument define the directionality of each code element in the source text as defined by the type of algorithm used within the *layout_object*. Whlie the algorithm may be implementation dependent, the resulting values and levels are defined such as to allow a single method to be used in determining the directionality of the sourece text. The base rules are:

- Odd levels are always RTL.
- Even levels are always LTR.
- The `Orientation` layout value setting determines the initial level (0 or 1) used.

Within a *Property* array each increment in the level indicates the corresponding code elements should be presented in the opposite direction. Callers of this function should realize that the *Property* values for certain code elements is dependent on the context of the given character and the layout values: `Orientation` and `ImplicitAlg`. Callers should not assume that a given code element always has the same *Property* value in all cases.

EXAMPLE 2 Algorithm to handle nesting

The following is an example of a standard presentation algorithm that handles nesting correctly. The goal of this algorithm is ultimately to return to a zero nest level. Note that more efficient algorithms do exist; the following is provided for clarity rather than for efficiency.

1. Search for the highest next level in the string.
2. Reverse all surrounding code elements of the same level. Reduce the nest level of these code elements by 1.
3. Repeat 1 and 2 until all code elements are of level 0.

The following shows the progression of the example from above:

```
Position:           0123456789    0123456789    0123456789
InpBuf:             AB cde 12Z    AB cde 21Z    AB 12 edcZ
Property.NestLevel: 0001111220    0001111110    0000000000
Property.CellBdry:  1111111111    1111111111    1111111111
```

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also attributes(5), standards(5)

Name nan, nanf, nanl – return quiet NaN

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double nan(const char *tagp);

float nanf(const char *tagp);

long double nanl(const char *tagp);
```

Description The function call nan("*n*-char-sequence") is equivalent to:

```
strtod("NAN(n-char-sequence)", (char **) NULL);
```

The function call nan("") is equivalent to:

```
strtod("NAN()", (char **) NULL)
```

If *tagp* does not point to an *n*-char sequence or an empty string, the function call is equivalent to:

```
strtod("NAN", (char **) NULL)
```

Function calls to `nanf()` and `nanl()` are equivalent to the corresponding function calls to `strtof()` and `strtold()`. See `strtod(3C)`.

Return Values These functions return a quiet NaN.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `math.h(3HEAD)`, `strtod(3C)`, `attributes(5)`, `standards(5)`

Name nearbyint, nearbyintf, nearbyintl – floating-point rounding functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double nearbyint(double x);

float nearbyintf(float x);

long double nearbyintl(long double x);
```

Description These functions round their argument to an integer value in floating-point format, using the current rounding direction and without raising the inexact floating-point exception.

Return Values Upon successful completion, these functions return the rounded integer value.

If *x* is NaN, a NaN is returned.

If *x* is ±0, ±0 is returned.

If *x* is ±Inf, *x* is returned.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), math.h(3HEAD), attributes(5), standards(5)

Name nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl – next representable double-precision floating-point number

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double nextafter(double x, double y);

float nextafterf(float x, float y);

long double nextafterl(long double x, long double y);

double nexttoward(double x, long double y);

float nexttowardf(float x, long double y);

long double nexttowardl(long double x, long double y);
```

Description The `nextafter()`, `nextafterf()`, and `nextafterl()` functions compute the next representable floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, `nextafter()` returns the largest representable floating-point number less than *x*. The `nextafter()`, `nextafterf()`, and `nextafterl()` functions return *y* if *x* equals *y*.

The `nexttoward()`, `nexttowardf()`, and `nexttowardl()` functions are equivalent to the corresponding `nextafter()` functions, except that the second parameter has type `long double` and the functions return *y* converted to the type of the function if *x* equals *y*.

Return Values Upon successful completion, these functions return the next representable floating-point value following *x* in the direction of *y*.

If *x* == *y*, *y* (of the type *x*) is returned.

If *x* is finite and the correct function value would overflow, a range error occurs and ±`HUGE_VAL`, ±`HUGE_VALF`, and ±`HUGE_VALL` (with the same sign as *x*) is returned as appropriate for the return type of the function.

If *x* or *y* is NaN, a NaN is returned.

If *x* != *y* and the correct function value is subnormal, zero, or underflows, a range error occurs and either the correct function value (if representable) or 0.0 is returned.

Errors These functions will fail if:

Range Error — The correct value overflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the overflow floating-point exception is raised.

The `nextafter()` function sets `errno` to `ERANGE` if the correct value would overflow.

Range Error — The correct value underflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the underflow floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `nextafter()`. On return, if `errno` is non-zero, an error has occurred. The `nextafterf()`, `nextafterl()`. `nexttoward()`, `nexttowardf()`, and `nexttowardl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `feclearexcept(3M)`, `fetestexcept(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name p2open, p2close – open, close pipes to and from a command

Synopsis

```
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

int p2open(const char *cmd, FILE *fp[2]);

int p2close(FILE *fp[2]);
```

Description The `p2open()`gfunction forks and execs a shell running the command line pointed to by *cmd*. On return, `fp[0]` points to a `FILE` pointer to write the command's standard input and `fp[1]` points to a `FILE` pointer to read from the command's standard output. In this way the program has control over the input and output of the command.

The function returns `0` if successful; otherwise, it returns `−1`.

The `p2close()` function is used to close the file pointers that `p2open()` opened. It waits for the process to terminate and returns the process status. It returns `0` if successful; otherwise, it returns `−1`.

Return Values A common problem is having too few file descriptors. The `p2close()` function returns `−1` if the two file pointers are not from the same `p2open()`.

Examples **EXAMPLE 1** Example of file descriptors.

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

main(argc,argv)
int argc;
char **argv;
{
    FILE *fp[2];
    pid_t pid;
    char buf[16];

        pid=p2open("/usr/bin/cat", fp);
        if ( pid == −1 ) {
        fprintf(stderr, "p2open failed\n");
        exit(1);
    }
    write(fileno(fp[0]),"This is a test\n", 16);
    if(read(fileno(fp[1]), buf, 16) <=0)
        fprintf(stderr, "p2open failed\n");
    else
        write(1, buf, 16);
    (void)p2close(fp);
}
```

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Unsafe

See Also fclose(3C), popen(3C), setbuf(3C), attributes(5)

Notes Buffered writes on `fp[0]` can make it appear that the command is not listening. Judiciously placed `fflush()` calls or unbuffering `fp[0]` can be a big help; see fclose(3C).

Many commands use buffered output when connected to a pipe. That, too, can make it appear as if things are not working.

Usage is not the same as for `popen()`, although it is closely related.

Name pathfind – search for named file in named directories

Synopsis

```
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

char *pathfind(const char *path, const char *name, const char *mode);
```

Description The `pathfind()` function searches the directories named in *path* for the file *name*. The directories named in *path* are separated by colons (:). The *mode* argument is a string of option letters chosen from the set `[rwxfbcdpugks]`:

Letter	Meaning
`r`	readable
`w`	writable
`x`	executable
`f`	normal file
`b`	block special
`c`	character special
`d`	directory
`p`	FIFO (pipe)
`u`	set user ID bit
`g`	set group ID bit
`k`	sticky bit
`s`	size non-zero

Options read, write, and execute are checked relative to the real (not the effective) user ID and group ID of the current process.

