NAME

POSIX - Perl interface to IEEE Std 1003.1

SYNOPSIS

```
    use POSIX ();
    use POSIX qw(setsid);
    use POSIX qw(:errno_h :fcntl_h);

    printf "EINTR is %d\n", EINTR;

    $sess_id = POSIX::setsid();

    $fd = POSIX::open($path, O_CREAT|O_EXCL|O_WRONLY, 0644);
 # note: that's a filedescriptor, *NOT* a filehandle
```

DESCRIPTION

The POSIX module permits you to access all (or nearly all) the standard POSIX 1003.1 identifiers. Many of these identifiers have been given Perl-ish interfaces.

Everything is exported by default with the exception of any POSIX functions with the same name as a built-in Perl function, such as `abs`, `alarm`, `rmdir`, `write`, etc.., which will be exported only if you ask for them explicitly. This is an unfortunate backwards compatibility feature. You can stop the exporting by saying `use POSIX ()` and then use the fully qualified names (*e.g.*, `POSIX::SEEK_END`), or by giving an explicit import list. If you do neither, and opt for the default, `use POSIX;` has to import *553 symbols*.

This document gives a condensed list of the features available in the POSIX module. Consult your operating system's manpages for general information on most features. Consult *perlfunc* for functions which are noted as being identical to Perl's builtin functions.

The first section describes POSIX functions from the 1003.1 specification. The second section describes some classes for signal objects, TTY objects, and other miscellaneous objects. The remaining sections list various constants and macros in an organization which roughly follows IEEE Std 1003.1b-1993.

CAVEATS

A few functions are not implemented because they are C specific. If you attempt to call these, they will print a message telling you that they aren't implemented, and suggest using the Perl equivalent, should one exist. For example, trying to access the `setjmp()` call will elicit the message "`setjmp() is C-specific: use eval {} instead`".

Furthermore, some evil vendors will claim 1003.1 compliance, but in fact are not so: they will not pass the PCTS (POSIX Compliance Test Suites). For example, one vendor may not define `EDEADLK`, or the semantics of the errno values set by `open(2)` might not be quite right. Perl does not attempt to verify POSIX compliance. That means you can currently successfully say "use POSIX", and then later in your program you find that your vendor has been lax and there's no usable `ICANON` macro after all. This could be construed to be a bug.

FUNCTIONS

`_exit`

This is identical to the C function `_exit()`. It exits the program immediately which means among other things buffered I/O is **not** flushed.

Note that when using threads and in Linux this is **not** a good way to exit a thread because in Linux processes and threads are kind of the same thing (Note: while this is the situation in early 2003 there are projects under way to have threads with more

POSIXly semantics in Linux). If you want not to return from a thread, detach the thread.

`abort`

This is identical to the C function `abort()`. It terminates the process with a `SIGABRT` signal unless caught by a signal handler or if the handler does not return normally (it e.g. does a `longjmp`).

`abs`

This is identical to Perl's builtin `abs()` function, returning the absolute value of its numerical argument.

`access`

Determines the accessibility of a file.

```
if( POSIX::access( "/", &POSIX::R_OK ) ){
 print "have read permission\n";
}
```

Returns `undef` on failure. Note: do not use `access()` for security purposes. Between the `access()` call and the operation you are preparing for the permissions might change: a classic *race condition*.

`acos`

This is identical to the C function `acos()`, returning the arcus cosine of its numerical argument. See also *Math::Trig*.

`alarm`

This is identical to Perl's builtin `alarm()` function, either for arming or disarming the `SIGARLM` timer.

`asctime`

This is identical to the C function `asctime()`. It returns a string of the form

```
"Fri Jun  2 18:22:13 2000\n\0"
```

and it is called thusly

```
$asctime = asctime($sec, $min, $hour, $mday, $mon,
     $year, $wday, $yday, $isdst);
```

The `$mon` is zero-based: January equals `0`. The `$year` is 1900-based: 2001 equals `101`. `$wday` and `$yday` default to zero (and are usually ignored anyway), and `$isdst` defaults to -1.

`asin`

This is identical to the C function `asin()`, returning the arcus sine of its numerical argument. See also *Math::Trig*.

`assert`

Unimplemented, but you can use *"die" in perlfunc* and the *Carp* module to achieve similar things.

`atan`

This is identical to the C function `atan()`, returning the arcus tangent of its numerical argument. See also *Math::Trig*.

`atan2`

This is identical to Perl's builtin `atan2()` function, returning the arcus tangent defined by its two numerical arguments, the *y* coordinate and the *x* coordinate. See also *Math::Trig*.

`atexit`

`atexit()` is C-specific: use `END {}` instead, see *perlmod*.

`atof`

`atof()` is C-specific. Perl converts strings to numbers transparently. If you need to force a scalar to a number, add a zero to it.

`atoi`

`atoi()` is C-specific. Perl converts strings to numbers transparently. If you need to force a scalar to a number, add a zero to it. If you need to have just the integer part, see *"int" in perlfunc*.

`atol`

`atol()` is C-specific. Perl converts strings to numbers transparently. If you need to force a scalar to a number, add a zero to it. If you need to have just the integer part, see *"int" in perlfunc*.

`bsearch`

`bsearch()` not supplied. For doing binary search on wordlists, see *Search::Dict*.

`calloc`

`calloc()` is C-specific. Perl does memory management transparently.

`ceil`

This is identical to the C function `ceil()`, returning the smallest integer value greater than or equal to the given numerical argument.

`chdir`

This is identical to Perl's builtin `chdir()` function, allowing one to change the working (default) directory, see *"chdir" in perlfunc*.

`chmod`

This is identical to Perl's builtin `chmod()` function, allowing one to change file and directory permissions, see *"chmod" in perlfunc*.

`chown`

This is identical to Perl's builtin `chown()` function, allowing one to change file and directory owners and groups, see *"chown" in perlfunc*.

`clearerr`

Use the method `IO::Handle::clearerr()` instead, to reset the error state (if any) and EOF state (if any) of the given stream.

`clock`

This is identical to the C function `clock()`, returning the amount of spent processor time in microseconds.

`close`

Close the file. This uses file descriptors such as those obtained by calling `POSIX::open`.

```
$fd = POSIX::open( "foo", &POSIX::O_RDONLY );
POSIX::close( $fd );
```

Returns undef on failure.

See also *"close" in perlfunc*.

closedir

This is identical to Perl's builtin closedir() function for closing a directory handle, see *"closedir" in perlfunc*.

cos

This is identical to Perl's builtin cos() function, for returning the cosine of its numerical argument, see *"cos" in perlfunc*. See also *Math::Trig*.

cosh

This is identical to the C function cosh(), for returning the hyperbolic cosine of its numeric argument. See also *Math::Trig*.

creat

Create a new file. This returns a file descriptor like the ones returned by POSIX::open. Use POSIX::close to close the file.

```
$fd = POSIX::creat( "foo", 0611 );
POSIX::close( $fd );
```

See also *"sysopen" in perlfunc* and its O_CREAT flag.

ctermid

Generates the path name for the controlling terminal.

```
$path = POSIX::ctermid();
```

ctime

This is identical to the C function ctime() and equivalent to asctime(localtime(...)), see *asctime* and *localtime*.

cuserid

Get the login name of the owner of the current process.

```
$name = POSIX::cuserid();
```

difftime

This is identical to the C function difftime(), for returning the time difference (in seconds) between two times (as returned by time()), see *time*.

div

div() is C-specific, use *"int" in perlfunc* on the usual / division and the modulus %.

dup

This is similar to the C function dup(), for duplicating a file descriptor.

