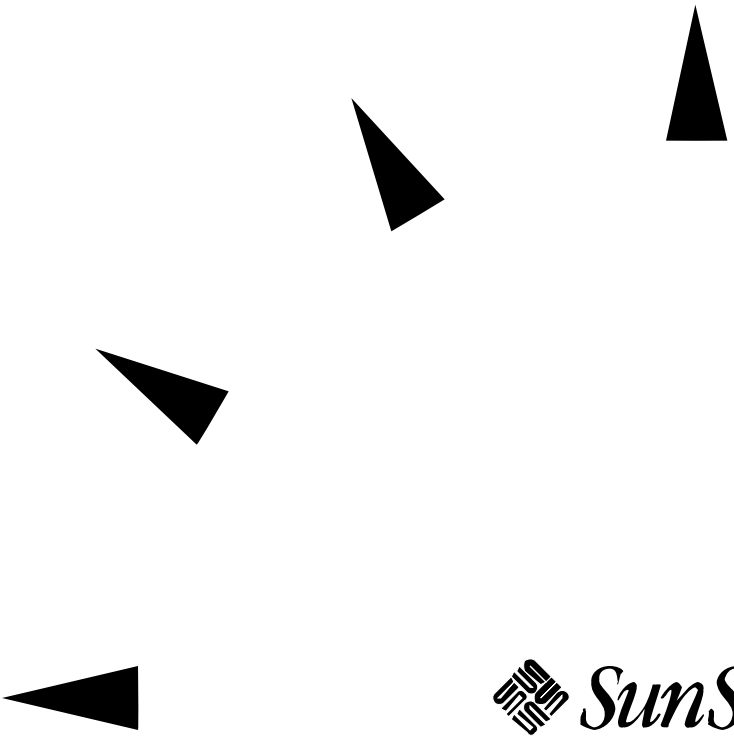
SunOS Reference Manual



Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft
A Sun Microsystems, Inc. Business

2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

Preface

OVERVIEW

A man page is provided for both the naive user, and sophisticated user who is familiar with the SunOS operating system and is in need of on-line information. 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.

The following contains a brief description of each section in the man pages 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 of this volume.

- 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, etc.

- 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 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver–Kernel Interface (DKI).

- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may 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 pathname is shown. Literal characters (commands and options) are in **bold** font and variables (arguments, parameters and substitution characters) are in *italic* font. 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:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument *must* be specified.

... Ellipses. Several values may 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 time.

PROTOCOL

This section occurs only in subsection 3R to indicate the protocol description file. The protocol specification pathname is always listed in **bold** font.

AVAILABILITY

This section briefly states any limitations on the availabilty of the command. These limitations could be hardware or software specific.

A specification of a class of hardware platform, such as **x86** or **SPARC**, denotes that the command or interface is applicable for the hardware platform specified.

In Section 1 and Section 1M, **AVAILABILITY** indicates which package contains the command being described on the manual page. In order to use the command, the specified package must have been installed with the operating system. If the package was not installed, see **pkgadd**(1) for information on how to upgrade.

MT-LEVEL

This section lists the **MT-LEVEL** of the library functions described in the Section 3 manual pages. The **MT-LEVEL** defines the libraries' ability to support threads. See **Intro**(3) for more information.

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, functions and such, are described under USAGE.

IOCTLS

This section appears on pages in Section 7 only. Only the device class which supplies appropriate parameters to the **ioctls**(2) system call is called **ioctls** and generates its own heading. IOCTLS for a specific device are listed alphabetically (on the man page for that specific device). IOCTLS are used for a particular class of devices all which have an **io** ending, such as **mtio**(7).

OPTIONS

This 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.

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 as **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 is provided as a *guidance* on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below 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 super-user,

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

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

FILES

This section lists all filenames 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.

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. Messages appear in **bold** font with the exception of variables, which are in *italic* font.

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.

NAME

Intro, intro – introduction to miscellany

DESCRIPTION

This section contains miscellaneous documentation such as character set tables, etc.

Name	Appears on Page	Description
advance	**regexp**(5)	regular expression compile and match routines
ascii	**ascii**(5)	map of ASCII character set
compile	**regexp**(5)	regular expression compile and match routines
environ	**environ**(5)	user environment
eqnchar	**eqnchar**(5)	special character definitions for eqn
fcntl	**fcntl**(5)	file control options
filesystem	**filesystem**(5)	file system organization
floatingpoint	**floatingpoint**(5)	IEEE floating point definitions
iconv	**iconv**(5)	code set conversion tables
langinfo	**langinfo**(5)	language information constants
man	**man**(5)	macros to format Reference Manual pages
mansun	**mansun**(5)	macros to format Reference Manual pages
math	**math**(5)	math functions and constants
me	**me**(5)	macros for formatting papers
ms	**ms**(5)	text formatting macros
nl_types	**nl_types**(5)	native language data types
prof	**prof**(5)	profile within a function
regexp	**regexp**(5)	regular expression compile and match routines
siginfo	**siginfo**(5)	signal generation information
signal	**signal**(5)	base signals
stat	**stat**(5)	data returned by stat system call
stdarg	**stdarg**(5)	handle variable argument list
step	**regexp**(5)	regular expression compile and match routines
term	**term**(5)	conventional names for terminals
types	**types**(5)	primitive system data types
ucontext	**ucontext**(5)	user context
values	**values**(5)	machine-dependent values
varargs	**varargs**(5)	handle variable argument list
wstat	**wstat**(5)	wait status

NAME

ascii – map of ASCII character set

DESCRIPTION

ascii is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel|
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si |
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb|
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us |
|040 sp |041 !  |042 "  |043 #  |044 $  |045 %  |046 &  |047 ´  |
|050 (  |051 )  |052 *  |053 +  |054 ,  |055 –  |056 .  |057 /  |
|060 0  |061 1  |062 2  |063 3  |064 4  |065 5  |066 6  |067 7  |
|070 8  |071 9  |072 :  |073 ;  |074 <  |075 =  |076 >  |077 ?  |
|100 @  |101 A  |102 B  |103 C  |104 D  |105 E  |106 F  |107 G  |
|110 H  |111 I  |112 J  |113 K  |114 L  |115 M  |116 N  |117 O  |
|120 P  |121 Q  |122 R  |123 S  |124 T  |125 U  |126 V  |127 W  |
|130 X  |131 Y  |132 Z  |133 [  |134 \  |135 ]  |136 ^  |137 _  |
|140 `  |141 a  |142 b  |143 c  |144 d  |145 e  |146 f  |147 g  |
|150 h  |151 i  |152 j  |153 k  |154 l  |155 m  |156 n  |157 o  |
|160 p  |161 q  |162 r  |163 s  |164 t  |165 u  |166 v  |167 w  |
|170 x  |171 y  |172 z  |173 {  |174 |  |175 }  |176 ~  |177 del|
```