If *name* begins with a slash, it is treated as an absolute path name, and *path* is ignored.

An empty *path* member is treated as the current directory. A slash (`/`) character is not prepended at the occurrence of the first match; rather, the unadorned *name* is returned.

Examples **EXAMPLE 1** Example of finding the `ls` command using the PATH environment variable.

To find the `ls` command using the `PATH` environment variable:

pathfind (getenv ("PATH"), "ls", "rx")

Return Values The `pathfind()` function returns a `(char *)` value containing static, thread-specific data that will be overwritten upon the next call from the same thread.

If the file *name* with all characteristics specified by *mode* is found in any of the directories specified by *path*, then `pathfind()` returns a pointer to a string containing the member of *path*, followed by a slash character (`/`), followed by *name*.

If no match is found, `pathname()` returns a null pointer, `((char *) 0)`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also `sh(1)`, `test(1)`, `access(2)`, `mknod(2)`, `stat(2)`, `getenv(3C)`, `attributes(5)`

Notes The string pointed to by the returned pointer is stored in an area that is reused on subsequent calls to `pathfind()`. The string should not be deallocated by the caller.

When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreadedapplications.

Name pow, powf, powl – power function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double pow(double x, double y);

float powf(float x, float y);

long double powl(long double x, long double y);

cc [ flag... ] file... -lm [ library... ]
#include <math.h>

double pow(double x, double y);

float powf(float x, float y);

long double powl(long double x, long double y);
```

Description These functions compute the value of x raised to the power y, x^y>. If x is negative, y must be an integer value.

Return Values Upon successful completion, these functions return the value of x raised to the power y.

For finite values of $x < 0$, and finite non-integer values of y, a domain error occurs and either a NaN (if representable), or an implementation-defined value is returned.

If the correct value would cause overflow, a range error occurs and `pow()`, `powf()`, and `powl()` return `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.

If x or y is a NaN, a NaN is returned unless:

- If x is +1 and y is NaN and the application was compiled with the `c99` compiler driver and is therefore SUSv3–conforming (see `standards(5)`), 1.0 is returned.
- For any value of x (including NaN), if y is +0, 1.0 is returned.

For any odd integer value of $y > 0$, if x is ±0, ±0 is returned.

For $y > 0$ and not an odd integer, if x is ±0, +0 is returned.

If x is ±1 and y is ±Inf, and the application was compiled with the `cc` compiler driver, NaN is returned. If, however, the application was compiled with the `c99` compiler driver and is therefore SUSv3-conforming (see`standards(5)`), 1.0 is returned.

For $|x| < 1$, if y is –Inf, +Inf is returned.

For $|x| > 1$, if y is –Inf, +0 is returned.

For $|x| < 1$, if y is +Inf, +0 is returned.

For $|x| > 1$, if y is +Inf, +Inf is returned.

For *y* an odd integer < 0, if *x* is –Inf, –0 is returned.

For *y* < 0 and not an odd integer, if *x* is –Inf, +0 is returned.

For *y* an odd integer > 0, if *x* is –Inf, –Inf is returned.

For *y* > 0 and not an odd integer, if *x* is –Inf, +Inf is returned.

For *y* < 0, if *x* is +Inf, +0 is returned.

For *y* > 0, if *x* is +Inf, +Inf is returned.

For *y* an odd integer < 0, if *x* is ±0, a pole error occurs and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL are returned for pow(), powf(), and powl(), respectively.

For *y* < 0 and not an odd integer, if *x* is ±0, a pole error occurs and HUGE_VAL, HUGE_VALF, and HUGE_VALL are returned for pow(), powf(), and powl(), respectively.

For exceptional cases, matherr(3M) tabulates the values to be returned by pow() as specified by SVID3 and XPG3.

Errors These functions will fail if:

Domain Error — The value of *x* is negative and *y* is a finite non-integer.

If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the invalid floating-point exception is raised.

The pow() function sets errno to EDOM if the value of *x* is negative and *y* is non-integral.

Pole Error — The value of *x* is 0 and *y* is negative.

If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the divide-by-zero floating-point exception is raised.

Range Error — The result overflows.

If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the overflow floating-point exception is raised.

The pow() function sets errno to EDOM if the value to be returned would cause overflow.

Usage An application wanting to check for exceptions should call feclearexcept(FE_ALL_EXCEPT) before calling these functions. On return, if fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `pow()`. On return, if `errno` is non-zero, an error has occurred. The `powf()` and `powl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `exp(3M)`, `feclearexcept(3M)`, `fetestexcept(3M)`, `isnan(3M)`, `math.h(3HEAD)`, `matherr(3M)`, `attributes(5)`, `standards(5)`

Notes Prior to Solaris 2.6, there was a conflict between the `pow()` function in this library and the `pow()` function in the `libmp` library. This conflict was resolved by prepending `mp_` to all functions in the `libmp` library. See `mp(3MP)` for more information.

Name regexpr, compile, step, advance – regular expression compile and match routines

Synopsis

```
cc  [flag]... [file]... -lgen [library]...

#include <regexpr.h>

char *compile(char *instring, char *expbuf, const char *endbuf);

int
step(const char *string, const char *expbuf);

int
advance(const char *string, const char *expbuf);

extern char *loc1, loc2, locs;

extern int nbra, regerrno, reglength;

extern char *braslist[], *braelist[];
```

Description These routines are used to compile regular expressions and match the compiled expressions against lines. The regular expressions compiled are in the form used by ed(1).

The parameter *instring* is a null-terminated string representing the regular expression.

The parameter *expbuf* points to the place where the compiled regular expression is to be placed. If *expbuf* is `NULL`, `compile()` uses malloc(3C) to allocate the space for the compiled regular expression. If an error occurs, this space is freed. It is the user's responsibility to free unneeded space after the compiled regular expression is no longer needed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. This argument is ignored if *expbuf* is `NULL`. If the compiled expression cannot fit in (*endbuf*–*expbuf*) bytes, `compile()` returns `NULL` and `regerrno` (see below) is set to 50.

The parameter *string* is a pointer to a string of characters to be checked for a match. This string should be null-terminated.

The parameter *expbuf* is the compiled regular expression obtained by a call of the function `compile()`.

The function `step()` returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to `step()`. The variables set in `step()` are `loc1` and `loc2`. `loc1` is a pointer to the first character that matched the regular expression. The variable `loc2` points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, `loc1` points to the first character of *string* and `loc2` points to the null at the end of *string*.

The purpose of `step()` is to step through the *string* argument until a match is found or until the end of *string* is reached. If the regular expression begins with `^`, `step()` tries to match the regular expression at the beginning of the string only.

The `advance()` function is similar to `step()`; but, it only sets the variable `loc2` and always restricts matches to the beginning of the string.

If one is looking for successive matches in the same string of characters, `locs` should be set equal to `loc2`, and `step()` should be called with *string* equal to `loc2`. `locs` is used by commands like `ed` and `sed` so that global substitutions like `s/y*//g` do not loop forever, and is `NULL` by default.

The external variable `nbra` is used to determine the number of subexpressions in the compiled regular expression. `braslist` and `braelist` are arrays of character pointers that point to the start and end of the `nbra` subexpressions in the matched string. For example, after calling `step()` or `advance()` with string `sabcdefg` and regular expression `\(abcdef\)`, `braslist[0]` will point at `a` and `braelist[0]` will point at `g`. These arrays are used by commands like `ed` and `sed` for substitute replacement patterns that contain the `\`*n* notation for subexpressions.

Note that it is not necessary to use the external variables `regerrno`, `nbra`, `loc1`, `loc2` `locs`, `braelist`, and `braslist` if one is only checking whether or not a string matches a regular expression.

Examples

EXAMPLE 1 The following is similar to the regular expression code from `grep`:

```
#include<regexpr.h>
 . . .
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
     regerr(regerrno);
 . . .
if (step(linebuf, expbuf))
   succeed( );
```

Return Values

If `compile()` succeeds, it returns a non-`NULL` pointer whose value depends on *expbuf*. If *expbuf* is non-`NULL`, `compile()` returns a pointer to the byte after the last byte in the compiled regular expression. The length of the compiled regular expression is stored in `reglength`. Otherwise, `compile()` returns a pointer to the space allocated by `malloc(3C)`.

The functions `step()` and `advance()` return non-zero if the given string matches the regular expression, and zero if the expressions do not match.

Errors

If an error is detected when compiling the regular expression, a `NULL` pointer is returned from `compile()` and `regerrno` is set to one of the non-zero error numbers indicated below:

ERROR	MEANING
11	Range endpoint too large.
16	Bad Number.
25	"\digit" out or range.
36	Illegal or missing delimiter.

ERROR	MEANING
41	No remembered string search.
42	\(~\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \[~\}.
45	} expected after \.
46	First number exceeds second in \{~\}.
49	[] imbalance.
50	Regular expression overflow.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also `ed(1)`, `grep(1)`, `sed(1)`, `malloc(3C)`, `attributes(5)`, `regexp(5)`

Notes When compiling multi-threaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-threaded applications.

Name remainder, remainderf, remainderl – remainder function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double remainder(double x, double y);

float remainderf(float x, float y);

long double remainderl(long double x, long double y);
```

Description These functions return the floating point remainder $r = x - ny$ when y is non-zero. The value n is the integral value nearest the exact value x/y. When $|n - x/y| = ½$, the value n is chosen to be even.

The behavior of `remainder()` is independent of the rounding mode.

Return Values Upon successful completion, these functions return the floating point remainder $r = x - ny$ when y is non-zero.

If x or y is NaN, a NaN is returned.

If x is infinite or y is 0 and the other is non-NaN, a domain error occurs and a NaN is returned.

Errors These functions will fail if:

Domain Error — The x argument is ±Inf, or the y argument is ±0 and the other argument is non-NaN.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception is raised.

The `remainder()` function sets `errno` to `EDOM` if y argument is 0 or the x argument is positive or negative infinity.

Usage An application wanting to check for error situations can set `errno` to 0 before calling `remainder()`. On return, if `errno` is non-zero, an error has occurred. The `remainderf()` and `remainderl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also abs(3C), div(3C), feclearexcept(3M), fetestexcept(3M), attributes(5), standards(5)

Name remquo, remquof, remquol – remainder functions

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double remquo(double x, double y, int *quo);

float remquof(float x, float y, int *quo);

long double remquol(long double x, long double y, int *quo);
```

Description The `remquo()`, `remquof()`, and `remquol()` functions compute the same remainder as the `remainder()`, `remainderf()`, and `remainderl()` functions, respectively. See `remainder(3M)`. In the object pointed to by *quo*, they store a value whose sign is the sign of *x*/*y* and whose magnitude is congruent modulo 2^n to the magnitude of the integral quotient of *x*/*y*, where *n* is an integer greater than or equal to 3.

Return Values These functions return *x* REM *y*.

If *x* or *y* is NaN, a NaN is returned.

If *x* is ±Inf or *y* is 0 and the other argument is non-NaN, a domain error occurs and a NaN is returned.

Errors These functions will fail if:

Domain Error — The *x* argument is Inf or the *y* argument is 0 and the other argument is non-NaN.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also feclearexcept(3M), fetestexcept(3M), math.h(3HEAD), remainder(3M), attributes(5), standards(5)

Name rint, rintf, rintl – round-to-nearest integral value

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double rint(double x);

float rintf(float x);

long double rintl(long double x);
```

Description These functions return the integral value (represented as a `double`) nearest *x* in the direction of the current rounding mode.

If the current rounding mode rounds toward negative infinity, `rint()` is equivalent to `floor`(3M). If the current rounding mode rounds toward positive infinity, `rint()` is equivalent to `ceil`(3M).

These functions differ from the `nearbyint`(3M), `nearbyintf()`, and `nearbyintl()` functions only in that they might raise the inexact floating-point exception if the result differs in value from the argument.

Return Values Upon successful completion, these functions return the integer (represented as a double precision number) nearest *x* in the direction of the current rounding mode.

If *x* is NaN, a NaN is returned.

If *x* is ±0 or ±Inf, *x* is returned.

Attributes See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards`(5).

See Also `abs`(3C), `ceil`(3M), `feclearexcept`(3M), `fetestexcept`(3M), `floor`(3M), `isnan`(3M), `math.h`(3HEAD), `nearbyint`(3M), `attributes`(5), `standards`(5)

Name round, roundf, roundl – round to nearest integer value in floating-point format

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double round(double x);

float roundf(float x);

long double roundl(long double x);
```

Description These functions round their argument to the nearest integer value in floating-point format, rounding halfway cases away from 0, regardless of the current rounding direction.

Return Values Upon successful completion, these functions return the rounded integer value.

If *x* is NaN, a NaN is returned.

If *x* is ±0 or ± Inf, *x* is returned.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), math.h(3HEAD), attributes(5), standards(5)

Name scalb, scalbf, scalbl – load exponent of a radix-independent floating-point number

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double scalb(double x, double n);

float scalbf(float x, float n);

long double scalbl(long double x, long double n);
```

Description These functions compute x * r^n, where *r* is the radix of the machine's floating point arithmetic. When *r* is 2, `scalb()` is equivalent to `ldexp(3M)`. The value of *r* is `FLT_RADIX` which is defined in `<float.h>`.

Return Values Upon successful completion, the `scalb()` function returns $x * r^n$.

If *x* or *n* is NaN, a NaN is returned.

If *n* is 0, *x* is returned.

If *x* is ±Inf and *n* is not –Inf, *x* is returned.

If *x* is ±0 and *n* is not +Inf, *x* is returned.