This uses file descriptors such as those obtained by calling POSIX::open.

Returns undef on failure.

dup2

This is similar to the C function dup2(), for duplicating a file descriptor to an another known file descriptor.

This uses file descriptors such as those obtained by calling POSIX::open.

Returns undef on failure.

`errno`

Returns the value of errno.

```
$errno = POSIX::errno();
```

This identical to the numerical values of the `$!`, see *"$ERRNO" in perlvar*.

`execl`

`execl()` is C-specific, see *"exec" in perlfunc*.

`execle`

`execle()` is C-specific, see *"exec" in perlfunc*.

`execlp`

`execlp()` is C-specific, see *"exec" in perlfunc*.

`execv`

`execv()` is C-specific, see *"exec" in perlfunc*.

`execve`

`execve()` is C-specific, see *"exec" in perlfunc*.

`execvp`

`execvp()` is C-specific, see *"exec" in perlfunc*.

`exit`

This is identical to Perl's builtin `exit()` function for exiting the program, see *"exit" in perlfunc*.

`exp`

This is identical to Perl's builtin `exp()` function for returning the exponent (*e*-based) of the numerical argument, see *"exp" in perlfunc*.

`fabs`

This is identical to Perl's builtin `abs()` function for returning the absolute value of the numerical argument, see *"abs" in perlfunc*.

`fclose`

Use method `IO::Handle::close()` instead, or see *"close" in perlfunc*.

`fcntl`

This is identical to Perl's builtin `fcntl()` function, see *"fcntl" in perlfunc*.

`fdopen`

Use method `IO::Handle::new_from_fd()` instead, or see *"open" in perlfunc*.

`feof`

Use method `IO::Handle::eof()` instead, or see *"eof" in perlfunc*.

`ferror`

Use method `IO::Handle::error()` instead.

`fflush`

Use method `IO::Handle::flush()` instead. See also *`"$OUTPUT_AUTOFLUSH" in perlvar`*.

`fgetc`

Use method `IO::Handle::getc()` instead, or see *"read" in perlfunc*.

`fgetpos`

Use method `IO::Seekable::getpos()` instead, or see *"seek" in perlfunc*.

`fgets`

Use method `IO::Handle::gets()` instead. Similar to <>, also known as *"readline" in perlfunc*.

`fileno`

Use method `IO::Handle::fileno()` instead, or see *"fileno" in perlfunc*.

`floor`

This is identical to the C function `floor()`, returning the largest integer value less than or equal to the numerical argument.

`fmod`

This is identical to the C function `fmod()`.

```
$r = fmod($x, $y);
```

It returns the remainder `$r = $x - $n*$y`, where `$n = trunc($x/$y)`. The `$r` has the same sign as `$x` and magnitude (absolute value) less than the magnitude of `$y`.

`fopen`

Use method `IO::File::open()` instead, or see *"open" in perlfunc*.

`fork`

This is identical to Perl's builtin `fork()` function for duplicating the current process, see *"fork" in perlfunc* and *perlfork* if you are in Windows.

`fpathconf`

Retrieves the value of a configurable limit on a file or directory. This uses file descriptors such as those obtained by calling `POSIX::open`.

The following will determine the maximum length of the longest allowable pathname on the filesystem which holds */var/foo*.

```
$fd = POSIX::open( "/var/foo", &POSIX::O_RDONLY );
$path_max = POSIX::fpathconf($fd, &POSIX::_PC_PATH_MAX);
```

Returns `undef` on failure.

`fprintf`

`fprintf()` is C-specific, see *"printf" in perlfunc* instead.

`fputc`

`fputc()` is C-specific, see *"print" in perlfunc* instead.

`fputs`

`fputs()` is C-specific, see *"print" in perlfunc* instead.

`fread`

`fread()` is C-specific, see *"read" in perlfunc* instead.

`free`

`free()` is C-specific. Perl does memory management transparently.

`freopen`

`freopen()` is C-specific, see *"open" in perlfunc* instead.

`frexp`

Return the mantissa and exponent of a floating-point number.

```
($mantissa, $exponent) = POSIX::frexp( 1.234e56 );
```

`fscanf`

`fscanf()` is C-specific, use <> and regular expressions instead.

`fseek`

Use method `IO::Seekable::seek()` instead, or see *"seek" in perlfunc*.

`fsetpos`

Use method `IO::Seekable::setpos()` instead, or seek *"seek" in perlfunc*.

`fstat`

Get file status. This uses file descriptors such as those obtained by calling `POSIX::open`. The data returned is identical to the data from Perl's builtin `stat` function.

```
$fd = POSIX::open( "foo", &POSIX::O_RDONLY );
@stats = POSIX::fstat( $fd );
```

`fsync`

Use method `IO::Handle::sync()` instead.

`ftell`

Use method `IO::Seekable::tell()` instead, or see *"tell" in perlfunc*.

`fwrite`

`fwrite()` is C-specific, see *"print" in perlfunc* instead.

`getc`

This is identical to Perl's builtin `getc()` function, see *"getc" in perlfunc*.

`getchar`

Returns one character from STDIN. Identical to Perl's `getc()`, see *"getc" in perlfunc*.

`getcwd`

Returns the name of the current working directory. See also *Cwd*.

`getegid`

Returns the effective group identifier. Similar to Perl' s builtin variable `$(`, see *"$EGID" in perlvar*.

`getenv`

Returns the value of the specified environment variable. The same information is available through the `%ENV` array.

`geteuid`

Returns the effective user identifier. Identical to Perl's builtin $> variable, see *"$EUID" in perlvar*.

`getgid`

Returns the user's real group identifier. Similar to Perl's builtin variable `$)`, see *"$GID" in perlvar*.

`getgrgid`

This is identical to Perl's builtin `getgrgid()` function for returning group entries by group identifiers, see *"getgrgid" in perlfunc*.

`getgrnam`

This is identical to Perl's builtin `getgrnam()` function for returning group entries by group names, see *"getgrnam" in perlfunc*.

`getgroups`

Returns the ids of the user's supplementary groups. Similar to Perl's builtin variable `$)` , see *"$GID" in perlvar*.

`getlogin`

This is identical to Perl's builtin `getlogin()` function for returning the user name associated with the current session, see *"getlogin" in perlfunc*.

`getpgrp`

This is identical to Perl's builtin `getpgrp()` function for returning the process group identifier of the current process, see *"getpgrp" in perlfunc*.

`getpid`

Returns the process identifier. Identical to Perl's builtin variable $$, see *"$PID" in perlvar*.

`getppid`

This is identical to Perl's builtin `getppid()` function for returning the process identifier of the parent process of the current process , see *"getppid" in perlfunc*.

`getpwnam`

This is identical to Perl's builtin `getpwnam()` function for returning user entries by user names, see *"getpwnam" in perlfunc*.

`getpwuid`

This is identical to Perl's builtin `getpwuid()` function for returning user entries by user identifiers, see *"getpwuid" in perlfunc*.

`gets`

Returns one line from `STDIN`, similar to <>, also known as the `readline()` function, see *"readline" in perlfunc*.

NOTE: if you have C programs that still use `gets()`, be very afraid. The `gets()` function is a source of endless grief because it has no buffer overrun checks. It should **never** be used. The `fgets()` function should be preferred instead.

`getuid`

Returns the user's identifier. Identical to Perl's builtin $< variable, see *"$UID" in perlvar* .