```
| 00 nul| 01 soh| 02 stx| 03 etx| 04 eot| 05 enq| 06 ack| 07 bel|
| 08 bs | 09 ht | 0a nl | 0b vt | 0c np | 0d cr | 0e so | 0f si |
| 10 dle| 11 dc1| 12 dc2| 13 dc3| 14 dc4| 15 nak| 16 syn| 17 etb|
| 18 can| 19 em | 1a sub| 1b esc| 1c fs | 1d gs | 1e rs | 1f us |
| 20 sp | 21 !  | 22 "  | 23 #  | 24 $  | 25 %  | 26 &  | 27 ´  |
| 28 (  | 29 )  | 2a *  | 2b +  | 2c ,  | 2d –  | 2e .  | 2f /  |
| 30 0  | 31 1  | 32 2  | 33 3  | 34 4  | 35 5  | 36 6  | 37 7  |
| 38 8  | 39 9  | 3a :  | 3b ;  | 3c <  | 3d =  | 3e >  | 3f ?  |
| 40 @  | 41 A  | 42 B  | 43 C  | 44 D  | 45 E  | 46 F  | 47 G  |
| 48 H  | 49 I  | 4a J  | 4b K  | 4c L  | 4d M  | 4e N  | 4f O  |
| 50 P  | 51 Q  | 52 R  | 53 S  | 54 T  | 55 U  | 56 V  | 57 W  |
| 58 X  | 59 Y  | 5a Z  | 5b [  | 5c \  | 5d ]  | 5e ^  | 5f _  |
| 60 `  | 61 a  | 62 b  | 63 c  | 64 d  | 65 e  | 66 f  | 67 g  |
| 68 h  | 69 i  | 6a j  | 6b k  | 6c l  | 6d m  | 6e n  | 6f o  |
| 70 p  | 71 q  | 72 r  | 73 s  | 74 t  | 75 u  | 76 v  | 77 w  |
| 78 x  | 79 y  | 7a z  | 7b {  | 7c |  | 7d }  | 7e ~  | 7f del|
```

FILES

/usr/pub/ascii

NAME

environ – user environment

DESCRIPTION

When a process begins execution, **exec** routines make available an array of strings called the environment (see **exec**(2)). By convention, these strings have the form *variable=value*, for example, **PATH=/sbin:/usr/sbin**. These environmental variables provide a way to make information about a program's environment available to programs. The following environmental variables can be used by applications and are expected to be set in the target run-time environment.

HOME The name of the user's login directory, set by **login**(1) from the password file (see **passwd**(4)).

LANG The string used to specify localization information that allows users to work with different national conventions. The **setlocale**(3C) function looks for the **LANG** environment variable when it is called with "" as the *locale* argument. **LANG** is used as the default locale if the corresponding environment variable for a particular category is unset.

For example, when **setlocale**() is invoked as

setlocale(LC_CTYPE, ""),

setlocale() will query the **LC_CTYPE** environment variable first to see if it is set and non-null. If **LC_CTYPE** is not set or null, then **setlocale**() will check the **LANG** environment variable to see if it is set and non-null. If both **LANG** and **LC_CTYPE** are unset or null, the default **C** locale will be used to set the **LC_CTYPE** category.

Most commands will invoke

setlocale(LC_ALL, "")

prior to any other processing. This allows the command to be used with different national conventions by setting the appropriate environment variables.

The following environment variables are supported to correspond with each category of **setlocale**(3C):

LC_COLLATE This category specifies the collation sequence being used. The information corresponding to this category is stored in a database created by the **colltbl**(1M) command. This environment variable affects **strcoll**(3C) and **strxfrm**(3C).

LC_CTYPE This category specifies character classification, character conversion, and widths of multibyte characters. The information corresponding to this category is stored in a database created by the **chrtbl**(1M) command. The default **C** locale corresponds to the 7-bit ASCII character set. This environment variable is used by **ctype**(3C), **mbchar**(3C), and many commands; for example: **cat**(1), **ed**(1), **ls**(1), and **vi**(1).

LC_MESSAGES This category specifies the language of the message database being used. For example, an application may have one message database with French messages, and another database with German messages. Message databases are created by the **mkmsgs**(1) command. This environment variable is used by **exstr**(1), **gettxt**(1), **gettxt**(3C), and **srchtxt**(1).

LC_MONETARY This category specifies the monetary symbols and delimiters used for a particular locale. The information corresponding to this category is stored in a database created by the **montbl**(1M) command. This environment variable is used by **localeconv**(3C).

LC_NUMERIC This category specifies the decimal and thousands delimiters. The information corresponding to this category is stored in a database created by the **chrtbl**(1M) command. The default **C** locale corresponds to "**.**" as the decimal delimiter and no thousands delimiter. This environment variable is used by **localeconv**(3C), **printf**(3S), and **strtod**(3C).

LC_TIME This category specifies date and time formats. The information corresponding to this category is stored in a database specified in **strftime**(4). The default **C** locale corresponds to U.S. date and time formats. This environment variable is used by many commands and functions; for example: **at**(1), **calendar**(1), **date**(1), **strftime**(3C), and **getdate**(3C).

MSGVERB Controls which standard format message components **fmtmsg** selects when messages are displayed to **stderr** (see **fmtmsg**(1) and **fmtmsg**(3C)).

SEV_LEVEL Define severity levels and associate and print strings with them in standard format error messages (see **addseverity**(3C), **fmtmsg**(1), and **fmtmsg**(3C)).

NETPATH A colon-separated list of network identifiers. A network identifier is a character string used by the Network Selection component of the system to provide application-specific default network search paths. A network identifier must consist of non-**NULL** characters and must have a length of at least 1. No maximum length is specified. Network identifiers are normally chosen by the system administrator. A network identifier is also the first field in any **/etc/netconfig** file entry. **NETPATH** thus provides a link into the **/etc/netconfig** file and the information about a network contained in that network's entry. **/etc/netconfig** is maintained by the system administrator. The library routines described in **getnetpath**(3N) access the **NETPATH** environment variable.

NLSPATH Contains a sequence of templates which **catopen**(3C) uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more substitution fields, a filename and an optional suffix.

For example:

NLSPATH="/system/nlslib/%N.cat"

defines that **catopen**( ) should look for all message catalogs in the directory **/system/nlslib**, where the catalog name should be constructed from the *name* parameter passed to **catopen**( ), **%N**, with the suffix **.cat**.

Substitution fields consist of a **%** symbol, followed by a single-letter keyword. The following keywords are currently defined:

%N	The value of the *name* parameter passed to **catopen**( ).
%L	The value of **LANG**.
%l	The language element from **LANG**.
%t	The territory element from **LANG**.
%c	The codeset element from **LANG**.
%%	A single **%** character.

An empty string is substituted if the specified value is not currently defined. The separators ''**_**'' and ''**.**'' are not included in **%t** and **%c** substitutions.

Templates defined in **NLSPATH** are separated by colons (**:**). A leading colon or two adjacent colons (**::**) is equivalent to specifying **%N**.

For example:

NLSPATH=":%N.cat:/nlslib/%L/%N.cat"

indicates to **catopen**( ) that it should look for the requested message catalog in *name*, *name***.cat** and **/nlslib/$LANG/***name***.cat**.

PATH The sequence of directory prefixes that **sh**(1), **time**(1), **nice**(1), **nohup**(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (**:**). **login**(1) sets **PATH=/usr/bin**. For more detail, see **sh**(1).

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as **vi**(1), which may exploit special capabilities of that terminal.

TZ Time zone information. The contents of the environment variable named **TZ** are used by the functions **ctime**(3C), **localtime**( ) (see **ctime**(3C)), **strftime**(3C) and **mktime**(3C) to override the default timezone. If the first character of **TZ** is a colon (**:**), the behavior is implementation defined, otherwise **TZ** has the form:

std offset **[** *dst* **[** *offset* **],[** *start* **[** */time* **],** *end* **[** */time* **]]]**

std and *dst* Three or more bytes that are the designation for the standard (*std*) and daylight savings time (*dst*) timezones. Only *std* is required. If *dst* is missing, then daylight savings time does not apply in this locale. Upper- and lower-case letters are allowed. Any characters except a leading colon (**:**),

digits, a comma (**,**), a minus (–) or a plus (**+**) are allowed.

offset Indicates the value one must add to the local time to arrive at Coordinated Universal Time. The offset has the form:

hh **[:** *mm* **[:** *ss* **]]**

The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*) is required and may be a single digit. The *offset* following *std* is required. If no *offset* follows *dst* , daylight savings time is assumed to be one hour ahead of standard time. One or more digits may be used; the value is always interpreted as a decimal number. The hour must be between 0 and 24, and the minutes (and seconds) if present between 0 and 59. Out of range values may cause unpredictable behavior. If preceded by a ''–'', the timezone is east of the Prime Meridian; otherwise it is west (which may be indicated by an optional preceding ''+'' sign).

start / *time*, *end* / *time*

Indicate when to change to and back from daylight savings time, where *start/time* describes when the change from standard time to daylight savings time occurs, and *end/time* describes when the change back happens. Each *time* field describes when, in current local time, the change is made.

The formats of *start* and *end* are one of the following:

J*n* The Julian day *n* ($1 \le n \le 365$). Leap days are not counted. That is, in all years, February 28 is day 59 and March 1 is day 60. It is impossible to refer to the occasional February 29.

n The zero-based Julian day ($0 \le n \le 365$). Leap days are counted, and it is possible to refer to February 29.

M*m.n.d*
The d^{th} day, ($0 \le d \le 6$) of week *n* of month *m* of the year ($1 \le n \le 5$, $1 \le m \le 12$), where week 5 means ''the last *d*-day in month *m*'' which may occur in either the fourth or the fifth week). Week 1 is the first week in which the d^{th} day occurs. Day zero is Sunday.

Implementation specific defaults are used for *start* and *end* if these optional fields are not given.

The *time* has the same format as *offset* except that no leading sign (''–'' or ''+'') is allowed. The default, if *time* is not given is 02:00:00.

Further names may be placed in the environment by the **export** command and *name=value* arguments in **sh**(1), or by **exec**(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL**, **PS1**, **PS2**, **IFS** (see **profile**(4)).

SEE ALSO

cat(1), **date**(1), **ed**(1), **fmtmsg**(1), **login**(1), **ls**(1), **mkmsgs**(1), **nice**(1), **nohup**(1), **sh**(1), **sort**(1), **time**(1), **vi**(1), **chrtbl**(1M), **colltbl**(1M), **montbl**(1M), **exec**(2), **addseverity**(3C), **catopen**(3C), **ctime**(3C), **ctype**(3C), **fmtmsg**(3C), **getdate**(3C), **getnetpath**(3N), **gettxt**(3C), **localeconv**(3C), **mbchar**(3C), **mktime**(3C), **printf**(3S), **strcoll**(3C), **strftime**(3C),

strtod(3C), **strxfrm**(3C), **netconfig**(4), **passwd**(4), **profile**(4), **strftime**(4), **TIMEZONE**(4)

NAME

eqnchar – special character definitions for eqn

SYNOPSIS

eqn /usr/share/lib/pub/eqnchar [*filename*] | **troff** [*options*]

neqn /usr/share/lib/pub/eqnchar [*filename*] | **nroff** [*options*]

DESCRIPTION

The **eqnchar** command contains **troff**(1) and **nroff**(1) character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with **eqn**(1) and **neqn**. It contains definitions for the following characters:

ciplus	⊕	| |	‖	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	~	*rangle*	⟩	*blot*	□
-wig	≈	*hbar*	ℏ	*bullet*	•
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	<->	↔	*empty*	∅
=wig	≅	<=>	⇔	*member*	∈
star	✻	| <	⊀	*nomem*	∉
bigstar	✳	| >	⊁	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	⊾	*incl*	⊆
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≙	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊈
oppE	∃	*3quarter*	¾	*!supset*	⊉
angstrom	Å	*degree*	°		

FILES

/usr/share/lib/pub/eqnchar

SEE ALSO

eqn(1), **nroff**(1), **troff**(1)

NAME

fcntl – file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

The **<fcntl.h>** header defines the following requests and arguments for use by the functions **fcntl**(2) and **open**(2).

Values for *cmd* used by **fcntl** (the following values are unique):

F_DUPFD	Duplicate file descriptor
F_GETFD	Get file descriptor flags
F_SETFD	Set file descriptor flags
F_GETFL	Get file status flags
F_SETFL	Set file status flags
F_GETLK	Get record locking information
F_SETLK	Set record locking information
F_SETLKW	Set record locking information; wait if blocked

File descriptor flags used for **fcntl**:

FD_CLOEXEC	Close the file descriptor upon execution of an exec function (see **exec**(2))

Values for **l_type** used for record locking with **fcntl** (the following values are unique):

F_RDLCK	Shared or read lock
F_UNLCK	Unlock
F_WRLCK	Exclusive or write lock

The following three sets of values are bitwise distinct: Values for **oflag** used by **open**:

O_CREAT	Create file if it does not exist
O_EXCL	Exclusive use flag
O_NOCTTY	Do not assign controlling tty
O_TRUNC	Truncate flag

File status flags used for **open** and **fcntl**:

O_APPEND	Set append mode
O_NDELAY	Non-blocking mode
O_NONBLOCK	Non-blocking mode (POSIX)
O_DSYNC	Write I/O operations on the file descriptor complete as defined by synchronized I/O data integrity completion
O_RSYNC	Read I/O operations on the file descriptor complete at the same level of integrity as specified by the the **O_DSYNC** and **O_SYNC** flags. If both **O_DSYNC** and **O_RSYNC** are set in *oflag*, all I/O operations on the file descriptor complete as defined by synchronized I/O data integrity completion. If both **O_SYNC** and **O_RSYNC** are set in *oflag*, all I/O operations on the file descriptor complete as defined by synchronized I/O file

integrity completion.

O_SYNC When opening a regular file, this flag affects subsequent writes. If set, each **write**(2) will wait for both the file data and file status to be physically updated. Write I/O operations on the file descriptor complete as defined by synchronized I/O file integrity completion.

Mask for use with file access modes:

O_ACCMODE Mask for file access modes

File access modes used for **open** and **fcntl**:

O_RDONLY Open for reading only
O_RDWR Open for reading and writing
O_WRONLY Open for writing only

The structure **flock** describes a file lock. It includes the following members:

```
short   l_type;     /* Type of lock */
short   l_whence;   /* Flag for starting offset */
off_t   l_start;    /* Relative offset in bytes */
off_t   l_len;      /* Size; if 0 then until EOF */
long    l_sysid;    /* Returned with F_GETLK */
pid_t   l_pid;      /* Returned with F_GETLK */
```

SEE ALSO

creat(2), **exec**(2), **fcntl**(2), **open**(2), **fsync**(3C), **fdatasync**(3R)

NOTES

Data is successfully transferred for a write operation to a regular file when the system ensures that all data written is readable on any subsequent open of the file (even one that follows a system or power failure) in the absense of a failure of the physical storage medium.

Data is successfully transferred for a read operation when an image of the data on the physical storage medium is available to the requesting process.

Synchronized I/O data integrity completion (see **fdatasync**(3R)):

For reads, the operation has been completed or diagnosed if unsuccessful. The read is complete only when an image of the data has been successfully transferred to the requesting process. If there were any pending write requests affecting the data to be read at the time that the synchronized read operation was requested, these write requests will be successfully transferred prior to reading the data.

For writes, the operation has been completed or diagnosed if unsuccessful. The write is complete only when the data specified in the write request is successfully transferred, and all file system information required to retrieve the data is successfully transferred.

File attributes that are not necessary for data retrieval (access time, modification time, status change time) need not be successfully transferred prior to returning to the calling process.

Synchronized I/O file integrity completion (see **fsync**(3C)):

Identical to a synchronized I/O data integrity completion with the addition that all file attributes relative tothe I/O operation (including access time, modification time, status change time) will be successfully transferred prior to returning to the calling process.

NAME

filesystem – file system organization

SYNOPSIS

/
/usr
/export

DESCRIPTION

The file system tree is organized for administrative convenience. Distinct areas within the file system tree are provided for files that are private to one machine, files that can be shared by multiple machines of a common architecture, files that can be shared by all machines, and home directories. This organization allows sharable files to be stored on one machine but accessed by many machines using a remote file access mechanism such as NFS. Grouping together similar files makes the file system tree easier to upgrade and manage.

The file system tree consists of a root file system and a collection of mountable file systems. The **mount**(2) program attaches mountable file systems to the file system tree at mount points (directory entries) in the root file system or other previously mounted file systems. Two file systems, **/** (the root) and **/usr**, must be mounted in order to have a completely functional system. The root file system is mounted automatically by the kernel at boot time; the **/usr** file system is mounted by the system start-up script, which is run as part of the booting process.

Root File System

The root file system contains files that are unique to each machine. It contains the following directories:

/dev	Primary location for special files. Typically, device files are built to match the kernel and hardware configuration of the machine.
/dev/dsk	Block disk devices.
/dev/pts	Pseudo-terminal devices.
/dev/rdsk	Raw disk devices.
/dev/rmt	Raw tape devices.
/dev/sad	Entry points for the STREAMS Administrative driver.
/dev/term	Terminal devices.
/etc	Host-specific administrative configuration files and databases. **/etc** may be viewed as the directory that defines the machine's identity.
/etc/acct	Accounting system configuration information.
/etc/cron.d	Configuration information for **cron**(1M).
/etc/default	Defaults information for various programs.
/etc/dfs	Configuration information for exported file systems.

/etc/fs	Binaries organized by file system types for operations required before **/usr** is mounted.
/etc/inet	Configuration files for Internet services.
/etc/init.d	Shell scripts for transitioning between run levels.
/etc/lib	Shared libraries needed during booting.
/etc/lp	Configuration information for the printer subsystem.
/etc/mail	Mail subsystem configuration.
/etc/net	Configuration information for transport independent network services.
/etc/opt	Configuration information for optional packages.
/etc/rc0.d	Scripts for entering or leaving run level 0. See **init**(1M).
/etc/rc1.d	Scripts for entering or leaving run level 1. See **init**(1M).
/etc/rc2.d	Scripts for entering or leaving run level 2. See **init**(1M).
/etc/rc3.d	Scripts for entering or leaving run level 3. See **init**(1M).
/etc/saf	Service Access Facility files.
/etc/skel	Default profile scripts for new user accounts. See **useradd**(1M).
/etc/sm	Status monitor information.
/etc/sm.bak	Backup status monitor information.
/etc/tm	Trademark files; contents displayed at boot time.
/etc/uucp	UUCP configuration information. See **uucp**(1C).
/export	Default root of the exported file system tree.
/home	Default root of a subtree for user directories.
/kernel	Subtree of loadable kernel modules, including the base kernel itself, **/kernel/unix**. See **kernel**(1M).
/mnt	Default temporary mount point for file systems. This is an empty directory on which file systems may be temporarily mounted.
/opt	Root of a subtree for add-on application packages.
/proc	Root of a subtree for the process file system.
/sbin	Essential executables used in the booting process and in manual system recovery. The full complement of utilities is available only after **/usr** is mounted.
/tmp	Temporary files; cleared during the boot operation.
/var	Root of a subtree for varying files. Varying files are files that are unique to a machine but that can grow to an arbitrary (that is, variable) size. An example is a log file.

/var/adm	System logging and accounting files.
/var/cron	Log files for **cron**(1M).
/var/mail	Directory where users' mail is kept.
/var/news	Community service messages. Note: this is not the same as USENET-style news.
/var/nis	NIS+ databases.
/var/opt	Root of a subtree for varying files associated with optional software packages.
/var/preserve	Backup files for **vi**(1) and **ex**(1).
/var/sadm	Databases maintained by the software package management utilities.
/var/saf	Service access facility logging and accounting files.
/var/spool	Root directory for files used in printer spooling, mail delivery, **cron**(1M), **at**(1), etc.
/var/spool/cron	**cron**(1M) and **at**(1) spooling files.
/var/spool/locks	Spooling lock files.
/var/spool/lp	Line printer spool files. See **lp**(1).
/var/spool/mqueue	Mail queued for delivery.
/var/spool/pkg	Spooled packages.
/var/spool/uucp	Queued **uucp**(1C) jobs.
/var/spool/uucppublic	Files deposited by **uucp**(1C).
/var/tmp	Transitory files; this directory is *not* cleared during the boot operation.
/var/uucp	**uucp**(1C) log and status files.
/var/yp	Databases needed for backwards compatibility with NIS and **ypbind**(1M); unnecessary after full transition to NIS+.

/usr File System

Because it is desirable to keep the root file system small and not volatile, on disk-based systems larger file systems are often mounted on **/home**, **/opt**, **/usr**, and **/var**.

The file system mounted on **/usr** contains architecture-dependent and architecture-independent sharable files. The subtree rooted at **/usr/share** contains architecture-independent sharable files; the rest of the **/usr** tree contains architecture-dependent files. By mounting a common remote file system, a group of machines with a common architecture may share a single **/usr** file system. A single **/usr/share** file system can be shared by machines of any architecture. A machine acting as a file server may export many different **/usr** file systems to support several different architectures and operating system releases. Clients usually mount **/usr** read-only so that they do not accidentally change any shared files.

The **/usr** file system contains the following subdirectories:

/usr/4lib	**a.out** libraries for the Binary Compatibility Package. See *Solaris Binary Compatibility Guide*.
/usr/bin	Primary location for standard system utilities.
/usr/bin/sunview1	SunView executables. This directory is only present when the Binary Compatibility Package is installed.
/usr/ccs	C compilation system.
/usr/ccs/bin	C compilation commands and system utilities.
/usr/ccs/lib	Libraries and auxiliary files.
/usr/demo	Demo programs and data.
/usr/dt	root of a subtree for CDE Motif.
/usr/dt/bin	Primary location for CDE Motif system utilities.
/usr/dt/include	Header files for CDE Motif.
/usr/dt/lib	Libraries for CDE Motif.
/usr/dt/man	On-line reference manual pages for CDE Motif.
/usr/games	Game binaries and data.
/usr/include	Include headers (for C programs, etc).
/usr/kvm	Implementation architecture-specific binaries and libraries.
/usr/lib	Program libraries, various architecture-dependent databases, and executables not invoked directly by the user (system daemons, etc).
/usr/lib/acct	Accounting scripts and binaries. See **acct**(1M).
/usr/lib/dict	Database files for **spell**(1).
/usr/lib/class	Scheduling class-specific directories containing executables for **priocntl**(1) and **dispadmin**(1M).
/usr/lib/font	**troff**(1) font description files.
/usr/lib/fs	File system type dependent modules; generally not intended to be invoked directly by the user.
/usr/lib/iconv	Conversion tables for **iconv**(1).
/usr/lib/libp	Profiled libraries.
/usr/lib/locale	Localization databases.
/usr/lib/lp	Line printer subsystem databases and back-end executables.
/usr/lib/mail	Auxiliary programs for the **mail**(1) subsystem.
/usr/lib/netsvc	Internet network services.
/usr/lib/nfs	Auxiliary NFS-related programs and daemons.
/usr/lib/pics	Position Independent Code (PIC) archives needed to rebuild the run-time linker.

/usr/lib/refer	Auxiliary programs for **refer**(1).
/usr/lib/sa	Scripts and commands for the system activity report package. See **sar**(1).
/usr/lib/saf	Auxiliary programs and daemons related to the service access facility.
/usr/lib/spell	Auxiliary programs and databases for **spell**(1). This directory is only present when the Binary Compatibility Package is installed.
/usr/lib/uucp	Auxiliary programs and daemons for **uucp**(1C).
/usr/local	Commands local to a site.
/usr/net/servers	Entry points for foreign name service requests relayed using the network listener. See **listen**(1M).
/usr/oasys	Commands and files related to the optional Framed Access Command Environment (FACE) package. See **face**(1).
/usr/old	Programs that are being phased out.
/usr/openwin	Installation or mount point for the OpenWindows software.
/usr/sadm	System administration files and directories.
/usr/sadm/bin	Binaries for the Form and Menu Language Interpreter (FMLI) scripts. See **fmli**(1).
/usr/sadm/install	Executables and scripts for package management.
/usr/sbin	Executables for system administration.