If *x* is ±0 and *n* is +Inf, a domain error occurs and a NaN is returned.

If *x* is ±Inf and *n* is –Inf, a domain error occurs and a NaN is returned.

If the result would cause an overflow, a range error occurs and ±`HUGE_VAL` (according to the sign of *x*) is returned.

For exceptional cases, `matherr(3M)` tabulates the values to be returned by `scalb()` as specified by SVID3 and XPG3. See `standards(5)`.

Errors These functions will fail if:

Domain Error — If *x* is 0 and *n* is +Inf, or *x* is Inf and *n* is –Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception is raised.

Range Error — The result would overflow.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the overflow floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect

exceptions.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	For scalb(), see standards(5).

See Also feclearexcept(3M), fetestexcept(3M), ilogb(3M), ldexp(3M), logb(3M), math.h(3HEAD), matherr(3M), scalbln(3M), attributes(5), standards(5)

Name scalbln, scalblnf, scalblnl, scalbn, scalbnf, scalbnl – compute exponent using FLT_RADIX

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double scalbln(double x, long n);

float scalblnf(float x, long n);

long double scalblnl(long double x, long n);

double scalbn(double x, int n);

float scalbnf(float x, int n);

long double scalbnl(long double x, int n);
```

Description These functions compute $x * \texttt{FLT\_RADIX}^{n}$ efficiently, not normally by computing $\texttt{FLT\_RADIX}^{n}$ explicitly.

Return Values Upon successful completion, these functions return $x * \texttt{FLT\_RADIX}^{n}$>.

If the result would cause overflow, a range error occurs and these functions return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (according to the sign of *x*) as appropriate for the return type of the function.

If *x* is NaN, a NaN is returned.

If *x* is ±0 or ±Inf, *x* is returned.

If *x* is 0, *x* is returned.

Errors These functions will fail if:

Range Error — The result overflows.

If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception is raised.

Usage An application wanting to check for exceptions should call feclearexcept(FE_ALL_EXCEPT) before calling these functions. On return, if fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), math.h(3HEAD), scalb(3M), attributes(5), standards(5)

Name signbit – test sign

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

int signbit(real-floating x);
```

Description The `signbit()` macro determines whether the sign of its argument value is negative. NaNs, zeros, and infinities have a sign bit.

Return Values The `signbit()` macro returns a non-zero value if and only if the sign of its argument value is negative.

Errors No errors are defined.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `fpclassify(3M)`, `isfinite(3M)`, `isinf(3M)`, `isnan(3M)`, `isnormal(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name significand, significandf, significandl – significand function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double significand(double x);

float significandf(float x);

long double significandl(long double x);
```

Description If x equals $sig * 2^n$ with $1 \leq sig < 2$, then these functions return sig.

Return Values Upon successful completion, these functions return *sig*.

If x is either 0, ±Inf or NaN, x is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also logb(3M), scalb(3M), attributes(5)

Name sin, sinf, sinl – sine function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double sin(double x);

float sinf(float x);

long double sinl(long double x);
```

Description These functions compute the sine of its argument *x*, measured in radians.

Return Values Upon successful completion, these functions return the sine of *x*.

If *x* is NaN, a NaN is returned.

If *x* is ±0, *x* is returned.

If *x* is ±Inf, a domain error occurs and a NaN is returned.

Errors These functions will fail if:

Domain Error — The *x* argument is ±Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `asin(3M)`, `feclearexcept(3M)`, `fetestexcept(3M)`, `isnan(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name sincos, sincosf, sincosl – combined sine and cosine function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

void sincos(double x, double *s, double *c);

void sincosf(float x, float *s, float *c);

void sincosl(long double x, long double *s, long double *c);
```

Description These functions compute the sine and cosine of the first argument *x*, measured in radians.

Return Values Upon successful completion, these functions return the sine of *x* in **s* and cosine of *x* in **c*.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also cos(3M), sin(3M), math.h(3HEAD), attributes(5)

Name sinh, sinhf, sinhl – hyperbolic sine function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double sinh(double x);

float sinhf(float x);

long double sinhl(long double x);
```

Description These functions compute the hyperbolic sine of *x*.

Return Values Upon successful completion, these functions return the hyperbolic sine of *x*.

If the result would cause an overflow, a range error occurs and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*) is returned as appropriate for the type of the function.

If *x* is NaN, a NaN is returned.

If *x* is ±0 or ±Inf, *x* is returned.

For exceptional cases, matherr(3M) tabulates the values to be returned by acos() as specified by SVID3 and XPG3.

Errors These functions will fail if:

Range Error — The result would cause an overflow.

If the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the overflow floating-point exception is raised.

The asinh() function sets errno to ERANGE if the result would cause an overflow.

Usage An application wanting to check for exceptions should call feclearexcept(FE_ALL_EXCEPT) before calling these functions. On return, if fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set errno to 0 before calling asinh(). On return, if errno is non-zero, an error has occurred. The asinhf() and asinhl() functions do not set errno.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe
Standard	See standards(5).

See Also asinh(3M), cosh(3M), feclearexcept(3M), fetestexcept(3M), isnan(3M), math.h(3HEAD), matherr(3M), tanh(3M), attributes(5), standards(5)

Name sqrt, sqrtf, sqrtl – square root function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double sqrt(double x);

float sqrtf(float x);

long double sqrtl(long double x);
```

Description These functions compute the square root of their argument *x*.

Return Values Upon successful completion, these functions return the square root of *x*.

For finite values of $x < -0$, a domain error occurs and either a NaN (if supported) or an implementation-defined value is returned.

If *x* is NaN, a NaN is returned.

If *x* is ±0 or +Inf, *x* is returned.

If *x* is –Inf, a domain error occurs and a NaN is returned.

Errors These functions will fail if:

Domain Error — The finite value of *x* is < -0 or *x* is –Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

The `sqrt()` function sets `errno` to `EDOM` if the value of *x* is negative.

Usage An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

An application can also set `errno` to 0 before calling `sqrt()`. On return, if `errno` is non-zero, an error has occurred. The `sqrtf()` and `sqrtl()` functions do not set `errno`.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also feclearexcept(3M), fetestexcept(3M), isnan(3M), math.h(3HEAD), attributes(5), standards(5)

Name strccpy, streadd, strcadd, strecpy – copy strings, compressing or expanding escape codes

Synopsis

```
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

char *strccpy(char *output, const char *input);

char *strcadd(char *output, const char *input);

char *strecpy(char *output, const char *input, const char *exceptions);

char *streadd(char *output, const char *input, const char *exceptions);
```

Description `strccpy()` copies the *input* string, up to a null byte, to the *output* string, compressing the C-language escape sequences (for example, `\n`, `\001`) to the equivalent character. A null byte is appended to the output. The *output* argument must point to a space big enough to accommodate the result. If it is as big as the space pointed to by *input* it is guaranteed to be big enough. `strccpy()` returns the *output* argument.

`strcadd()` is identical to `strccpy()`, except that it returns the pointer to the null byte that terminates the output.

`strecpy()` copies the *input* string, up to a null byte, to the *output* string, expanding non-graphic characters to their equivalent C-language escape sequences (for example, `\n`, `\001`). The *output* argument must point to a space big enough to accommodate the result; four times the space pointed to by *input* is guaranteed to be big enough (each character could become `\` and 3 digits). Characters in the *exceptions* string are not expanded. The *exceptions* argument may be zero, meaning all non-graphic characters are expanded. `strecpy()` returns the *output* argument.

`streadd()` is identical to `strecpy()`, except that it returns the pointer to the null byte that terminates the output.

Examples **EXAMPLE 1** Example of expanding and compressing escape codes.