`gmtime`

This is identical to Perl's builtin `gmtime()` function for converting seconds since the epoch to a date in Greenwich Mean Time, see *"gmtime" in perlfunc*.

`isalnum`

Deprecated function whose use raises a warning, and which is slated to be removed in

a future Perl version. It is very similar to matching against `qr/ ^ [[:alnum:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

You may want to use the `/\w/` construct instead.

`isalpha`

Deprecated function whose use raises a warning, and which is slated to be removed in a future Perl version. It is very similar to matching against `qr/ ^ [[:alpha:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

`isatty`

Returns a boolean indicating whether the specified filehandle is connected to a tty. Similar to the `-t` operator, see *"-X" in perlfunc*.

`iscntrl`

Deprecated function whose use raises a warning, and which is slated to be removed in a future Perl version. It is very similar to matching against `qr/ ^ [[:cntrl:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

`isdigit`

Deprecated function whose use raises a warning, and which is slated to be removed in a future Perl version. It is very similar to matching against `qr/ ^ [[:digit:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

You may want to use the `/\d/` construct instead.

`isgraph`

Deprecated function whose use raises a warning, and which is slated to be removed in

a future Perl version. It is very similar to matching against `qr/ ^ [[:graph:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

`islower`

Deprecated function whose use raises a warning, and which is slated to be removed in a future Perl version. It is very similar to matching against `qr/ ^ [[:lower:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

Do **not** use `/[a-z]/` unless you don't care about the current locale.

`isprint`

Deprecated function whose use raises a warning, and which is slated to be removed in a future Perl version. It is very similar to matching against `qr/ ^ [[:print:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

`ispunct`

Deprecated function whose use raises a warning, and which is slated to be removed in a future Perl version. It is very similar to matching against `qr/ ^ [[:punct:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

`isspace`

Deprecated function whose use raises a warning, and which is slated to be removed in a future Perl version. It is very similar to matching against `qr/ ^ [[:space:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect

and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

You may want to use the `/\s/` construct instead.

`isupper`

Deprecated function whose use raises a warning, and which is slated to be removed in a future Perl version. It is very similar to matching against `qr/ ^ [[:upper:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

Do **not** use `/[A-Z]/` unless you don't care about the current locale.

`isxdigit`

Deprecated function whose use raises a warning, and which is slated to be removed in a future Perl version. It is very similar to matching against `qr/ ^ [[:xdigit:]]+ $ /x`, which you should convert to use instead. The function is deprecated because 1) it doesn't handle UTF-8 encoded strings properly; and 2) it returns TRUE even if the input is the empty string. The function return is always based on the current locale, whereas using locale rules is optional with the regular expression, based on pragmas in effect and pattern modifiers (see *"Character set modifiers" in perlre* and *"Which character set modifier is in effect?" in perlre*).

The function returns TRUE if the input string is empty, or if the corresponding C function returns TRUE for every byte in the string.

`kill`

This is identical to Perl's builtin `kill()` function for sending signals to processes (often to terminate them), see *"kill" in perlfunc*.

`labs`

(For returning absolute values of long integers.) `labs()` is C-specific, see *"abs" in perlfunc* instead.

`lchown`

This is identical to the C function, except the order of arguments is consistent with Perl's builtin `chown()` with the added restriction of only one path, not an list of paths. Does the same thing as the `chown()` function but changes the owner of a symbolic link instead of the file the symbolic link points to.

`ldexp`

This is identical to the C function `ldexp()` for multiplying floating point numbers with powers of two.

```
$x_quadrupled = POSIX::ldexp($x, 2);
```

`ldiv`

(For computing dividends of long integers.) `ldiv()` is C-specific, use `/` and `int()` instead.

`link`

This is identical to Perl's builtin `link()` function for creating hard links into files, see *"link" in perlfunc*.

`localeconv`

Get numeric formatting information. Returns a reference to a hash containing the current locale formatting values. Users of this function should also read *perllocale*, which provides a comprehensive discussion of Perl locale handling, including *a section devoted to this function*.

Here is how to query the database for the **de** (Deutsch or German) locale.

```
my $loc = POSIX::setlocale( &POSIX::LC_ALL, "de" );
print "Locale: \"$loc\"\n";
my $lconv = POSIX::localeconv();
foreach my $property (qw(
 decimal_point
 thousands_sep
 grouping
 int_curr_symbol
 currency_symbol
 mon_decimal_point
 mon_thousands_sep
 mon_grouping
 positive_sign
 negative_sign
 int_frac_digits
 frac_digits
 p_cs_precedes
 p_sep_by_space
 n_cs_precedes
 n_sep_by_space
 p_sign_posn
 n_sign_posn
))
{
 printf qq(%s: "%s",\n),
  $property, $lconv->{$property};
}
```

`localtime`

This is identical to Perl's builtin `localtime()` function for converting seconds since the epoch to a date see *"localtime" in perlfunc*.

`log`

This is identical to Perl's builtin `log()` function, returning the natural (*e*-based) logarithm of the numerical argument, see *"log" in perlfunc*.

`log10`

This is identical to the C function `log10()`, returning the 10-base logarithm of the numerical argument. You can also use

```
    sub log10 { log($_[0]) / log(10) }
```

or

```
    sub log10 { log($_[0]) / 2.30258509299405 }
```

or

```
sub log10 { log($_[0]) * 0.434294481903252 }
```

`longjmp`

`longjmp()` is C-specific: use *"die" in perlfunc* instead.

`lseek`

Move the file's read/write position. This uses file descriptors such as those obtained by calling `POSIX::open`.

```
$fd = POSIX::open( "foo", &POSIX::O_RDONLY );
$off_t = POSIX::lseek( $fd, 0, &POSIX::SEEK_SET );
```

Returns `undef` on failure.

`malloc`

`malloc()` is C-specific. Perl does memory management transparently.

`mblen`

This is identical to the C function `mblen()`. Perl does not have any support for the wide and multibyte characters of the C standards, so this might be a rather useless function.

`mbstowcs`

This is identical to the C function `mbstowcs()`. Perl does not have any support for the wide and multibyte characters of the C standards, so this might be a rather useless function.

`mbtowc`

This is identical to the C function `mbtowc()`. Perl does not have any support for the wide and multibyte characters of the C standards, so this might be a rather useless function.

`memchr`

`memchr()` is C-specific, see *"index" in perlfunc* instead.

`memcmp`

`memcmp()` is C-specific, use `eq` instead, see *perlop*.

`memcpy`

`memcpy()` is C-specific, use =, see *perlop*, or see *"substr" in perlfunc*.

`memmove`

`memmove()` is C-specific, use =, see *perlop*, or see *"substr" in perlfunc*.

`memset`

`memset()` is C-specific, use `x` instead, see *perlop*.

`mkdir`

This is identical to Perl's builtin `mkdir()` function for creating directories, see *"mkdir" in perlfunc*.

`mkfifo`

This is similar to the C function `mkfifo()` for creating FIFO special files.

```
if (mkfifo($path, $mode)) { ....
```

Returns undef on failure. The $mode is similar to the mode of mkdir(), see *"mkdir" in perlfunc*, though for mkfifo you **must** specify the $mode.

mktime

Convert date/time info to a calendar time.

Synopsis:

```
mktime(sec, min, hour, mday, mon, year, wday = 0,
       yday = 0, isdst = -1)
```

The month (mon), weekday (wday), and yearday (yday) begin at zero. I.e. January is 0, not 1; Sunday is 0, not 1; January 1st is 0, not 1. The year (year) is given in years since 1900. I.e. The year 1995 is 95; the year 2001 is 101. Consult your system's mktime() manpage for details about these and the other arguments.