/usr/sbin/static	Statically linked version of selected programs from **/usr/bin** and **/usr/sbin**. These are used to recover from broken dynamic linking and before all pieces necessary for dynamic linking are present.
/usr/share	Architecture-independent sharable files.
/usr/share/man	On-line reference manual pages (if present).
/usr/share/lib	Architecture-independent databases.
/usr/share/lib/keytables	Keyboard layout description tables.
/usr/share/lib/mailx	Help files for **mailx**(1).
/usr/share/lib/nterm	**nroff**(1) terminal tables.
/usr/share/lib/pub	Character set data files.
/usr/share/lib/spell	Auxiliary scripts and databases for **spell**(1).
/usr/share/lib/tabset	Tab setting escape sequences.
/usr/share/lib/terminfo	Terminal description files for **terminfo**(4).

/usr/share/lib/tmac	Macro packages and related files for text processing tools, for example, **nroff**(1) and **troff**(1).
/usr/share/lib/zoneinfo	Time zone information.
/usr/share/src	Source code for utilities and libraries.
/usr/snadm	SNAG files.
/usr/ucb	Berkeley compatibility package binaries. See *Solaris Source Compatibility Guide*.
/usr/ucbinclude	Berkeley compatibility package headers.
/usr/ucblib	Berkeley compatibility package libraries.
/usr/vmsys	Commands and files related to the optional FACE package. See **face**(1). Berkeley compatibility package libraries.

/export File System

A machine with disks may export root file systems, swap files, and **/usr** file systems to diskless or partially-disked machines that mount them into the standard file system hierarchy. The standard directory tree for sharing these file systems from a server is:

/export	The default root of the exported file system tree.
/export/exec/*architecture-name*	The exported **/usr** file system supporting *architecture-name* for the current release.
/export/exec/*architecture-name***.***release-name*	The exported **/usr** file system supporting *architecture-name* for *release-name*.
/export/exec/share	The exported common **/usr/share** directory tree.
/export/exec/share.*release-name*	The exported common **/usr/share** directory tree for *release-name*.
/export/root/*hostname*	The exported root file system for *hostname*.
/export/swap/*hostname*	The exported swap file for *hostname*.
/export/var/*hostname*	The exported **/var** directory tree for *hostname*.

SEE ALSO

at(1), **ex**(1), **face**(1), **fmli**(1), **iconv**(1), **lp**(1), **mail**(1), **mailx**(1), **nroff**(1), **priocntl**(1), **refer**(1), **sar**(1), **sh**(1), **spell**(1), **troff**(1), **uucp**(1C), **vi**(1), **acct**(1M), **cron**(1M), **dispadmin**(1M), **fsck**(1M), **init**(1M), **kernel**(1M), **mknod**(1M), **mount**(1M), **useradd**(1M), **ypbind**(1M), **mount**(2), **intro**(4), **terminfo**(4)

Solaris Binary Compatibility Guide
Solaris Source Compatibility Guide

NAME

floatingpoint – IEEE floating point definitions

SYNOPSIS

#include <floatingpoint.h>

DESCRIPTION

This file defines constants, types, and functions used to implement standard floating point according to ANSI/IEEE Std 754-1985. The functions are implemented in **libc**. The included header file **<sys/ieeefp.h>** defines certain types of interest to the kernel.

IEEE Rounding Modes:

fp_direction_type The type of the IEEE rounding direction mode. Note: the order of enumeration varies according to hardware.

fp_precision_type The type of the IEEE rounding precision mode, which only applies on systems that support extended precision such as machines based on the Intel 80387 FPU or the 80486.

SIGFPE handling:

sigfpe_code_type The type of a **SIGFPE** code.

sigfpe_handler_type The type of a user-definable **SIGFPE** exception handler called to handle a particular **SIGFPE** code.

SIGFPE_DEFAULT A macro indicating the default **SIGFPE** exception handling, namely to perform the exception handling specified by the user, if any, and otherwise to dump core using **abort**(3C).

SIGFPE_IGNORE A macro indicating an alternate **SIGFPE** exception handling, namely to ignore and continue execution.

SIGFPE_ABORT A macro indicating an alternate **SIGFPE** exception handling, namely to abort with a core dump.

IEEE Exception Handling:

N_IEEE_EXCEPTION The number of distinct IEEE floating-point exceptions.

fp_exception_type The type of the **N_IEEE_EXCEPTION** exceptions. Each exception is given a bit number.

fp_exception_field_type
The type intended to hold at least **N_IEEE_EXCEPTION** bits corresponding to the IEEE exceptions numbered by **fp_exception_type**. Thus **fp_inexact** corresponds to the least significant bit and **fp_invalid** to the fifth least significant bit. Note: some operations may set more than one exception.

IEEE Formats and Classification:

single; **extended**; **quadruple**
Definitions of IEEE formats.

fp_class_type An enumeration of the various classes of IEEE values and symbols.

IEEE Base Conversion:

The functions described under **floating_to_decimal**(3) and **decimal_to_floating**(3) satisfy not only the IEEE Standard, but also the stricter requirements of correct rounding for all arguments.

DECIMAL_STRING_LENGTH
: The length of a **decimal_string**.

decimal_string
: The digit buffer in a **decimal_record**.

decimal_record
: The canonical form for representing an unpacked decimal floating-point number.

decimal_form
: The type used to specify fixed or floating binary to decimal conversion.

decimal_mode
: A struct that contains specifications for conversion between binary and decimal.

decimal_string_form
: An enumeration of possible valid character strings representing floating-point numbers, infinities, or NaNs.

FILES

/usr/include/sys/ieeefp.h

SEE ALSO

abort(3C), **decimal_to_floating**(3), **econvert**(3), **floating_to_decimal**(3), **sigfpe**(3), **string_to_decimal**(3), **strtod**(3C)

NAME

iconv – code set conversion tables

DESCRIPTION

The following code set conversions are supported:

Code Set Conversions Supported				
Code	Symbol	Target Code	Symbol	comment
ISO 646	646	ISO 8859-1	8859	US Ascii
ISO 646de	646de	ISO 8859-1	8859	German
ISO 646da	646da	ISO 8859-1	8859	Danish
ISO 646en	646en	ISO 8859-1	8859	English Ascii
ISO 646es	646es	ISO 8859-1	8859	Spanish
ISO 646fr	646fr	ISO 8859-1	8859	French
ISO 646it	646it	ISO 8859-1	8859	Italian
ISO 646sv	646sv	ISO 8859-1	8859	Swedish
ISO 8859-1	8859	ISO 646	646	7 bit Ascii
ISO 8859-1	8859	ISO 646de	646de	German
ISO 8859-1	8859	ISO 646da	646da	Danish
ISO 8859-1	8859	ISO 646en	646en	English Ascii
ISO 8859-1	8859	ISO 646es	646es	Spanish
ISO 8859-1	8859	ISO 646fr	646fr	French
ISO 8859-1	8859	ISO 646it	646it	Italian
ISO 8859-1	8859	ISO 646sv	646sv	Swedish

The conversions are performed according to the tables following. All values in the tables are given in octal.

ISO 646 (US ASCII) to ISO 8859-1

For the conversion of ISO 646 to ISO 8859-1 all characters in ISO 646 can be mapped unchanged to ISO 8859-1

ISO 646de (GERMAN) to ISO 8859-1

For the conversion of ISO 646de to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646de	ISO 8859-1
100	247
133	304
134	326
135	334
173	344
174	366
175	374
176	337

ISO 646da (DANISH) to ISO 8859-1

For the conversion of ISO 646da to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646da	ISO 8859-1
133	306
134	330
135	305
173	346
174	370
175	345

ISO 646en (ENGLISH ASCII) to ISO 8859-1

For the conversion of ISO 646en to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646en	ISO 8859-1
043	243

ISO 646fr (FRENCH) to ISO 8859-1

For the conversion of ISO 646fr to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646fr	ISO 8859-1
043	243
100	340
133	260
134	347
135	247
173	351
174	371
175	350
176	250

ISO 646it (ITALIAN) to ISO 8859-1

For the conversion of ISO 646it to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646it	ISO 8859-1
043	243
100	247
133	260
134	347
135	351
140	371
173	340
174	362
175	350
176	354

ISO 646es (SPANISH) to ISO 8859-1

For the conversion of ISO 646es to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646es	ISO 8859-1
100	247
133	241
134	321
135	277
173	260
174	361
175	347

ISO 646sv (SWEDISH) to ISO 8859-1

For the conversion of ISO 646sv to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646sv	ISO 8859-1
100	311
133	304
134	326
135	305
136	334
140	351
173	344
174	366
175	345
176	374

ISO 8859-1 to ISO 646 (ASCII)

For the conversion of ISO 8859-1 to ISO 646 all characters not in the following table are mapped unchanged.

Converted to Underscore '_' (137)
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
340 341 342 343 344 345 346 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 366 367
370 371 372 373 374 375 376 377

ISO 8859-1 to ISO 646de (GERMAN)

For the conversion of ISO 8859-1 to ISO 646de all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646de
247	100
304	133
326	134
334	135
337	176
344	173
366	174
374	175

Converted to Underscore '_' (137)
100 133 134 135 173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 327
330 331 332 333 335 336 337
340 341 342 343 345 346 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 367
370 371 372 373 375 376 377

ISO 8859-1 to ISO 646da (DANISH)

For the conversion of ISO 8859-1 to ISO 646da all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646da
305	135
306	133
330	134
345	175
346	173
370	174

Converted to Underscore '_' (137)
133 134 135 173 174 175
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
331 332 333 334 335 336 337
340 341 342 343 344 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 366 367
371 372 373 374 376 377

ISO 8859-1 to ISO 646en (ENGLISH ASCII)

For the conversion of ISO 8859-1 to ISO 646en all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646en
243	043

Converted to Underscore '_' (137)
043
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
340 341 342 343 344 345 346 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 366 367
370 371 372 373 374 375 376 377

ISO 8859-1 to ISO 646fr (FRENCH)

For the conversion of ISO 8859-1 to ISO 646fr all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646fr
243	043
247	135
250	176
260	133
340	100
347	134
350	175
351	173
371	174

Converted to Underscore '_' (137)
043
100 133 134 135 173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 244 245 246
251 252 253 254 255 256 257
261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
341 342 343 344 345 346
352 353 354 355 356 357
360 361 362 363 364 365 366 367
370 372 373 374 375 376 377

ISO 8859-1 to ISO 646it (ITALIAN)

For the conversion of ISO 8859-1 to ISO 646it all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646it
243	043
247	100
260	133
340	173
347	134
350	175
351	135
354	176
362	174
371	140

Converted to Underscore '_' (137)
043
100 133 134 135 173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 244 245 246
250 251 252 253 254 255 256 257
261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
341 342 343 344 345 346
352 353 354 355 356 357
360 361 363 364 365 366 367
370 372 373 374 375 376 377

ISO 8859-1 to ISO 646es (SPANISH)

For the conversion of ISO 8859-1 to ISO 646es all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646es
241	133
247	100
260	173
277	135
321	134
347	175
361	174

Converted to Underscore '_' (137)
100 133 134 135 173 174 175
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 242 243 244 245 246
250 251 252 253 254 255 256 257