```
/* expand all but newline and tab */
strecpy( output, input, "\n\t" );

/* concatenate and compress several strings */
cp = strcadd( output, input1 );
cp = strcadd( cp, input2 );
cp = strcadd( cp, input3 );
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also `string(3C)`, `strfind(3GEN)`, `attributes(5)`

Notes When compiling multi-thread applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multi-thread applications.

Name strfind, strrspn, strtrns – string manipulations

Synopsis

```
cc [ flag ... ] file ... -lgen [ library ... ]
#include <libgen.h>

int strfind(const char *as1, const char *as2);

char *strrspn(const char *string, const char *tc);

char * strtrns(const char *string, const char *old,
     const char *new, char *result);
```

Description The `strfind()` function returns the offset of the first occurrence of the second string, *as2*, if it is a substring of string *as1*. If the second string is not a substring of the first string `strfind()` returns −1.

The `strrspn()` function trims chartacters from a string. It searches from the end of *string* for the first character that is not contained in *tc*. If such a character is found, `strrspn()` returns a pointer to the next character; otherwise, it returns a pointer to *string*.

The `strtrns()` function transforms *string* and copies it into *result*. Any character that appears in *old* is replaced with the character in the same position in *new*. The *new* result is returned.

Usage When compiling multithreaded applications, the `_REENTRANT` flag must be defined on the compile line. This flag should only be used in multithreaded applications.

Examples **EXAMPLE 1** An example of the `strfind()` function.

```
/* find offset to substring "hello" within as1 */
i = strfind(as1, "hello");
/* trim junk from end of string */
s2 = strrspn(s1, "*?#$%");
*s2 = '\0';
/* transform lower case to upper case */
a1[] = "abcdefghijklmnopqrstuvwxyz";
a2[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
s2 = strtrns(s1, a1, a2, s2);
```

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

See Also `string(3C)`, `attributes(5)`

Name Sun_MP_SendScsiCmd – send a SCSI command to a logical unit

Synopsis

```
cc [ flag... ] file... -lMPAPI [ library... ]
#include <mpapi.h>
#include <mpapi_sun.h>

MP_STATUS MP_SendScsiCmd(MP_OID oid, struct uscsi_cmd *cmd);
```

Parameters

oid The object ID of the logical unit path.

cmd A `uscsi_cmd` structure. See `uscsi(7I)`.

Description The `Sun_MP_SendScsiCmd()` function sends a SCSI command on a specific path to a logical unit. This function is applicable only to an OID whose `MP_PLUGIN_PROPERTIES` *driverVendor*, as defined by the Multipath Management API, is equal to "Sun Microsystems". See MP_GetPluginProperties(3MPAPI) and *Multipath Management API Version 1.0*.

Return Values

`MP_STATUS_INVALID_PARAMETER`
The *pProps* is null or specifies a memory area to which data cannot be written, or the *oid* has a type subfield other than `MP_OBJECT_TYPE_PLUGIN`.

`MP_STATUS_INVALID_OBJECT_TYPE`
The *oid* does not specify any valid object type. This is most likely to happen if an uninitialized object ID is passed to the API.

`MP_STATUS_OBJECT_NOT_FOUND`
The *oid* owner ID or object sequence number is invalid.

`MP_STATUS_SUCCESS`
The operation is successful.

Warnings The `uscsi` command is very powerful but somewhat dangerous. See the WARNINGS section on `attributes(5)` before using this interface.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	Safe

See Also `libMPAPI(3LIB)`, MP_GetPluginProperties(3MPAPI), `attributes(5)`, `uscsi(7I)`

Multipath Management API Version 1.0

Name tan, tanf, tanl – tangent function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double tan(double x);

float tanf(float x);

long double tanl(long double x);
```

Description These functions compute the tangent of their argument *x*, measured in radians.

Return Values Upon successful completion, these functions return the tangent of *x*.

If *x* is NaN, a NaN is returned.

If *x* is ±0, *x* is returned.

If *x* is ±Inf, a domain error occurs and a NaN is returned.

Errors These functions will fail if:

Domain Error — The value of *x* is ±Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, the invalid floating-point exception is raised.

Usage There are no known floating-point representations such that for a normal argument, `tan`(*x*) is either overflow or underflow.

An application wanting to check for exceptions should call `feclearexcept`(`FE_ALL_EXCEPT`) before calling these functions. On return, if `fetestexcept`(`FE_INVALID` | `FE_DIVBYZERO` | `FE_OVERFLOW` | `FE_UNDERFLOW`) is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See `standards(5)`.

See Also `atan(3M)`, `feclearexcept(3M)`, `fetestexcept(3M)`, `isnan(3M)`, `math.h(3HEAD)`, `attributes(5)`, `standards(5)`

Name tanh, tanhf, tanhl – hyperbolic tangent function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double tanh(double x);

float tanhf(float x);

long double tanhl(long double x);
```

Description These functions compute the hyperbolic tangent of their argument *x*.

Return Values Upon successful completion, these functions return the hyperbolic tangent of *x*.

If *x* is NaN, a NaN is returned.

If *x* is ±0, *x* is returned.

If *x* is ±Inf, ±1 is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also atanh(3M), isnan(3M), math.h(3HEAD), tan(3M), attributes(5), standards(5)

Name tgamma, tgammaf, tgammal – compute gamma function

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double tgamma(double x);

float tgammaf(float x);

long double tgammal(long double x);
```

Description These functions compute the `gamma()` function of *x*.

Return Values Upon successful completion, these functions return `gamma`(*x*).

If *x* is a negative integer, a domain error occurs and a NaN is returned.

If the correct value would cause overflow, a range error occurs and `tgamma()`, `tgammaf()`, and `tgammal()` return the value of the macro ±`HUGE_VAL`, ±`HUGE_VALF`, or ±`HUGE_VALL`, respectively.

If *x* is NaN, a NaN is returned.

If *x* is ±Inf, *x* is returned.

If x is ±0, a pole error occurs and `tgamma()`, `tgammaf()`, and `tgammal()` return ±`HUGE_VAL`, ±`HUGE_VALF`, and ±`HUGE_VALL`, respectively.

If *x* is +Inf, a domain error occurs and a NaN is returned.

Errors These functions will fail if:

Domain Error — The value of *x* is a negative integer or *x* is –Inf.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception is raised.

Pole Error — The value of *x* is zero.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the divide–by-zero floating-point exception is raised.

Range Error — The value overflows.

If the integer expression (`math_errhandling` & `MATH_ERREXCEPT`) is non-zero, then the overflow floating-point exception is raised.

Usage An application wanting to check for exceptions should call `feclearexcept`(`FE_ALL_EXCEPT`) before calling these functions. On return, if `fetestexcept`(`FE_INVALID` | `FE_DIVBYZERO` | `FE_OVERFLOW` | `FE_UNDERFLOW`) is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect

exceptions.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also feclearexcept(3M), fetestexcept(3M), lgamma(3M), math.h(3HEAD), attributes(5), standards(5)

Name trunc, truncf, truncl – round to truncated integer value

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double trunc(double x);

float truncf(float x);

long double truncl(long double x);
```

Description These functions round their argument to the integer value, in floating format, nearest to but no larger in magnitude than the argument.