Calendar time for December 12, 1995, at 10:30 am.

```
$time_t = POSIX::mktime( 0, 30, 10, 12, 11, 95 );
print "Date = ", POSIX::ctime($time_t);
```

Returns undef on failure.

modf

Return the integral and fractional parts of a floating-point number.

```
($fractional, $integral) = POSIX::modf( 3.14 );
```

nice

This is similar to the C function nice(), for changing the scheduling preference of the current process. Positive arguments mean more polite process, negative values more needy process. Normal user processes can only be more polite.

Returns undef on failure.

offsetof

offsetof() is C-specific, you probably want to see *"pack" in perlfunc* instead.

open

Open a file for reading for writing. This returns file descriptors, not Perl filehandles. Use POSIX::close to close the file.

Open a file read-only with mode 0666.

```
$fd = POSIX::open( "foo" );
```

Open a file for read and write.

```
$fd = POSIX::open( "foo", &POSIX::O_RDWR );
```

Open a file for write, with truncation.

```
$fd = POSIX::open(
 "foo", &POSIX::O_WRONLY | &POSIX::O_TRUNC
);
```

Create a new file with mode 0640. Set up the file for writing.

```
$fd = POSIX::open(
 "foo", &POSIX::O_CREAT | &POSIX::O_WRONLY, 0640
);
```

Returns undef on failure.

See also *"sysopen" in perlfunc*.

`opendir`

Open a directory for reading.

```
$dir = POSIX::opendir( "/var" );
@files = POSIX::readdir( $dir );
POSIX::closedir( $dir );
```

Returns `undef` on failure.

`pathconf`

Retrieves the value of a configurable limit on a file or directory.

The following will determine the maximum length of the longest allowable pathname on the filesystem which holds `/var`.

```
$path_max = POSIX::pathconf( "/var",
          &POSIX::_PC_PATH_MAX );
```

Returns `undef` on failure.

`pause`

This is similar to the C function `pause()`, which suspends the execution of the current process until a signal is received.

Returns `undef` on failure.

`perror`

This is identical to the C function `perror()`, which outputs to the standard error stream the specified message followed by `": "` and the current error string. Use the `warn()` function and the `$!` variable instead, see *"warn" in perlfunc* and *"$ERRNO" in perlvar*.

`pipe`

Create an interprocess channel. This returns file descriptors like those returned by `POSIX::open`.

```
my ($read, $write) = POSIX::pipe();
POSIX::write( $write, "hello", 5 );
POSIX::read( $read, $buf, 5 );
```

See also *"pipe" in perlfunc*.

`pow`

Computes `$x` raised to the power `$exponent`.

```
$ret = POSIX::pow( $x, $exponent );
```

You can also use the `**` operator, see *perlop*.

`printf`

Formats and prints the specified arguments to STDOUT. See also *"printf" in perlfunc*.

`putc`

`putc()` is C-specific, see *"print" in perlfunc* instead.

`putchar`

`putchar()` is C-specific, see *"print" in perlfunc* instead.

`puts`

`puts()` is C-specific, see *"print" in perlfunc* instead.

`qsort`

`qsort()` is C-specific, see *"sort" in perlfunc* instead.

`raise`

Sends the specified signal to the current process. See also *"kill" in perlfunc* and the `$$` in *"$PID" in perlvar*.

`rand`

`rand()` is non-portable, see *"rand" in perlfunc* instead.

`read`

Read from a file. This uses file descriptors such as those obtained by calling `POSIX::open`. If the buffer `$buf` is not large enough for the read then Perl will extend it to make room for the request.

```
$fd = POSIX::open( "foo", &POSIX::O_RDONLY );
$bytes = POSIX::read( $fd, $buf, 3 );
```

Returns `undef` on failure.

See also *"sysread" in perlfunc*.

`readdir`

This is identical to Perl's builtin `readdir()` function for reading directory entries, see *"readdir" in perlfunc*.

`realloc`

`realloc()` is C-specific. Perl does memory management transparently.

`remove`

This is identical to Perl's builtin `unlink()` function for removing files, see *"unlink" in perlfunc*.

`rename`

This is identical to Perl's builtin `rename()` function for renaming files, see *"rename" in perlfunc*.

`rewind`

Seeks to the beginning of the file.

`rewinddir`

This is identical to Perl's builtin `rewinddir()` function for rewinding directory entry streams, see *"rewinddir" in perlfunc*.

`rmdir`

This is identical to Perl's builtin `rmdir()` function for removing (empty) directories, see *"rmdir" in perlfunc*.

`scanf`

`scanf()` is C-specific, use <> and regular expressions instead, see *perlre*.

`setgid`

Sets the real group identifier and the effective group identifier for this process. Similar to assigning a value to the Perl's builtin `$)` variable, see *"$EGID" in perlvar*, except

that the latter will change only the real user identifier, and that the setgid() uses only a single numeric argument, as opposed to a space-separated list of numbers.

`setjmp`

`setjmp()` is C-specific: use `eval {}` instead, see *"eval" in perlfunc*.

`setlocale`

Modifies and queries the program's underlying locale. Users of this function should read *perllocale*, whch provides a comprehensive discussion of Perl locale handling, knowledge of which is necessary to properly use this function. It contains *a section devoted to this function*. The discussion here is merely a summary reference for `setlocale()`. Note that Perl itself is almost entirely unaffected by the locale except within the scope of `"use locale"`. (Exceptions are listed in *"Not within the scope of any "use locale" variant" in perllocale*.)

The following examples assume

```
use POSIX qw(setlocale LC_ALL LC_CTYPE);
```

has been issued.

The following will set the traditional UNIX system locale behavior (the second argument `"C"`).

```
$loc = setlocale( LC_ALL, "C" );
```

The following will query the current `LC_CTYPE` category. (No second argument means 'query'.)

```
$loc = setlocale( LC_CTYPE );
```

The following will set the `LC_CTYPE` behaviour according to the locale environment variables (the second argument `""`). Please see your system's `setlocale(3)` documentation for the locale environment variables' meaning or consult *perllocale*.

```
$loc = setlocale( LC_CTYPE, "" );
```

The following will set the `LC_COLLATE` behaviour to Argentinian Spanish. **NOTE**: The naming and availability of locales depends on your operating system. Please consult *perllocale* for how to find out which locales are available in your system.

```
$loc = setlocale( LC_COLLATE, "es_AR.ISO8859-1" );
```

`setpgid`

This is similar to the C function `setpgid()` for setting the process group identifier of the current process.

Returns `undef` on failure.

`setsid`

This is identical to the C function `setsid()` for setting the session identifier of the current process.

`setuid`

Sets the real user identifier and the effective user identifier for this process. Similar to assigning a value to the Perl's builtin $< variable, see *"$UID" in perlvar*, except that the latter will change only the real user identifier.

`sigaction`

Detailed signal management. This uses `POSIX::SigAction` objects for the `action` and `oldaction` arguments (the oldaction can also be just a hash reference). Consult

your system's sigaction manpage for details, see also POSIX::SigRt.

Synopsis:

```
sigaction(signal, action, oldaction = 0)
```

Returns undef on failure. The signal must be a number (like SIGHUP), not a string (like "SIGHUP"), though Perl does try hard to understand you.