261 262 263 264 265 266 267
270 271 272 273 274 275 276
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 322 323 324 325 326 327
330 331 332 333 334 335 336 337
340 341 342 343 344 345 346
350 351 352 353 354 355 356 357
360 362 363 364 365 366 367
370 371 372 373 374 375 376 377

ISO 8859-1 to ISO 646sv (SWEDISH)

For the conversion of ISO 8859-1 to ISO 646sv all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646sv
304	133
305	135
311	100
326	134
334	136
344	173
345	175
351	140
366	174
374	176

Converted to Underscore '_' (137)
100 133 134 135 136 140
173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 306 307
310 312 313 314 315 316 317
320 321 322 323 324 325 327
330 331 332 333 335 336 337
340 341 342 343 346 347
350 352 353 354 355 356 357
360 361 362 363 364 365 367
370 371 372 373 375 376 377

FILES

/usr/lib/iconv/iconv_data	lists the conversions supported
/usr/lib/iconv/*.t	conversion tables

SEE ALSO

iconv(1)

NAME

langinfo – language information constants

SYNOPSIS

#include <langinfo.h>

DESCRIPTION

This header contains the constants used to identify items of langinfo data. The mode of *items* is given in **nl_types**.

DAY_1	Locale's equivalent of 'sunday'
DAY_2	Locale's equivalent of 'monday'
DAY_3	Locale's equivalent of 'tuesday'
DAY_4	Locale's equivalent of 'wednesday'
DAY_5	Locale's equivalent of 'thursday'
DAY_6	Locale's equivalent of 'friday'
DAY_7	Locale's equivalent of 'saturday'
ABDAY_1	Locale's equivalent of 'sun'
ABDAY_2	Locale's equivalent of 'mon'
ABDAY_3	Locale's equivalent of 'tue'
ABDAY_4	Locale's equivalent of 'wed'
ABDAY_5	Locale's equivalent of 'thur'
ABDAY_6	Locale's equivalent of 'fri'
ABDAY_7	Locale's equivalent of 'sat'
MON_1	Locale's equivalent of 'january'
MON_2	Locale's equivalent of 'february'
MON_3	Locale's equivalent of 'march'
MON_4	Locale's equivalent of 'april'
MON_5	Locale's equivalent of 'may'
MON_6	Locale's equivalent of 'june'
MON_7	Locale's equivalent of 'july'
MON_8	Locale's equivalent of 'august'
MON_9	Locale's equivalent of 'september'
MON_10	Locale's equivalent of 'october'
MON_11	Locale's equivalent of 'november'
MON_12	Locale's equivalent of 'december'
ABMON_1	Locale's equivalent of 'jan'
ABMON_2	Locale's equivalent of 'feb'
ABMON_3	Locale's equivalent of 'mar'
ABMON_4	Locale's equivalent of 'apr'
ABMON_5	Locale's equivalent of 'may'
ABMON_6	Locale's equivalent of 'jun'
ABMON_7	Locale's equivalent of 'jul'
ABMON_8	Locale's equivalent of 'aug'
ABMON_9	Locale's equivalent of 'sep'
ABMON_10	Locale's equivalent of 'oct'
ABMON_11	Locale's equivalent of 'nov'
ABMON_12	Locale's equivalent of 'dec'

RADIXCHAR	Locale's equivalent of '.'
THOUSEP	Locale's equivalent of ','
YESSTR	Locale's equivalent of 'yes'
NOSTR	Locale's equivalent of 'no'
CRNCYSTR	Locale's currency symbol
D_T_FMT	Locale's default format for date and time
D_FMT	Locale's default format for the date
T_FMT	Locale's default format for the time
AM_STR	Locale's equivalent of 'AM'
PM_STR	Locale's equivalent of 'PM'

This information is retrieved by **nl_langinfo**.

The items **CRNCYSTR**, **RADIXCHAR** and **THOUSEP** are extracted from the fields **currency_symbol**, **decimal_point** and **thousands_sep** in the structure returned by **localeconv**.

The items **T_FMT**, **D_FMT**, **D_T_FMT**, **YESSTR** and **NOSTR** are retrieved from a special message catalog named **Xopen_info** which should be generated for each locale supported and installed in the appropriate directory [see **gettxt**(3C) and **mkmsgs**(1)]. This catalog should have the messages in the order **T_FMT**, **D_FMT**, **D_T_FMT**, **YESSTR** and **NOSTR**.

All other items are as returned by **strftime**.

SEE ALSO

mkmsgs(1), **chrtbl**(1M), **gettxt**(3C), **localeconv**(3C), **nl_langinfo**(3C), **strftime**(3C), **nl_types**(5)

NAME

man – macros to format Reference Manual pages

SYNOPSIS

nroff –**man** *filename*...

troff –**man** *filename*...

DESCRIPTION

These macros are used to lay out the reference pages in this manual. Note: if *filename* contains format input for a preprocessor, the commands shown above must be piped through the appropriate preprocessor. This is handled automatically by the **man**(1) command. See the ''Conventions'' section.

Any text argument *t* may be zero to six words. Quotes may be used to include SPACE characters in a "word". If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way **.I** may be used to italicize a whole line, or **.SB** may be used to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by **–man**:

***R** '®', '(Reg)' in **nroff**.
***S** Change to default type size.

Requests

* n.t.l. = next text line; p.i. = prevailing indent

Request	*Cause Break*	*If no Argument*	*Explanation*
.B *t*	no	*t*=n.t.l.*	Text is in bold font.
.BI *t*	no	*t*=n.t.l.	Join words, alternating bold and italic.
.BR *t*	no	*t*=n.t.l.	Join words, alternating bold and roman.
.DT	no	.5i 1i...	Restore default tabs.
.HP *i*	yes	*i*=p.i.*	Begin paragraph with hanging indent. Set prevailing indent to *i*.
.I *t*	no	*t*=n.t.l.	Text is italic.
.IB *t*	no	*t*=n.t.l.	Join words, alternating italic and bold.
.IP *x i*	yes	*x*=""	Same as **.TP** with tag *x*.
.IR *t*	no	*t*=n.t.l.	Join words, alternating italic and roman.
.IX *t*	no	-	Index macro, for SunSoft internal use.
.LP	yes	-	Begin left-aligned paragraph. Set prevailing indent to .5i.
.P	yes	-	Same as **.LP**.
.PD *d*	no	*d*=.4v	Set vertical distance between paragraphs.
.PP	yes	-	Same as **.LP**.
.RE	yes	-	End of relative indent. Restores prevailing indent.
.RB *t*	no	*t*=n.t.l.	Join words, alternating roman and bold.

.RI *t*	no	*t*=n.t.l.	Join words, alternating roman and italic.
.RS *i*	yes	*i*=p.i.	Start relative indent, increase indent by *i*. Sets prevailing indent to .5i for nested indents.
.SB *t*	no	-	Reduce size of text by 1 point, make text bold.
.SH *t*	yes	-	Section Heading.
.SM *t*	no	*t*=n.t.l.	Reduce size of text by 1 point.
.SS *t*	yes	*t*=n.t.l.	Section Subheading.
.TH *n s d f m*	yes	-	Begin reference page *n*, of of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Sets prevailing indent and tabs to .5i.
.TP *i*	yes	*i*=p.i.	Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*.
.TX *t p*	no	-	Resolve the title abbreviation *t*; join to punctuation mark (or text) *p*.

Conventions

When formatting a manual page, **man** examines the first line to determine whether it requires special processing. For example a first line consisting of:

'\" t

indicates that the manual page must be run through the **tbl**(1) preprocessor.

A typical manual page for a command or function is laid out as follows:

.TH *title* [1-9]

The name of the command or function, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

.SH NAME

The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in roman font, this section contains no **troff**(1) commands or escapes, and no macro requests. It is used to generate the **windex** database, which is used by the **whatis**(1) command.

.SH SYNOPSIS

Commands:

The syntax of the command and its arguments, as typed on the command line. When in boldface, a word must be typed exactly as printed. When in italics, a word can be replaced with an argument that you supply. References to bold or italicized items are not capitalized in other sections, even when they begin a sentence.

Syntactic symbols appear in roman face:

[ ] An argument, when surrounded by brackets is optional.

| Arguments separated by a vertical bar are exclusive. You can supply only one item from such a list.

... Arguments followed by an ellipsis can be repeated. When an ellipsis follows a bracketed set, the expression within the brackets can be repeated.

Functions:

If required, the data declaration, or **#include** directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

.SH DESCRIPTION

A narrative overview of the command or function's external behavior. This includes how it interacts with files or data, and how it handles the standard input, standard output and standard error. Internals and implementation details are normally omitted. This section attempts to provide a succinct overview in answer to the question, "what does it do?"

Literal text from the synopsis appears in constant width, as do literal filenames and references to items that appear elsewhere in the reference manuals. Arguments are italicized.

If a command interprets either subcommands or an input grammar, its command interface or input grammar is normally described in a USAGE section, which follows the OPTIONS section. The DESCRIPTION section only describes the behavior of the command itself, not that of subcommands.

.SH OPTIONS

The list of options along with a description of how each affects the command's operation.

.SH FILES

A list of files associated with the command or function.

.SH SEE ALSO

A comma-separated list of related manual pages, followed by references to other published materials.

.SH DIAGNOSTICS

A list of diagnostic messages and an explanation of each.

.SH BUGS

A description of limitations, known defects, and possible problems associated with the command or function.

FILES

/usr/share/lib/tmac/an
/usr/share/man/windex

SEE ALSO

man(1), **nroff**(1), **troff**(1), **whatis**(1)

Dale Dougherty and Tim O'Reilly, *Unix Text Processing*

NAME

mansun – macros to format Reference Manual pages

SYNOPSIS

nroff –**mansun** *filename*...

troff –**mansun** *filename*...

DESCRIPTION

These macros are used to lay out the reference pages in this manual. Note: if *filename* contains format input for a preprocessor, the commands shown above must be piped through the appropriate preprocessor. This is handled automatically by **man**(1). See the ''Conventions'' section.

Any text argument *t* may be zero to six words. Quotes may be used to include SPACE characters in a "word". If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way **.I** may be used to italicize a whole line, or **.SB** may be used to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by **–mansun**:

***R** '®', '(Reg)' in **nroff**.
***S** Change to default type size.

Requests

* n.t.l. = next text line; p.i. = prevailing indent

Request	*Cause Break*	*If no Argument*	*Explanation*
.B *t*	no	*t*=n.t.l.*	Text is in bold font.
.BI *t*	no	*t*=n.t.l.	Join words, alternating bold and italic.
.BR *t*	no	*t*=n.t.l.	Join words, alternating bold and Roman.
.DT	no	.5i 1i...	Restore default tabs.
.HP *i*	yes	*i*=p.i.*	Begin paragraph with hanging indent. Set prevailing indent to *i*.
.I *t*	no	*t*=n.t.l.	Text is italic.
.IB *t*	no	*t*=n.t.l.	Join words, alternating italic and bold.
.IP *x i*	yes	*x*=""	Same as **.TP** with tag *x*.
.IR *t*	no	*t*=n.t.l.	Join words, alternating italic and Roman.
.IX *t*	no	-	Index macro, for SunSoft internal use.
.LP	yes	-	Begin left-aligned paragraph. Set prevailing indent to .5i.
.P	yes	-	Same as **.LP**.
.PD *d*	no	*d*=.4v	Set vertical distance between paragraphs.
.PP	yes	-	Same as **.LP**.
.RE	yes	-	End of relative indent. Restores prevailing indent.
.RB *t*	no	*t*=n.t.l.	Join words, alternating Roman and bold.

.RI *t*	no	*t*=n.t.l.	Join words, alternating Roman and italic.
.RS *i*	yes	*i*=p.i.	Start relative indent, increase indent by *i*. Sets prevailing indent to .5i for nested indents.
.SB *t*	no	-	Reduce size of text by 1 point, make text bold.