Return Values Upon successful completion, these functions return the truncated integer value.

If *x* is NaN, a NaN is returned.

If *x* is ±0 or ±Inf, *x* is returned.

Errors No errors are defined.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See standards(5).

See Also math.h(3HEAD), attributes(5), standards(5)

Name vatan2_, vatan2f_ – vector atan2 functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vatan2_(int *n, double * restrict y, int *stridey,
     double * restrict x, int *stridex, double * restrict z,
     int *stridez);

void vatan2f_(int *n, float * restrict y, int *stridey,
     float * restrict x, int *stridex, float * restrict z,
     int *stridez);
```

Description These functions evaluate the function `atan2`(y, x) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vatan2_` (n, y, sy, x, sx, z, sz) computes $z[i * *sz] =$ `atan2`($y[i * *sy]$, $x[i * *sx]$) for each $i = 0, 1, ..., *n - 1$. The `vatan2f_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the atan2(3M) functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count $*n$ must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the same way as the `atan2()` functions when `c99 MATHERREXCEPT` conventions are in effect. See atan2(3M) for the results for special cases.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the

numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also atan2(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vatan_, vatanf_ – vector arctangent functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vatan_(int *n,double * restrict x, int *stridex,
     double * restrict y, int *stridey);

void vatanf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey);
```

Description These functions evaluate the function `atan`(*x*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vatan_`(*n*, *x*, *sx*, *y*, *sy*) computes *y*[*i* * *sy*] = `atan`(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vatanf_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the `atan`(3M) functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the same way as the `atan()` functions when `c99` `MATHERREXCEPT` conventions are in effect. See `atan`(3M) for the results for special cases.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also atan(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vcos_, vcosf_ – vector cosine functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vcos_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey);

void vcosf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey);
```

Description These functions evaluate the function `cos`(*x*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vcos_`(*n*, *x*, *sx*, *y*, *sy*) computes *y*[*i* * *sy*] = `cos`(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vcosf_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the cos(3M) functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the same way as the `cos()` functions when `c99` `MATHERREXCEPT` conventions are in effect. See cos(3M) for the results for special cases.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also cos(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vcospi_, vcospif_ – vector cospi functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vcospi_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey);

void vcosfpi_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey);
```

Description These functions evaluate the function `cospi`(*x*), defined by `cospi`(*x*) = `cos`(π * *x*), for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vcospi_`(*n*, *x*, *sx*, *y*, *sy*) computes *y*[*i* * *sy*] = `cospi`(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vcospif_()` function performs the same computation for single precision data.

Non-exceptional results are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the spirit of IEEE 754. In particular,

- `cospi`(NaN) is NaN,
- `cospi`(±Inf) is NaN, and an invalid operation exception is raised.

An application wanting to check for exceptions should call `feclearexcept`(`FE_ALL_EXCEPT`) before calling these functions. On return, if `fetestexcept`(`FE_INVALID` | `FE_DIVBYZERO` | `FE_OVERFLOW` | `FE_UNDERFLOW`) is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vexp_, vexpf_ – vector exponential functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vexp_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey);

void vexpf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey);
```

Description These functions evaluate the function exp(*x*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, vexp_(*n*, *x*, *sx*, *y*, *sy*) computes *y*[*i* * *sy*] = exp(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The vexpf_() function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the exp(3M) functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

On SPARC, the vexpf_() function delivers +0 rather than a subnormal result for arguments in the range –103.2789 <= *x* <= -87.3365. Otherwise, these functions handle special cases and exceptions in the same way as the exp() functions when c99 MATHERREXCEPT conventions are in effect. See exp(3M) for the results for special cases.

An application wanting to check for exceptions should call feclearexcept(FE_ALL_EXCEPT) before calling these functions. On return, if fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also exp(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vhypot_, vhypotf_ – vector hypotenuse functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vhypot_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey, double * restrict z,
     int *stridez);

void vhypotf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey, float * restrict z,
     int *stridez);
```

Description These functions evaluate the function `hypot`(*x*, *y*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vhypot_`(*n*, *x*, *sx*, *y*, *sy*, *z*, *sz*) computes *z*[*i* * *sz*] = `hypot`(*x*[*i* * *sx*], *y*[*i* * *sy*]) for each *i* = 0, 1, ..., **n* - 1. The `vhypotf_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the `hypot`(3M) functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the same way as the `hypot()` functions when `c99 MATHERREXCEPT` conventions are in effect. See `hypot`(3M) for the results for special cases.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the

numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also hypot(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vlog_, vlogf_ – vector logarithm functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vlog_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey);

void vlogf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey);
```

Description These functions evaluate the function `log`(*x*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vlog_`(*n*, *x*, *sx*, *y*, *sy*) computes *y*[*i* * *sy*] = `log`(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vlogf_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the log(3M) functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the same way as the `log()` functions when `c99` `MATHERREXCEPT` conventions are in effect. See log(3M) for the results for special cases.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also log(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vpow_, vpowf_ – vector power functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vpow_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey, double * restrict z,
     int *stridez);

void vpowf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey, float * restrict z,
     int *stridez);
```

Description These functions evaluate the function pow(*x*, *y*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, vpow_(*n*, *x*, *sx*, *y*, *sy*, *z*, *sz*) computes *z*[*i* * *sz*] = pow(*x*[*i* * *sx*], *y*[*i* * *sy*]) for each *i* = 0, 1, ..., **n* - 1. The vpowf_() function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the pow(3M) functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

The results of these functions for special cases and exceptions match that of the pow() functions when the latter are used in a program compiled with the cc compiler driver (that is, not SUSv3-conforming) and the expression (math_errhandling & MATH_ERREXCEPT) is non-zero. These functions do not set errno. See pow(3M) for the results for special cases.