If you use the SA_SIGINFO flag, the signal handler will in addition to the first argument, the signal name, also receive a second argument, a hash reference, inside which are the following keys with the following semantics, as defined by POSIX/SUSv3:

```
signo       the signal number
errno       the error number
code        if this is zero or less, the signal was sent by
            a user process and the uid and pid make sense,
            otherwise the signal was sent by the kernel
```

The following are also defined by POSIX/SUSv3, but unfortunately not very widely implemented:

```
pid         the process id generating the signal
uid         the uid of the process id generating the signal
status      exit value or signal for SIGCHLD
band        band event for SIGPOLL
```

A third argument is also passed to the handler, which contains a copy of the raw binary contents of the siginfo structure: if a system has some non-POSIX fields, this third argument is where to unpack() them from.

Note that not all siginfo values make sense simultaneously (some are valid only for certain signals, for example), and not all values make sense from Perl perspective, you should to consult your system's sigaction and possibly also siginfo documentation.

siglongjmp

siglongjmp() is C-specific: use *"die" in perlfunc* instead.

sigpending

Examine signals that are blocked and pending. This uses POSIX::SigSet objects for the sigset argument. Consult your system's sigpending manpage for details.

Synopsis:

```
sigpending(sigset)
```

Returns undef on failure.

sigprocmask

Change and/or examine calling process's signal mask. This uses POSIX::SigSet objects for the sigset and oldsigset arguments. Consult your system's sigprocmask manpage for details.

Synopsis:

```
sigprocmask(how, sigset, oldsigset = 0)
```

Returns undef on failure.

Note that you can't reliably block or unblock a signal from its own signal handler if you're using safe signals. Other signals can be blocked or unblocked reliably.

`sigsetjmp`

`sigsetjmp()` is C-specific: use `eval {}` instead, see *"eval" in perlfunc*.

`sigsuspend`

Install a signal mask and suspend process until signal arrives. This uses `POSIX::SigSet` objects for the `signal_mask` argument. Consult your system's `sigsuspend` manpage for details.

Synopsis:

```
sigsuspend(signal_mask)
```

Returns `undef` on failure.

`sin`

This is identical to Perl's builtin `sin()` function for returning the sine of the numerical argument, see *"sin" in perlfunc*. See also *Math::Trig*.

`sinh`

This is identical to the C function `sinh()` for returning the hyperbolic sine of the numerical argument. See also *Math::Trig*.

`sleep`

This is functionally identical to Perl's builtin `sleep()` function for suspending the execution of the current for process for certain number of seconds, see *"sleep" in perlfunc*. There is one significant difference, however: `POSIX::sleep()` returns the number of **unslept** seconds, while the `CORE::sleep()` returns the number of slept seconds.

`sprintf`

This is similar to Perl's builtin `sprintf()` function for returning a string that has the arguments formatted as requested, see *"sprintf" in perlfunc*.

`sqrt`

This is identical to Perl's builtin `sqrt()` function. for returning the square root of the numerical argument, see *"sqrt" in perlfunc*.

`srand`

Give a seed the pseudorandom number generator, see *"srand" in perlfunc*.

`sscanf`

`sscanf()` is C-specific, use regular expressions instead, see *perlre*.

`stat`

This is identical to Perl's builtin `stat()` function for returning information about files and directories.

`strcat`

`strcat()` is C-specific, use `.=` instead, see *perlop*.

`strchr`

`strchr()` is C-specific, see *"index" in perlfunc* instead.

`strcmp`

`strcmp()` is C-specific, use `eq` or `cmp` instead, see *perlop*.

`strcoll`

This is identical to the C function `strcoll()` for collating (comparing) strings transformed using the `strxfrm()` function. Not really needed since Perl can do this transparently, see *perllocale*.

`strcpy`

`strcpy()` is C-specific, use = instead, see *perlop*.

`strcspn`

`strcspn()` is C-specific, use regular expressions instead, see *perlre*.

`strerror`

Returns the error string for the specified errno. Identical to the string form of the `$!`, see *"$ERRNO" in perlvar*.

`strftime`

Convert date and time information to string. Returns the string.

Synopsis:

```
strftime(fmt, sec, min, hour, mday, mon, year,
  wday = -1, yday = -1, isdst = -1)
```

The month (`mon`), weekday (`wday`), and yearday (`yday`) begin at zero. I.e. January is 0, not 1; Sunday is 0, not 1; January 1st is 0, not 1. The year (`year`) is given in years since 1900. I.e., the year 1995 is 95; the year 2001 is 101. Consult your system's `strftime()` manpage for details about these and the other arguments.

If you want your code to be portable, your format (`fmt`) argument should use only the conversion specifiers defined by the ANSI C standard (C89, to play safe). These are `aAbBcdHIjmMpSUwWxXyYZ%`. But even then, the **results** of some of the conversion specifiers are non-portable. For example, the specifiers `aAbBcpZ` change according to the locale settings of the user, and both how to set locales (the locale names) and what output to expect are non-standard. The specifier `c` changes according to the timezone settings of the user and the timezone computation rules of the operating system. The `Z` specifier is notoriously unportable since the names of timezones are non-standard. Sticking to the numeric specifiers is the safest route.

The given arguments are made consistent as though by calling `mktime()` before calling your system's `strftime()` function, except that the `isdst` value is not affected.

The string for Tuesday, December 12, 1995.

```
$str = POSIX::strftime( "%A, %B %d, %Y",
    0, 0, 0, 12, 11, 95, 2 );
print "$str\n";
```

`strlen`

`strlen()` is C-specific, use `length()` instead, see *"length" in perlfunc*.

`strncat`

`strncat()` is C-specific, use `.=` instead, see *perlop*.

`strncmp`

`strncmp()` is C-specific, use `eq` instead, see *perlop*.

`strncpy`

`strncpy()` is C-specific, use = instead, see *perlop*.

`strpbrk`

`strpbrk()` is C-specific, use regular expressions instead, see *perlre*.

`strrchr`

`strrchr()` is C-specific, see *"rindex" in perlfunc* instead.

`strspn`

`strspn()` is C-specific, use regular expressions instead, see *perlre*.

`strstr`

This is identical to Perl's builtin `index()` function, see *"index" in perlfunc*.

`strtod`

String to double translation. Returns the parsed number and the number of characters in the unparsed portion of the string. Truly POSIX-compliant systems set `$!` (`$ERRNO`) to indicate a translation error, so clear `$!` before calling strtod. However, non-POSIX systems may not check for overflow, and therefore will never set `$!`.

strtod respects any POSIX *setlocale()* `LC_TIME` settings, regardless of whether or not it is called from Perl code that is within the scope of `use locale`.

To parse a string `$str` as a floating point number use

```
$! = 0;
($num, $n_unparsed) = POSIX::strtod($str);
```

The second returned item and `$!` can be used to check for valid input:

```
if (($str eq '') || ($n_unparsed != 0) || $!) {
    die "Non-numeric input $str" . ($! ? ": $!\n" : "\n");
}
```

When called in a scalar context strtod returns the parsed number.

`strtok`

`strtok()` is C-specific, use regular expressions instead, see *perlre*, or *"split" in perlfunc*.

`strtol`

String to (long) integer translation. Returns the parsed number and the number of characters in the unparsed portion of the string. Truly POSIX-compliant systems set `$!` (`$ERRNO`) to indicate a translation error, so clear `$!` before calling `strtol`. However, non-POSIX systems may not check for overflow, and therefore will never set `$!`.

`strtol` should respect any POSIX *setlocale()* settings.