.SH *t*	yes	-	Section Heading.
.SM *t*	no	*t*=n.t.l.	Reduce size of text by 1 point.
.SS *t*	yes	*t*=n.t.l.	Section Subheading.
.TH *n s d f m*	yes	-	Begin reference page *n*, of of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Sets prevailing indent and tabs to .5i.
.TP *i*	yes	*i*=p.i.	Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*.
.TX *t p*	no	-	Resolve the title abbreviation *t*; join to punctuation mark (or text) *p*.

Conventions

When formatting a manual page, **mansun** examines the first line to determine whether it requires special processing. For example a first line consisting of:

'\" t

indicates that the manual page must be run through the **tbl**(1) preprocessor.

A typical manual page for a command or function is laid out as follows:

.TH *title* [1-8]

The name of the command or function, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

.SH NAME

The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in Roman font, this section contains no **troff**(1) commands or escapes, and no macro requests. It is used to generate the **windex** database, which is used by the **whatis**(1) command.

.SH SYNOPSIS

Commands:

The syntax of the command and its arguments, as typed on the command line. When in boldface, a word must be typed exactly as printed. When in italics, a word can be replaced with an argument that you supply. References to bold or italicized items are not capitalized in other sections, even when they begin a sentence.

Syntactic symbols appear in Roman face:

[ ] An argument, when surrounded by brackets is optional.

| Arguments separated by a vertical bar are exclusive. You can supply only one item from such a list.

... Arguments followed by an ellipsis can be repeated. When an ellipsis follows a bracketed set, the expression within the brackets can be repeated.

Functions:

If required, the data declaration, or **#include** directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

.SH DESCRIPTION

A narrative overview of the command or function's external behavior. This includes how it interacts with files or data, and how it handles the standard input, standard output and standard error. Internals and implementation details are normally omitted. This section attempts to provide a succinct overview in answer to the question, "what does it do?"

Literal text from the synopsis appears in constant width, as do literal filenames and references to items that appear elsewhere in the reference manuals. Arguments are italicized.

If a command interprets either subcommands or an input grammar, its command interface or input grammar is normally described in a USAGE section, which follows the OPTIONS section. The DESCRIPTION section only describes the behavior of the command itself, not that of subcommands.

.SH OPTIONS

The list of options along with a description of how each affects the command's operation.

.SH FILES

A list of files associated with the command or function.

.SH SEE ALSO

A comma-separated list of related manual pages, followed by references to other published materials.

.SH DIAGNOSTICS

A list of diagnostic messages and an explanation of each.

.SH BUGS

A description of limitations, known defects, and possible problems associated with the command or function.

FILES

/usr/share/lib/tmac/ansun
/usr/share/man/windex

SEE ALSO

man(1), **nroff**(1), **troff**(1), **whatis**(1)

Dale Dougherty and Tim O'Reilly, *Unix Text Processing*

NAME

math – math functions and constants

SYNOPSIS

#include <math.h>

DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various functions in the C Library (Section 3C) that return floating-point values.

It defines the structure and constants used by the **matherr**(3M) error-handling mechanisms, including the following constant used as a error-return value:

HUGE The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E The base of natural logarithms (*e*).

M_LOG2E The base-2 logarithm of *e*.

M_LOG10E The base-10 logarithm of *e*.

M_LN2 The natural logarithm of 2.

M_LN10 The natural logarithm of 10.

M_PI π, the ratio of the circumference of a circle to its diameter.

M_PI_2 $\pi / 2$.

M_PI_4 $\pi / 4$.

M_1_PI $1 / \pi$.

M_2_PI $2 / \pi$.

M_2_SQRTPI $2 / \sqrt{\pi}$.

M_SQRT2 The positive square root of 2.

M_SQRT1_2 The positive square root of 1/2.

The following mathematical constants are also defined in this header file:

MAXFLOAT The maximum value of a non-infinite single-precision floating point number.

HUGE_VAL positive infinity.

For the definitions of various machine-dependent constants see **values**(5).

SEE ALSO

intro(3), **matherr**(3M), **values**(5)

NAME

me – macros for formatting papers

SYNOPSIS

nroff –me [*options*] *filename* . . .
troff –me [*options*] *filename* . . .

DESCRIPTION

This package of **nroff** and **troff** macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through **col**(1).

The macro requests are defined below. Many **nroff** and **troff** requests are unsafe in conjunction with this package, however, these requests may be used with impunity after the first .pp:

.bp begin new page
.br break output line here
.sp *n* insert n spacing lines
.ls *n* (line spacing) n=1 single, n=2 double space
.na no alignment of right margin
.ce *n* center next n lines
.ul *n* underline next n lines
.sz +*n* add n to point size

Output of the **eqn**(1), **neqn**(1), **refer**(1), and **tbl**(1) preprocessors for equations and tables is acceptable as input.

REQUESTS

In the following list, "initialization" refers to the first **.pp, .lp**, **.ip**, **.np**, **.sh**, or **.uh** macro. This list is incomplete.

Request	Initial Value	Cause Break	Explanation
.(c	-	yes	Begin centered block.
.(d	-	no	Begin delayed text.
.(f	-	no	Begin footnote.
.(l	-	yes	Begin list.
.(q	-	yes	Begin major quote.
.(x*x*	-	no	Begin indexed item in index *x*.
.(z	-	no	Begin floating keep.
.)c	-	yes	End centered block.
.)d	-	yes	End delayed text.
.)f	-	yes	End footnote.
.)l	-	yes	End list.
.)q	-	yes	End major quote.
.)x	-	yes	End index item.
.)z	-	yes	End floating keep.
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be **C** (chapter), **A** (appendix), **P** (preliminary, for instance,

			abstract, table of contents, etc.), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by **.++**). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by **eqn** or **neqn**.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.GE	-	yes	End *gremlin* picture.
.GS	-	yes	Begin *gremlin* picture.
.PE	-	yes	End **pic** picture.
.PS	-	yes	Begin **pic** picture.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba *+n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column.
.bi *x*	no	no	Print *x* in bold italics (nofill only).
.bu	-	yes	Begin bulleted paragraph.
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *´x´y´z*	´´´´´	no	Set even footer to *x y z*.
.eh *´x´y´z*	´´´´´	no	Set even header to *x y z*.
.fo *´x´y´z*	´´´´´	no	Set footer to *x y z*.
.hx	-	no	Suppress headers and footers on next page.
.he *´x´y´z*	´´´´´	no	Set header to *x y z*.

.hl	-	yes	Draw a horizontal line.
.i *x*	no	no	Italicize *x*; if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form **.***x*. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of ´*x*´*y*´*z*	´´´´´	no	Set odd footer to x y z.
.oh ´*x*´*y*´*z*	´´´´´	no	Set odd header to x y z.
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sm *x*	-	*no*	*Set x in a smaller pointsize.*
.sz +*n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in **troff**). (Nofill only).
.uh	-	yes	Like **.sh** but unnumbered.
.xp *x*	-	no	Print index *x*.

FILES

/usr/share/lib/tmac/e
/usr/share/lib/tmac/∗.me

SEE ALSO

eqn(1), **nroff**(1), **refer**(1), **tbl**(1), **troff**(1)

NAME

ms – text formatting macros

SYNOPSIS

nroff –ms [*options*] *filename* . . .

troff –ms [*options*] *filename* . . .

DESCRIPTION

This package of **nroff**(1) and **troff**(1) macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through **col**(1). All external **–ms** macros are defined below.

Note: this **–ms** macro package is an extended version written at Berkeley and is a superset of the standard **–ms** macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed; for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many **nroff** and **troff** requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

.bp	begin new page
.br	break output line
.sp *n*	insert n spacing lines
.ce *n*	center next n lines
.ls *n*	line spacing: *n*=**1** single, *n*=**2** double space
.na	no alignment of right margin

Font and point size changes with **\f** and **\s** are also allowed; for example, **\fIword\fR** will italicize *word*. Output of the **tbl**(1), **eqn**(1) and **refer**(1) preprocessors for equations, tables, and references is acceptable as input.

REQUESTS

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	–	y	begin abstract; if *x*=no do not label abstract
.AE	–	y	end abstract
.AI	–	y	author's institution
.AM	–	n	better accent mark definitions
.AU	–	y	author's name
.B *x*	–	n	embolden *x*; if no *x*, switch to boldface
.B1	–	y	begin text to be enclosed in a box
.B2	–	y	end boxed text and print it
.BT	date	n	bottom title, printed at foot of page
.BX *x*	–	n	print word *x* in a box
.CM	if t	n	cut mark between pages
.CT	–	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*
.DE	–	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x*=I, L, C, B; *y*=indent

.ID *y*	8n,.5i	y	indented display with no keep; *y*=indent
.LD	–	y	left display with no keep
.CD	–	y	centered display with no keep
.BD	–	y	block display; center entire block
.EF *x*	–	n	even page footer *x* (3 part as for **.tl**)
.EH *x*	–	n	even page header *x* (3 part as for **.tl**)
.EN	–	y	end displayed equation produced by **eqn**
.EQ *x y*	–	y	break out equation; *x*=L,I,C; *y*=equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y*=indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.
.MC *x*	–	y,y	multiple columns; *x*=column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for **.tl**)
.OH *x*	–	n	odd page header *x* (3 part as for **.tl**)
.P1	if TM	n	print header on first page
.PP	–	y,y	paragraph with first line indented
.PT	- % -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x*=no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x*=no stops title on first page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set TAB characters to 8n 16n ... (**nroff**) 5n 10n ... (**troff**)
.TC *x*	–	y	print table of contents at end; *x*=no suppresses title
.TE	–	y	end of table processed by **tbl**

.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.UL *x*	–	n	underline *x*, even in **troff**
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x*=page or no for none; y=indent
.XE	–	y	end index entry (or series of **.IX** entries)
.XP	–	y,y	paragraph with first line indented, others indented
.XS *x y*	–	y	begin index entry; *x*=page or no for none; *y*=indent
.1C	on	y,y	one column format, on a new page
.2C	–	y,y	begin two column format
.]–	–	n	beginning of **refer** reference
.[0	–	n	end of unclassifiable type of reference
.[N	–	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

REGISTERS

Formatting distances can be controlled in **–ms** by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr LL 6.5i
```