An application wanting to check for exceptions should call feclearexcept(FE_ALL_EXCEPT) before calling these functions. On return, if fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can

raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also pow(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vrhypot_, vrhypotf_ – vector reciprocal hypotenuse functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vrhypot_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey, double * restrict z,
     int *stridez);

void vrhypotf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey, float * restrict z,
     int *stridez);
```

Description These functions evaluate the function rhypot(*x*, *y*), defined by rhypot(*x*, *y*) = 1 / hypot(*x*, *y*), for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, vrhypot_(*n*, *x*, *sx*, *y*, *sy*, *z*, *sz*) computes *z*[*i* * *sz*] = rhypot(*x*[*i* * *sx*], *y*[*i* * *sy*]) for each *i* = 0, 1, ..., **n* - 1. The vrhypotf_() function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of evaluating 1.0 / hypot(*x*, *y*) given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the spirit of IEEE 754. In particular,

- if x or *y* is ±Inf, rhypot(*x*, *y*) is +0, even if the other of *x* or *y* is NaN,
- if x or *y* is NaN and neither is infinite, rhypot(*x*, *y*) is NaN
- if *x* and *y* are both zero, rhypot(*x*, *y*) is +0, and a division-by-zero exception is raised.

An application wanting to check for exceptions should call feclearexcept(FE_ALL_EXCEPT) before calling these functions. On return, if fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an exception has been raised. The application can

then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also hypot(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vrsqrt_, vrsqrtf_ – vector reciprocal square root functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vrsqrt_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey);

void vrsqrtf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey);
```

Description These functions evaluate the function `rsqrt`(*x*), defined by `rsqrt`(*x*) = 1 / `sqrt`(*x*), for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vrsqrt_`(*n*, *x*, *sx*, *y*, *sy*) computes *y*[*i* * *sy*] = `rsqrt`(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vrsqrtf_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of evaluating 1.0 / `sqrt`(*x*) given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the spirit of IEEE 754. In particular,

- if *x* < 0, `rsqrt`(*x*) is NaN, and an invalid operation exception is raised,
- `rsqrt`(NaN) is NaN,
- `rsqrt`(+Inf) is +0,
- `rsqrt`(±0) is ±Inf, and a division-by-zero exception is raised.

An application wanting to check for exceptions should call `feclearexcept`(`FE_ALL_EXCEPT`) before calling these functions. On return, if `fetestexcept`(`FE_INVALID` | `FE_DIVBYZERO` | `FE_OVERFLOW` | `FE_UNDERFLOW`) is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the

numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also sqrt(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vsin_, vsinf_ – vector sine functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vsin_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey);

void vsinf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey);
```

Description These functions evaluate the function `sin`(*x*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vsin_`(*n*, *x*, *sx*, *y*, *sy*) computes *y*[*i* * *sy*] = `sin`(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vsinf_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the sin(3M) functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the same way as the `sin()` functions when `c99` `MATHERREXCEPT` conventions are in effect. See sin(3M) for the results for special cases.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also sin(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vsincos_, vsincosf_ – vector sincos functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vsincos_(int *n, double * restrict x, int *stridex,
     double * restrict s, int *strides, double * restrict c,
     int *stridec);

void vsincosf_(int *n, float * restrict x, int *stridex,
     float * restrict s, int *strides, float * restrict c,
     int *stridec);
```

Description These functions evaluate both `sin`(*x*) and `cos`(*x*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vsincos_`(*n*, *x*, *sx*, *s*, *ss*, *c*, *sc*) simultaneously computes *s*[*i* * *ss*] = `sin`(*x*[*i* * *sx*]) and *c*[*i* * *sc*] = `cos`(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vsincosf_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the `sincos(3M)` functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the same way as the `sin()` and `cos()` functions when `c99` `MATHERREXCEPT` conventions are in effect. See `sin(3M)` and `cos(3M)` for the results for special cases.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the

numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also cos(3M), sin(3M), sincos(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vsincospi_, vsincospif_ – vector sincospi functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vsincospi_(int *n, double * restrict x, int *stridex,
     double * restrict s, int *strides, double * restrict c,
     int *stridec);

void vsincospif_(int *n, float * restrict x, int *stridex,
     float * restrict s, int *strides, float * restrict c,
     int *stridec);
```

Description These functions evaluate both `sinpi`(*x*) and `cospi`(*x*), defined by `sinpi`(*x*) = `sin`(π * *x*) and `cospi`(*x*) = `cos`(π * *x*), for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vsincospi_`(*n*, *x*, *sx*, *s*, *ss*, *c*, *sc*) simultaneously computes *s*[*i* * **ss*] = `sinpi`(*x*[*i* * **sx*]) and *c*[*i* * **sc*] = `cospi`(*x*[*i* * **sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vsincosf_()` function performs the same computation for single precision data.

Non-exceptional results are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the spirit of IEEE 754. In particular,

- `sinpi`(NaN), `cospi`(NaN) are NaN,
- `sinpi`(±0) is ±0,
- `sinpi`(±Inf), `cospi`(±Inf) are NaN, and an invalid operation exception is raised.

An application wanting to check for exceptions should call `feclearexcept`(`FE_ALL_EXCEPT`) before calling these functions. On return, if `fetestexcept`(`FE_INVALID` | `FE_DIVBYZERO` | `FE_OVERFLOW` | `FE_UNDERFLOW`) is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can

raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vsinpi_, vsinpif_ – vector sinpi functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vsinpi_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey);

void vsinpif_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey);
```

Description These functions evaluate the function `sinpi`(*x*), defined by `sinpi`(*x*) = `sin`(π * *x*), for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vsinpi_`(*n*, *x*, *sx*, *y*, *sy*) computes *y*[*i* * *sy*] = `sinpi`(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vsinpif_()` function performs the same computation for single precision data.

Non-exceptional results are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the spirit of IEEE 754. In particular,

- `sinpi`(NaN) is NaN,
- `sinpi`(±0) is ±0,
- `sinpi`(±Inf) is NaN, and an invalid operation exception is raised.

An application wanting to check for exceptions should call `feclearexcept`(`FE_ALL_EXCEPT`) before calling these functions. On return, if `fetestexcept`(`FE_INVALID` | `FE_DIVBYZERO` | `FE_OVERFLOW` | `FE_UNDERFLOW`) is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vsqrt_, vsqrtf_ – vector square root functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vsqrt_(int *n, double * restrict x, int *stridex,
     double * restrict y, int *stridey);

void vsqrtf_(int *n, float * restrict x, int *stridex,
     float * restrict y, int *stridey);
```

Description These functions evaluate the function `sqrt`(*x*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vsqrt_`(*n*, *x*, *sx*, *y*, *sy*) computes *y*[*i* * *sy*] = `sqrt`(*x*[*i* * *sx*]) for each *i* = 0, 1, ..., **n* - 1. The `vsqrtf_()` function performs the same computation for single precision data.

Unlike their scalar counterparts, these functions do not always deliver correctly rounded results. However, the error in each non-exceptional result is less than one unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the same way as the `sqrt()` functions when `c99 MATHERREXCEPT` conventions are in effect. See `sqrt(3M)` for the results for special cases.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also sqrt(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vz_abs_, vc_abs_ – vector complex absolute value functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vz_abs_(int *n, double complex * restrict z,
     int *stridez, double * restrict y, int *stridey);

void vc_abs_(int *n, float complex * restrict z,
     int *stridez, float * restrict y, int *stridey);
```

Description These functions compute the magnitude (or modulus) | *z* | for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, `vz_abs_`(*n*, *z*, *sz*, *y*, *sy*) computes *y*[*i* * *sy*] = | *z*[*i* * *sz*] | for each *i* = 0, 1, ..., **n* - 1. The `vc_abs_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the `cabs`(3M) functions given the same arguments. Non-exceptional results, however, are accurate to within a unit in the last place.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

These functions handle special cases and exceptions in the spirit of IEEE 754. See `cabs`(3M) for the results for special cases.