To parse a string `$str` as a number in some base `$base` use

```
$! = 0;
($num, $n_unparsed) = POSIX::strtol($str, $base);
```

The base should be zero or between 2 and 36, inclusive. When the base is zero or omitted strtol will use the string itself to determine the base: a leading "0x" or "0X" means hexadecimal; a leading "0" means octal; any other leading characters mean decimal. Thus, "1234" is parsed as a decimal number, "01234" as an octal number, and "0x1234" as a hexadecimal number.

The second returned item and `$!` can be used to check for valid input:

```
if (($str eq '') || ($n_unparsed != 0) || !$!) {
    die "Non-numeric input $str" . $! ? ": $!\n" : "\n";
}
```

When called in a scalar context strtol returns the parsed number.

`strtoul`

String to unsigned (long) integer translation. `strtoul()` is identical to `strtol()` except that `strtoul()` only parses unsigned integers. See *strtol* for details.

Note: Some vendors supply `strtod()` and `strtol()` but not `strtoul()`. Other vendors that do supply `strtoul()` parse "-1" as a valid value.

`strxfrm`

String transformation. Returns the transformed string.

```
$dst = POSIX::strxfrm( $src );
```

Used in conjunction with the `strcoll()` function, see *strcoll*.

Not really needed since Perl can do this transparently, see *perllocale*.

`sysconf`

Retrieves values of system configurable variables.

The following will get the machine's clock speed.

```
$clock_ticks = POSIX::sysconf( &POSIX::_SC_CLK_TCK );
```

Returns `undef` on failure.

`system`

This is identical to Perl's builtin `system()` function, see *"system" in perlfunc*.

`tan`

This is identical to the C function `tan()`, returning the tangent of the numerical argument. See also *Math::Trig*.

`tanh`

This is identical to the C function `tanh()`, returning the hyperbolic tangent of the numerical argument. See also *Math::Trig*.

`tcdrain`

This is similar to the C function `tcdrain()` for draining the output queue of its argument stream.

Returns `undef` on failure.

`tcflow`

This is similar to the C function `tcflow()` for controlling the flow of its argument stream.

Returns `undef` on failure.

`tcflush`

This is similar to the C function `tcflush()` for flushing the I/O buffers of its argument stream.

Returns `undef` on failure.

`tcgetpgrp`

This is identical to the C function `tcgetpgrp()` for returning the process group identifier of the foreground process group of the controlling terminal.

`tcsendbreak`

This is similar to the C function `tcsendbreak()` for sending a break on its argument stream.

Returns `undef` on failure.

`tcsetpgrp`

This is similar to the C function `tcsetpgrp()` for setting the process group identifier of the foreground process group of the controlling terminal.

Returns `undef` on failure.

`time`

This is identical to Perl's builtin `time()` function for returning the number of seconds since the epoch (whatever it is for the system), see *"time" in perlfunc*.

`times`

The `times()` function returns elapsed realtime since some point in the past (such as system startup), user and system times for this process, and user and system times used by child processes. All times are returned in clock ticks.

```
    ($realtime, $user, $system, $cuser, $csystem)
 = POSIX::times();
```

Note: Perl's builtin `times()` function returns four values, measured in seconds.

`tmpfile`

Use method `IO::File::new_tmpfile()` instead, or see *File::Temp*.

`tmpnam`

Returns a name for a temporary file.

```
 $tmpfile = POSIX::tmpnam();
```

For security reasons, which are probably detailed in your system's documentation for the C library `tmpnam()` function, this interface should not be used; instead see *File::Temp*.

`tolower`

This is identical to the C function, except that it can apply to a single character or to a whole string. Consider using the `lc()` function, see *"lc" in perlfunc*, or the equivalent `\L` operator inside doublequotish strings.

`toupper`

This is identical to the C function, except that it can apply to a single character or to a whole string. Consider using the `uc()` function, see *"uc" in perlfunc*, or the equivalent `\U` operator inside doublequotish strings.

`ttyname`

This is identical to the C function `ttyname()` for returning the name of the current terminal.

`tzname`

Retrieves the time conversion information from the `tzname` variable.

```
 POSIX::tzset();
 ($std, $dst) = POSIX::tzname();
```

`tzset`

This is identical to the C function `tzset()` for setting the current timezone based on

the environment variable TZ, to be used by `ctime()`, `localtime()`, `mktime()`, and `strftime()` functions.

`umask`

This is identical to Perl's builtin `umask()` function for setting (and querying) the file creation permission mask, see *"umask" in perlfunc*.

`uname`

Get name of current operating system.

```
($sysname, $nodename, $release, $version, $machine)
 = POSIX::uname();
```

Note that the actual meanings of the various fields are not that well standardized, do not expect any great portability. The `$sysname` might be the name of the operating system, the `$nodename` might be the name of the host, the `$release` might be the (major) release number of the operating system, the `$version` might be the (minor) release number of the operating system, and the `$machine` might be a hardware identifier. Maybe.

`ungetc`

Use method `IO::Handle::ungetc()` instead.

`unlink`

This is identical to Perl's builtin `unlink()` function for removing files, see *"unlink" in perlfunc*.

`utime`

This is identical to Perl's builtin `utime()` function for changing the time stamps of files and directories, see *"utime" in perlfunc*.

`vfprintf`

`vfprintf()` is C-specific, see *"printf" in perlfunc* instead.

`vprintf`

`vprintf()` is C-specific, see *"printf" in perlfunc* instead.

`vsprintf`

`vsprintf()` is C-specific, see *"sprintf" in perlfunc* instead.

`wait`

This is identical to Perl's builtin `wait()` function, see *"wait" in perlfunc*.

`waitpid`

Wait for a child process to change state. This is identical to Perl's builtin `waitpid()` function, see *"waitpid" in perlfunc*.

```
$pid = POSIX::waitpid( -1, POSIX::WNOHANG );
print "status = ", ($? / 256), "\n";
```

`wcstombs`

This is identical to the C function `wcstombs()`. Perl does not have any support for the wide and multibyte characters of the C standards, so this might be a rather useless function.

`wctomb`

This is identical to the C function `wctomb()`. Perl does not have any support for the

wide and multibyte characters of the C standards, so this might be a rather useless function.

`write`

Write to a file. This uses file descriptors such as those obtained by calling `POSIX::open`.

```
$fd = POSIX::open( "foo", &POSIX::O_WRONLY );
$buf = "hello";
$bytes = POSIX::write( $fd, $buf, 5 );
```

Returns `undef` on failure.

See also *"syswrite" in perlfunc*.

CLASSES

POSIX::SigAction

`new`

Creates a new `POSIX::SigAction` object which corresponds to the C `struct sigaction`. This object will be destroyed automatically when it is no longer needed. The first parameter is the handler, a sub reference. The second parameter is a `POSIX::SigSet` object, it defaults to the empty set. The third parameter contains the `sa_flags`, it defaults to 0.

```
$sigset = POSIX::SigSet->new(SIGINT, SIGQUIT);
$sigaction = POSIX::SigAction->new(
  \&handler, $sigset, &POSIX::SA_NOCLDSTOP
      );
```

This `POSIX::SigAction` object is intended for use with the `POSIX::sigaction()` function.