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as **LL**
FL	footnote length	next **.FS**	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next **.QP**	5n
FI	footnote indent	next **.FS**	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next **.FS**	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting **FF** to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an **.IP**-like footnote paragraph.

Here is a list of string registers available in **–ms**; they may be used anywhere in the text:

Name	String's Function
***Q**	quote (" in **nroff,** `` in **troff**)
***U**	unquote (" in **nroff,** '' in **troff**)
***–**	dash (**--** in **nroff,** **—** in **troff**)
***(MO**	month (month of the year)
***(DY**	day (current date)
******	automatically numbered footnote
***´**	acute accent (before letter)
***`**	grave accent (before letter)
***^**	circumflex (before letter)
***,**	cedilla (before letter)
***:**	umlaut (before letter)
***~**	tilde (before letter)

When using the extended accent mark definitions available with **.AM**, these strings should come after, rather than before, the letter to be accented.

FILES

/usr/share/lib/tmac/s
/usr/share/lib/tmac/ms.???

SEE ALSO

col(1), **eqn**(1), **nroff**(1), **refer**(1), **tbl**(1), **troff**(1)

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

NAME

nl_types – native language data types

SYNOPSIS

#include <nl_types.h>

DESCRIPTION

This header contains the following definitions:

nl_catd	Used by the message catalog functions **catopen**, **catgets** and **catclose** to identify a catalogue.
nl_item	Used by **nl_langinfo** to identify items of langinfo data. Values for objects of type **nl_item** are defined in **<langinfo.h>**.
NL_SETD	Used by **gencat** when no **$set** directive is specified in a message text source file. This constant can be used in subsequent calls to **catgets** as the value of the set identifier parameter.
NL_MGSMAX	Maximum number of messages per set.
NL_SETMAX	Maximum number of sets per catalogue.
NL_TEXTMAX	Maximum size of a message.

SEE ALSO

gencat(1), **catgets**(3C), **catopen**(3C), **nl_langinfo**(3C), **langinfo**(5)

NAME

prof – profile within a function

SYNOPSIS

```
#define MARK
#include <prof.h>
void MARK( name);
```

DESCRIPTION

MARK introduces a mark called *name* that is treated the same as a function entry point. Execution of the mark adds to a counter for that mark, and program-counter time spent is accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

name may be any combination of letters, numbers, or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol **MARK** must be defined before the header **prof.h** is included, either by a preprocessor directive as in the synopsis, or by a command line argument:

```
cc –p –DMARK work.c
```

If **MARK** is not defined, the **MARK(***name***)** statements may be left in the source files containing them and are ignored. **prof –g** must be used to get information on all labels.

EXAMPLE

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with **MARK** defined on the command line, the marks are ignored.

```
#include <prof.h>
work( )
{
        int i, j;
        . . .
        MARK(loop1);
        for (i = 0; i < 2000; i++) {
                . . .
        }
        MARK(loop2);
        for (j = 0; j < 2000; j++) {
                . . .
        }
}
```

SEE ALSO

profil(2), **monitor**(3C)

NAME

regexp, compile, step, advance – regular expression compile and match routines

SYNOPSIS

#define INIT *declarations*
#define GETC(void) *getc code*
#define PEEKC(void) *peekc code*
#define UNGETC(void) *ungetc code*
#define RETURN(*ptr***)** *return code*
#define ERROR(*val***)** *error code*

#include <regexp.h>

char **instring,* **char** **expbuf,* **char** **endbuf,* **int** *eof***);**

int step(char **string,* **char** **expbuf***);**

int advance(char **string,* **char** **expbuf***);**

extern char ***__loc1__**,** ***__loc2__**,** ***__locs__**;**

DESCRIPTION

These functions are general purpose regular expression matching routines to be used in programs that perform regular expression matching. These functions are defined by the **<regexp.h>** header.

The functions **step( )** and **advance( )** do pattern matching given a character string and a compiled regular expression as input.

The function **compile( )** takes as input a regular expression as defined below and produces a compiled expression that can be used with **step( )** or **advance( )**.

A regular expression specifies a set of character strings. A member of this set of strings is said to be matched by the regular expression. Some characters have special meaning when used in a regular expression; other characters stand for themselves.

The regular expressions available for use with the regexp functions are constructed as follows:

Expression	*Meaning*
c	the character *c* where *c* is not a special character.
c	the character *c* where *c* is any character, except a digit in the range **1**–**9**.
^	the beginning of the line being compared.
$	the end of the line being compared.
.	any character in the input.
[*s***]**	any character in the set *s*, where *s* is a sequence of characters and/or a range of characters, for example, **[***c*–*c***]**.
[^*s***]**	any character not in the set *s*, where *s* is defined as above.
*r**	zero or more successive occurrences of the regular expression *r*. The longest leftmost match is chosen.

rx the occurrence of regular expression *r* followed by the occurrence of regular expression *x*. (Concatenation)

r\{*m*,*n*\} any number of *m* through *n* successive occurrences of the regular expression *r*. The regular expression *r*\{*m*\} matches exactly *m* occurrences; *r*\{*m*,\} matches at least *m* occurrences.

\(*r*\) the regular expression *r*. When *n* (where *n* is a number greater than zero) appears in a constructed regular expression, it stands for the regular expression *x* where *x* is the n^{th} regular expression enclosed in \(and \) that appeared earlier in the constructed regular expression. For example, \(*r*\)*x*\(*y*\)*z*\2 is the concatenation of regular expressions *rxyzy*.

Characters that have special meaning except when they appear within square brackets ([ ]) or are preceded by \ are: ., *, [, \. Other special characters, such as $ have special meaning in more restricted contexts.

The character ^ at the beginning of an expression permits a successful match only immediately after a newline, and the character $ at the end of an expression requires a trailing newline.

Two characters have special meaning only when used within square brackets. The character – denotes a range, [*c*–*c*], unless it is just after the open bracket or before the closing bracket, [–*c*] or [*c*–] in which case it has no special meaning. When used within brackets, the character ^ has the meaning *complement of* if it immediately follows the open bracket (example: [^*c*]); elsewhere between brackets (example: [*c*^]) it stands for the ordinary character ^.

The special meaning of the \ operator can be escaped only by preceding it with another \, for example \\.

Programs must have the following five macros declared before the **#include <regexp.h>** statement. These macros are used by the **compile()** routine. The macros **GETC**, **PEEKC**, and **UNGETC** operate on the regular expression given as input to **compile()**.

GETC This macro returns the value of the next character (byte) in the regular expression pattern. Successive calls to **GETC** should return successive characters of the regular expression.

PEEKC This macro returns the next character (byte) in the regular expression. Immediately successive calls to **PEEKC** should return the same character, which should also be the next character returned by **GETC**.

UNGETC This macro causes the argument **c** to be returned by the next call to **GETC** and **PEEKC**. No more than one character of pushback is ever needed and this character is guaranteed to be the last character read by **GETC**. The return value of the macro **UNGETC(c)** is always ignored.

RETURN(*ptr***)** This macro is used on normal exit of the **compile( )** routine. The value of the argument *ptr* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.

ERROR(*val***)** This macro is the abnormal return from the **compile( )** routine. The argument *val* is an error number (see ERRORS below for meanings). This call should never return.

The syntax of the **compile( )** routine is as follows:

compile(*instring***,** *expbuf***,** *endbuf***,** *eof***)**

The first parameter, *instring*, is never used explicitly by the **compile( )** routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the **INIT** declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of **(char** ∗**)0** for this parameter.

The next parameter, *expbuf*, is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in **(endbuf**–**expbuf)** bytes, a call to **ERROR(50)** is made.

The parameter *eof* is the character which marks the end of the regular expression. This character is usually a **/**.

Each program that includes the **<regexp.h>** header file must have a **#define** statement for **INIT**. It is used for dependent declarations and initializations. Most often it is used to set a register variable to point to the beginning of the regular expression so that this register variable can be used in the declarations for **GETC**, **PEEKC**, and **UNGETC**. Otherwise it can be used to declare external variables that might be used by **GETC**, **PEEKC** and **UNGETC**. (See EXAMPLE below.)