An application wanting to check for exceptions should call `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-zero, an exception has been raised. The application can then examine the result or argument vectors for exceptional values. Some vector functions can raise the inexact exception even if all elements of the argument array are such that the numerical results are exact.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also cabs(3M), feclearexcept(3M), fetestexcept(3M), attributes(5)

Name vz_exp_, vc_exp_ – vector complex exponential functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vz_exp_(int *n, double complex * restrict z,
     int *stridez, double  complex * restrict w int *stridew,
     double * tmp);

void vc_exp_(int *n, float complex * restrict z,
     int *stridez, float complex * restrict w, int *stridew,
     float * tmp);
```

Description These functions evaluate the complex function exp(*z*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements. The last argument is a pointer to scratch storage; this storage must be large enough to hold **n* consecutive values of the real type corresponding to the complex type of the argument and result.

Specifically, vz_exp_(*n*, *z*, *sz*, *w*, *sw*, *tmp*) computes *w*[*i* * **sw*] = exp(*z*[*i* * **sz*]) for each *i* = 0, 1, ..., **n* - 1. The vc_exp_() function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the cexp(3M) functions given the same arguments.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

Unlike the c99 cexp(3M) functions, the vector complex exponential functions make no attempt to handle special cases and exceptions; they simply use textbook formulas to compute a complex exponential in terms of real elementary functions. As a result, these functions can raise different exceptions and/or deliver different results from cexp().

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also cexp(3M), attributes(5)

Name vz_log_, vc_log_ – vector complex logarithm functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vz_log_(int *n, double complex * restrict z,
     int *stridez, double _complex * restrict w, int *stridew);

void vc_log_(int *n, float complex * restrict z,
     int *stridez, float complex * restrict w, int *stridew);
```

Description These functions evaluate the complex function log(*z*) for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements.

Specifically, vz_log_(*n*, *z*, *sz*, *w*, *sw*) computes *w*[*i* * *sw*] = log(*z*[*i* * *sz*]) for each *i* = 0, 1, ..., **n* - 1. The vc_log_() function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the clog(3M) functions given the same arguments.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

Unlike the c99 clog(3M) functions, the vector complex exponential functions make no attempt to handle special cases and exceptions; they simply use textbook formulas to compute a complex exponential in terms of real elementary functions. As a result, these functions can raise different exceptions and/or deliver different results from clog().

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also clog(3M), attributes(5)

Name vz_pow_, vc_pow_ – vector complex power functions

Synopsis

```
cc [ flag... ] file... -lmvec [ library... ]

void vz_pow_(int *n, double complex * restrict z,
     int *stridez, double complex * restrict w, int *stridew,
     double complex * restrict u, int *strideu,
     double * tmp);

void vc_pow_(int *n, float complex * restrict z,
     int *stridez, float complex * restrict w, int *stridew,
     float complex * restrict u, int *strideu,
     float * tmp);
```

Description These functions evaluate the complex function *z*^*w* for an entire vector of values at once. The first parameter specifies the number of values to compute. Subsequent parameters specify the argument and result vectors. Each vector is described by a pointer to the first element and a stride, which is the increment between successive elements. The last argument is a pointer to scratch storage; this storage must be large enough to hold 3 * **n* consecutive values of the real type corresponding to the complex type of the argument and result.

Specifically, `vz_pow_`(*n*, *z*, *sz*, *w*, *sw*, *u*, *su*, *tmp*) computes *u*[*i* * **su*] = (*z*[*i* * **sz*])^(*w*[*i* * **sw*]) for each *i* = 0, 1, ..., **n* - 1. The `vc_pow_()` function performs the same computation for single precision data.

These functions are not guaranteed to deliver results that are identical to the results of the cpow(3M) functions given the same arguments.

Usage The element count **n* must be greater than zero. The strides for the argument and result arrays can be arbitrary integers, but the arrays themselves must not be the same or overlap. A zero stride effectively collapses an entire vector into a single element. A negative stride causes a vector to be accessed in descending memory order, but note that the corresponding pointer must still point to the first element of the vector to be used; if the stride is negative, this will be the highest-addressed element in memory. This convention differs from the Level 1 BLAS, in which array parameters always refer to the lowest-addressed element in memory even when negative increments are used.

These functions assume that the default round-to-nearest rounding direction mode is in effect. On x86, these functions also assume that the default round-to-64-bit rounding precision mode is in effect. The result of calling a vector function with a non-default rounding mode in effect is undefined.

Unlike the c99 cpow(3M) functions, the vector complex exponential functions make no attempt to handle special cases and exceptions; they simply use textbook formulas to compute a complex exponential in terms of real elementary functions. As a result, these functions can raise different exceptions and/or deliver different results from `cpow()`.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe

See Also cpow(3M), attributes(5)

Name y0, y0f, y0l, y1, y1f, y1l, yn, ynf, ynl – Bessel functions of the second kind

Synopsis

```
c99 [ flag... ] file... -lm [ library... ]
#include <math.h>

double y0(double x);

float y0f(float x);

long double y0l(long double x);

double y1(double x);

float y1f(float x);

long double y1l(long double x);

double yn(int n, double x);

float ynf(int n, float x);

long double ynl(int n, long double x);
```

Description These functions compute Bessel functions of *x* of the second kind of orders 0, 1 and *n*, respectively.

Return Values Upon successful completion, these functions return the relevant Bessel value of *x* of the second kind.

If *x* is NaN, a NaN is returned.

If *x* is negative, –HUGE_VAL or NaN is returned.

If *x* is 0.0, –HUGE_VAL is returned.

If the correct result would cause overflow, –HUGE_VAL is returned.

For exceptional cases, matherr(3M) tabulates the values to be returned as specified by SVID3 and XPG3.

Errors No errors are returned.

Usage An application wanting to check for exceptions should call feclearexcept(FE_ALL_EXCEPT) before calling these functions. On return, if fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an exception has been raised. An application should either examine the return value or check the floating point exception flags to detect exceptions.

Attributes See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Committed
MT-Level	MT-Safe
Standard	See below.

For `y0()`, `y1()`, and `yn()`, see `standards(5)`

See Also `isnan(3M)`, `feclearexcept(3M)`, `fetestexcept(3M)`, `j0(3M)`, `math.h(3HEAD)`, `matherr(3M)`, `attributes(5)`, `standards(5)`