`handler`

`mask`

`flags`

accessor functions to get/set the values of a SigAction object.

```
$sigset = $sigaction->mask;
$sigaction->flags(&POSIX::SA_RESTART);
```

`safe`

accessor function for the "safe signals" flag of a SigAction object; see *perlipc* for general information on safe (a.k.a. "deferred") signals. If you wish to handle a signal safely, use this accessor to set the "safe" flag in the `POSIX::SigAction` object:

```
$sigaction->safe(1);
```

You may also examine the "safe" flag on the output action object which is filled in when given as the third parameter to `POSIX::sigaction()`:

```
sigaction(SIGINT, $new_action, $old_action);
if ($old_action->safe) {
    # previous SIGINT handler used safe signals
}
```

POSIX::SigRt

`%SIGRT`

A hash of the POSIX realtime signal handlers. It is an extension of the standard `%SIG`, the `$POSIX::SIGRT{SIGRTMIN}` is roughly equivalent to `$SIG{SIGRTMIN}`, but the right POSIX moves (see below) are made with the `POSIX::SigSet` and `POSIX::sigaction` instead of accessing the `%SIG`.

You can set the `%POSIX::SIGRT` elements to set the POSIX realtime signal handlers, use `delete` and `exists` on the elements, and use `scalar` on the `%POSIX::SIGRT` to find out how many POSIX realtime signals there are available (`SIGRTMAX - SIGRTMIN + 1`, the `SIGRTMAX` is a valid POSIX realtime signal).

Setting the `%SIGRT` elements is equivalent to calling this:

```
sub new {
  my ($rtsig, $handler, $flags) = @_;
  my $sigset = POSIX::SigSet($rtsig);
  my $sigact = POSIX::SigAction->new($handler,$sigset,$flags);
  sigaction($rtsig, $sigact);
}
```

The flags default to zero, if you want something different you can either use `local` on `$POSIX::SigRt::SIGACTION_FLAGS`, or you can derive from POSIX::SigRt and define your own `new()` (the tied hash STORE method of the `%SIGRT` calls `new($rtsig, $handler, $SIGACTION_FLAGS)`, where the `$rtsig` ranges from zero to `SIGRTMAX - SIGRTMIN + 1`).

Just as with any signal, you can use `sigaction($rtsig, undef, $oa)` to retrieve the installed signal handler (or, rather, the signal action).

NOTE: whether POSIX realtime signals really work in your system, or whether Perl has been compiled so that it works with them, is outside of this discussion.

`SIGRTMIN`

Return the minimum POSIX realtime signal number available, or `undef` if no POSIX realtime signals are available.

`SIGRTMAX`

Return the maximum POSIX realtime signal number available, or `undef` if no POSIX realtime signals are available.

POSIX::SigSet

`new`

Create a new SigSet object. This object will be destroyed automatically when it is no longer needed. Arguments may be supplied to initialize the set.

Create an empty set.

```
$sigset = POSIX::SigSet->new;
```

Create a set with `SIGUSR1`.

```
$sigset = POSIX::SigSet->new( &POSIX::SIGUSR1 );
```

`addset`

Add a signal to a SigSet object.

```
$sigset->addset( &POSIX::SIGUSR2 );
```

Returns `undef` on failure.

`delset`

Remove a signal from the SigSet object.

```
$sigset->delset( &POSIX::SIGUSR2 );
```

Returns `undef` on failure.

`emptyset`

Initialize the SigSet object to be empty.

```
$sigset->emptyset();
```

Returns `undef` on failure.

`fillset`

Initialize the SigSet object to include all signals.

```
$sigset->fillset();
```

Returns `undef` on failure.

`ismember`

Tests the SigSet object to see if it contains a specific signal.

```
if( $sigset->ismember( &POSIX::SIGUSR1 ) ){
 print "contains SIGUSR1\n";
}
```

POSIX::Termios

`new`

Create a new Termios object. This object will be destroyed automatically when it is no longer needed. A Termios object corresponds to the termios C struct. `new()` mallocs a new one, `getattr()` fills it from a file descriptor, and `setattr()` sets a file descriptor's parameters to match Termios' contents.

```
$termios = POSIX::Termios->new;
```

`getattr`

Get terminal control attributes.

Obtain the attributes for stdin.

```
$termios->getattr( 0 ) # Recommended for clarity.
$termios->getattr()
```

Obtain the attributes for stdout.

```
$termios->getattr( 1 )
```

Returns `undef` on failure.

`getcc`

Retrieve a value from the c_cc field of a termios object. The c_cc field is an array so an index must be specified.

```
$c_cc[1] = $termios->getcc(1);
```

`getcflag`

Retrieve the c_cflag field of a termios object.

```
    $c_cflag = $termios->getcflag;
```

getiflag

Retrieve the c_iflag field of a termios object.

```
    $c_iflag = $termios->getiflag;
```

getispeed

Retrieve the input baud rate.

```
    $ispeed = $termios->getispeed;
```

getlflag

Retrieve the c_lflag field of a termios object.

```
    $c_lflag = $termios->getlflag;
```

getoflag

Retrieve the c_oflag field of a termios object.

```
    $c_oflag = $termios->getoflag;
```

getospeed

Retrieve the output baud rate.

```
    $ospeed = $termios->getospeed;
```

setattr

Set terminal control attributes.

Set attributes immediately for stdout.

```
    $termios->setattr( 1, &POSIX::TCSANOW );
```

Returns `undef` on failure.

setcc

Set a value in the c_cc field of a termios object. The c_cc field is an array so an index must be specified.

```
    $termios->setcc( &POSIX::VEOF, 1 );
```

setcflag

Set the c_cflag field of a termios object.

```
    $termios->setcflag( $c_cflag | &POSIX::CLOCAL );
```

setiflag

Set the c_iflag field of a termios object.

```
    $termios->setiflag( $c_iflag | &POSIX::BRKINT );
```

setispeed

Set the input baud rate.

```
    $termios->setispeed( &POSIX::B9600 );
```

Returns undef on failure.

setlflag

Set the c_lflag field of a termios object.

```
$termios->setlflag( $c_lflag | &POSIX::ECHO );
```

setoflag

Set the c_oflag field of a termios object.

```
$termios->setoflag( $c_oflag | &POSIX::OPOST );
```

setospeed

Set the output baud rate.

```
$termios->setospeed( &POSIX::B9600 );
```

Returns undef on failure.

Baud rate values

B38400 B75 B200 B134 B300 B1800 B150 B0 B19200 B1200 B9600 B600 B4800 B50 B2400 B110

Terminal interface values

TCSADRAIN TCSANOW TCOON TCIOFLUSH TCOFLUSH TCION TCIFLUSH TCSAFLUSH TCIOFF TCOOFF

c_cc field values

VEOF VEOL VERASE VINTR VKILL VQUIT VSUSP VSTART VSTOP VMIN VTIME NCCS

c_cflag field values

CLOCAL CREAD CSIZE CS5 CS6 CS7 CS8 CSTOPB HUPCL PARENB PARODD

c_iflag field values

BRKINT ICRNL IGNBRK IGNCR IGNPAR INLCR INPCK ISTRIP IXOFF IXON PARMRK

c_lflag field values

ECHO ECHOE ECHOK ECHONL ICANON IEXTEN ISIG NOFLSH TOSTOP

c_oflag field values

OPOST

PATHNAME CONSTANTS

Constants

_PC_CHOWN_RESTRICTED _PC_LINK_MAX _PC_MAX_CANON _PC_MAX_INPUT _PC_NAME_MAX _PC_NO_TRUNC _PC_PATH_MAX _PC_PIPE_BUF _PC_VDISABLE

POSIX CONSTANTS

Constants

_POSIX_ARG_MAX _POSIX_CHILD_MAX _POSIX_CHOWN_RESTRICTED _POSIX_JOB_CONTROL _POSIX_LINK_MAX _POSIX_MAX_CANON _POSIX_MAX_INPUT _POSIX_NAME_MAX _POSIX_NGROUPS_MAX _POSIX_NO_TRUNC _POSIX_OPEN_MAX _POSIX_PATH_MAX _POSIX_PIPE_BUF _POSIX_SAVED_IDS _POSIX_SSIZE_MAX _POSIX_STREAM_MAX _POSIX_TZNAME_MAX _POSIX_VDISABLE _POSIX_VERSION

SYSTEM CONFIGURATION

Constants

_SC_ARG_MAX _SC_CHILD_MAX _SC_CLK_TCK _SC_JOB_CONTROL
_SC_NGROUPS_MAX _SC_OPEN_MAX _SC_PAGESIZE _SC_SAVED_IDS
_SC_STREAM_MAX _SC_TZNAME_MAX _SC_VERSION

ERRNO

Constants

E2BIG EACCES EADDRINUSE EADDRNOTAVAIL EAFNOSUPPORT EAGAIN EALREADY
EBADF EBADMSG EBUSY ECANCELED ECHILD ECONNABORTED ECONNREFUSED
ECONNRESET EDEADLK EDESTADDRREQ EDOM EDQUOT EEXIST EFAULT EFBIG
EHOSTDOWN EHOSTUNREACH EIDRM EILSEQ EINPROGRESS EINTR EINVAL EIO
EISCONN EISDIR ELOOP EMFILE EMLINK EMSGSIZE ENAMETOOLONG ENETDOWN
ENETRESET ENETUNREACH ENFILE ENOBUFS ENODATA ENODEV ENOENT ENOEXEC
ENOLCK ENOLINK ENOMEM ENOMSG ENOPROTOOPT ENOSPC ENOSR ENOSTR ENOSYS
ENOTBLK ENOTCONN ENOTDIR ENOTEMPTY ENOTRECOVERABLE ENOTSOCK ENOTSUP
ENOTTY ENXIO EOPNOTSUPP EOTHER EOVERFLOW EOWNERDEAD EPERM
EPFNOSUPPORT EPIPE EPROCLIM EPROTO EPROTONOSUPPORT EPROTOTYPE
ERANGE EREMOTE ERESTART EROFS ESHUTDOWN ESOCKTNOSUPPORT ESPIPE
ESRCH ESTALE ETIME ETIMEDOUT ETOOMANYREFS ETXTBSY EUSERS
EWOULDBLOCK EXDEV

FCNTL

Constants

FD_CLOEXEC F_DUPFD F_GETFD F_GETFL F_GETLK F_OK F_RDLCK F_SETFD
F_SETFL F_SETLK F_SETLKW F_UNLCK F_WRLCK O_ACCMODE O_APPEND O_CREAT
O_EXCL O_NOCTTY O_NONBLOCK O_RDONLY O_RDWR O_TRUNC O_WRONLY

FLOAT

Constants

DBL_DIG DBL_EPSILON DBL_MANT_DIG DBL_MAX DBL_MAX_10_EXP
DBL_MAX_EXP DBL_MIN DBL_MIN_10_EXP DBL_MIN_EXP FLT_DIG FLT_EPSILON
FLT_MANT_DIG FLT_MAX FLT_MAX_10_EXP FLT_MAX_EXP FLT_MIN
FLT_MIN_10_EXP FLT_MIN_EXP FLT_RADIX FLT_ROUNDS LDBL_DIG
LDBL_EPSILON LDBL_MANT_DIG LDBL_MAX LDBL_MAX_10_EXP LDBL_MAX_EXP
LDBL_MIN LDBL_MIN_10_EXP LDBL_MIN_EXP

LIMITS

Constants

ARG_MAX CHAR_BIT CHAR_MAX CHAR_MIN CHILD_MAX INT_MAX INT_MIN
LINK_MAX LONG_MAX LONG_MIN MAX_CANON MAX_INPUT MB_LEN_MAX NAME_MAX
NGROUPS_MAX OPEN_MAX PATH_MAX PIPE_BUF SCHAR_MAX SCHAR_MIN SHRT_MAX
SHRT_MIN SSIZE_MAX STREAM_MAX TZNAME_MAX UCHAR_MAX UINT_MAX
ULONG_MAX USHRT_MAX

LOCALE

Constants

LC_ALL LC_COLLATE LC_CTYPE LC_MONETARY LC_NUMERIC LC_TIME

MATH

Constants

HUGE_VAL

SIGNAL

Constants

SA_NOCLDSTOP SA_NOCLDWAIT SA_NODEFER SA_ONSTACK SA_RESETHAND SA_RESTART SA_SIGINFO SIGABRT SIGALRM SIGCHLD SIGCONT SIGFPE SIGHUP SIGILL SIGINT SIGKILL SIGPIPE SIGQUIT SIGSEGV SIGSTOP SIGTERM SIGTSTP SIGTTIN SIGTTOU SIGUSR1 SIGUSR2 SIG_BLOCK SIG_DFL SIG_ERR SIG_IGN SIG_SETMASK SIG_UNBLOCK

STAT

Constants

S_IRGRP S_IROTH S_IRUSR S_IRWXG S_IRWXO S_IRWXU S_ISGID S_ISUID S_IWGRP S_IWOTH S_IWUSR S_IXGRP S_IXOTH S_IXUSR

Macros

S_ISBLK S_ISCHR S_ISDIR S_ISFIFO S_ISREG

STDLIB

Constants

EXIT_FAILURE EXIT_SUCCESS MB_CUR_MAX RAND_MAX

STDIO

Constants

BUFSIZ EOF FILENAME_MAX L_ctermid L_cuserid L_tmpname TMP_MAX

TIME

Constants

CLK_TCK CLOCKS_PER_SEC

UNISTD

Constants

R_OK SEEK_CUR SEEK_END SEEK_SET STDIN_FILENO STDOUT_FILENO STDERR_FILENO W_OK X_OK

WAIT

Constants

WNOHANG WUNTRACED

WNOHANG

: Do not suspend the calling process until a child process changes state but instead return immediately.

WUNTRACED

: Catch stopped child processes.

Macros

WIFEXITED WEXITSTATUS WIFSIGNALED WTERMSIG WIFSTOPPED WSTOPSIG

WIFEXITED

: WIFEXITED(${^CHILD_ERROR_NATIVE}) returns true if the child process exited normally (exit() or by falling off the end of main())

WEXITSTATUS

: WEXITSTATUS(${^CHILD_ERROR_NATIVE}) returns

the normal exit status of the child process (only meaningful if `WIFEXITED(${^CHILD_ERROR_NATIVE})` is true)

`WIFSIGNALED`

`WIFSIGNALED(${^CHILD_ERROR_NATIVE})` returns true if the child process terminated because of a signal

`WTERMSIG`

`WTERMSIG(${^CHILD_ERROR_NATIVE})` returns the signal the child process terminated for (only meaningful if `WIFSIGNALED(${^CHILD_ERROR_NATIVE})` is true)

`WIFSTOPPED`

`WIFSTOPPED(${^CHILD_ERROR_NATIVE})` returns true if the child process is currently stopped (can happen only if you specified the WUNTRACED flag to `waitpid()`)

`WSTOPSIG`

`WSTOPSIG(${^CHILD_ERROR_NATIVE})` returns the signal the child process was stopped for (only meaningful if `WIFSTOPPED(${^CHILD_ERROR_NATIVE})` is true)