The first parameter to the **step( )** and **advance( )** functions is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter, *expbuf*, is the compiled regular expression which was obtained by a call to the function **compile( )**.

The function **step( )** returns non-zero if some substring of *string* matches the regular expression in *expbuf* and zero if there is no match. If there is a match, two external character pointers are set as a side effect to the call to **step( )**. The variable **loc1** points 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 input string, **loc1** will point to the first character of *string* and **loc2** will point to the null at the end of *string*.

The function **advance( )** returns non-zero if the initial substring of *string* matches the regular expression in *expbuf*. If there is a match, an external character pointer, **loc2**, is set as a side effect. The variable **loc2** points to the next character in *string* after the last character that matched.

When **advance()** encounters a * or **\{ \}** sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, **advance()** will back up along the string until it finds a match or reaches the point in the string that initially matched the * or **\{ \}**. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer **locs** is equal to the point in the string at sometime during the backing up process, **advance()** will break out of the loop that backs up and will return zero.

The external variables **circf**, **sed**, and **nbra** are reserved.

EXAMPLE

The following is an example of how the regular expression macros and calls might be defined by an application program:

```
#define INIT       register char *sp = instring;
#define GETC       (*sp++)
#define PEEKC      (*sp)
#define UNGETC(c)  (--sp)
#define RETURN(*c)  return;
#define ERROR(c)   regerr

#include <regexp.h>
...
    (void) compile(*argv, expbuf, &expbuf[ESIZE],'\0');
...
    if (step(linebuf, expbuf))
            succeed;
```

DIAGNOSTICS

The function **compile()** uses the macro **RETURN** on success and the macro **ERROR** on failure (see above). The functions **step()** and **advance()** return non-zero on a successful match and zero if there is no match. Errors are:

11 range endpoint too large.

16 bad number.

25 \ *digit* out of range.

36 illegal or missing delimiter.

41 no remembered search string.

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.

NAME

siginfo – signal generation information

SYNOPSIS

```
#include <siginfo.h>
```

DESCRIPTION

If a process is catching a signal, it may request information that tells why the system generated that signal (see **sigaction**(2)). If a process is monitoring its children, it may receive information that tells why a child changed state (see **waitid**(2)). In either case, the system returns the information in a structure of type **siginfo_t**, which includes the following information:

```
int           si_signo   /* signal number */
int           si_errno   /* error number */
int           si_code    /* signal code */
union sigval  si_value   /* signal value */
```

si_signo contains the system-generated signal number. For the **waitid**(2) function, **si_signo** is always **SIGCHLD**.

If **si_errno** is non-zero, it contains an error number associated with this signal, as defined in **<errno.h>**.

si_code contains a code identifying the cause of the signal.

If the value of the **si_code** member is **SI_NOINFO**, only the **si_signo** member of **siginfo_t** is meaningful, and the value of all other members is unspecified.

User Signals

If the value of **si_code** is less than or equal to 0, then the signal was generated by a user process (see **kill**(2), **_lwp_kill**(2), **sigsend**(2), **abort**(3C), and **raise**(3C)) and the **siginfo** structure contains the following additional information:

```
typedef long pid_t   si_pid   /* sending process ID */
typedef long uid_t   si_uid   /* sending user ID */
```

If the signal was generated by a user process, the following values are defined for **si_code**:

SI_USER	the implementation sets **si_code** to **SI_USER** if the signal was sent by **kill**(2), **sigsend**(2), **raise**(3C) or **abort**(3C).
SI_LWP	the signal was sent by **_lwp_kill**(2).
SI_QUEUE	the signal was sent by
SI_TIMER	the signal was generated by the expiration of a timer set by
SI_ASYNCIO	the signal was generated by the completion of an asynchronous I/O request.
SI_MESGQ	the signal was generated by the arrival of a message on an empty message queue. (see **mq_notify**(3R)).

si_value contains the application specified value, which is passed to the application's signal-catching function at the time of the signal delivery, if **si_code** is any of **SI_QUEUE**, **SI_TIMER**, **SI_ASYNCHIO**, or **SI_MESGQ**.

System Signals

Otherwise, **si_code** contains a positive value reflecting the reason why the system generated the signal:

Signal	Code	Reason
SIGILL	**ILL_ILLOPC**	illegal opcode
	ILL_ILLOPN	illegal operand
	ILL_ILLADR	illegal addressing mode
	ILL_ILLTRP	illegal trap
	ILL_PRVOPC	privileged opcode
	ILL_PRVREG	privileged register
	ILL_COPROC	co-processor error
	ILL_BADSTK	internal stack error
SIGFPE	**FPE_INTDIV**	integer divide by zero
	FPE_INTOVF	integer overflow
	FPE_FLTDIV	floating point divide by zero
	FPE_FLTOVF	floating point overflow
	FPE_FLTUND	floating point underflow
	FPE_FLTRES	floating point inexact result
	FPE_FLTINV	invalid floating point operation
	FPE_FLTSUB	subscript out of range
SIGSEGV	**SEGV_MAPERR**	address not mapped to object
	SEGV_ACCERR	invalid permissions for mapped object
SIGBUS	**BUS_ADRALN**	invalid address alignment
	BUS_ADRERR	non-existent physical address
	BUS_OBJERR	object specific hardware error
SIGTRAP	**TRAP_BRKPT**	process breakpoint
	TRAP_TRACE	process trace trap
SIGCHLD	**CLD_EXITED**	child has exited
	CLD_KILLED	child was killed
	CLD_DUMPED	child terminated abnormally
	CLD_TRAPPED	traced child has trapped
	CLD_STOPPED	child has stopped
	CLD_CONTINUED	stopped child had continued
SIGPOLL	**POLL_IN**	data input available
	POLL_OUT	output buffers available
	POLL_MSG	input message available
	POLL_ERR	I/O error
	POLL_PRI	high priority input available
	POLL_HUP	device disconnected

In addition, the following signal-dependent information is available for kernel-generated signals:

Signal	Field	Value
SIGILL **SIGFPE**	**caddr_t si_addr**	address of faulting instruction
SIGSEGV **SIGBUS**	**caddr_t si_addr**	address of faulting memory reference
SIGCHLD	**pid_t si_pid** **int si_status**	child process ID exit value or signal
SIGPOLL	**long si_band**	band event for **POLL_IN**, **POLL_OUT**, or **POLL_MSG**

SEE ALSO

_lwp_kill(2), **kill**(2), **sigaction**(2), **sigsend**(2), **waitid**(2), **abort**(3C), **raise**(3C), **aio_read**(3R), **mq_notify**(3R), **sigqueue**(3R), **timer_create**(3R), **signal**(5)

NOTES

For **SIGCHLD** signals, if **si_code** is equal to **CLD_EXITED**, then **si_status** is equal to the exit value of the process; otherwise, it is equal to the signal that caused the process to change state. For some implementations, the exact value of **si_addr** may not be available; in that case, **si_addr** is guaranteed to be on the same page as the faulting instruction or memory reference.

NAME

signal – base signals

SYNOPSIS

#include <signal.h>

DESCRIPTION

A signal is an asynchronous notification of an event. A signal is said to be generated for (or sent to) a process when the event associated with that signal first occurs. Examples of such events include hardware faults, timer expiration and terminal activity, as well as the invocation of the **kill**(2) or **sigsend**(2) system calls. In some circumstances, the same event generates signals for multiple processes. A process may request a detailed notification of the source of the signal and the reason why it was generated (see **siginfo**(5)).

A process responds to signals in similar ways whether it is using threads (see **thr_create**(3T)) or it is using lightweight processes (LWPs). Each process may specify a system action to be taken in response to each signal sent to it, called the signal's disposition. All threads or LWPs in the process share the disposition. The set of system signal actions for a process is initialized from that of its parent. Once an action is installed for a specific signal, it usually remains installed until another disposition is explicitly requested by a call to either **sigaction, signal** or **sigset,** or until the process **execs** (see **sigaction**(2) and **signal**(3C)). When a process execs, all signals whose disposition has been set to catch the signal will be set to **SIG_DFL**. Alternatively, a process may request that the system automatically reset the disposition of a signal to **SIG_DFL** after it has been caught (see **sigaction**(2) and **signal**(3C)).

A signal is said to be delivered to a process when a thread or LWP within the process takes the appropriate action for the disposition and signal. Delivery of a signal can be blocked. Each thread or LWP has a signal mask (see **thr_sigsetmask**(3T) or **sigprocmask**(2)) that defines the set of signals currently blocked from delivery to it. The signal mask of the main thread or LWP is inherited from the signal mask of the thread or LWP that created it in the parent process. The selection of the thread or LWP within the process that is to take the appropriate action for the signal is based on the method of signal generation and the signal masks of the threads or LWPs in the receiving process. Signals that are a generated by action of a particular thread or LWP such as hardware faults or alarms (see **alarm**(2)), are delivered to the thread or LWP that caused the signal. Signals that are directed to a particular thread or LWP (see **thr_kill**(3T) or **_lwp_kill**(2)) are delivered to the targeted thread or LWP. If the selected thread or LWP has blocked the signal, it remains pending on the thread or LWP until it is unblocked. For all other types of signal generation (e.g. **kill**(2), **sigsend**(2), terminal activity, and other external events not ascribable to a particular thread or LWP) one of the threads or LWPs that does not have the signal blocked is selected to process the signal. If all the threads or LWPs within the process block the signal, it remains pending on the process until a thread or LWP in the process unblocks it. If the action associated with a signal is set to ignore the signal then both currently pending and subsequently generated signals of this type are discarded immediately for this process.

The determination of which action is taken in response to a signal is made at the time the signal is delivered to a thread or LWP within the process, allowing for any changes since the time of generation. This determination is independent of the means by which the signal was originally generated.

The signals currently defined by **<signal.h>** are as follows:

Name	Value	Default	Event
SIGHUP	1	Exit	Hangup (see **termio**(7))
SIGINT	2	Exit	Interrupt (see **termio**(7))
SIGQUIT	3	Core	Quit (see **termio**(7))
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace/Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status Changed
SIGPWR	19	Ignore	Power Fail/Restart
SIGWINCH	20	Ignore	Window Size Change
SIGURG	21	Ignore	Urgent Socket Condition
SIGPOLL	22	Exit	Pollable Event (see **streamio**(7))
SIGSTOP	23	Stop	Stopped (signal)
SIGTSTP	24	Stop	Stopped (user) (see **termio**(7))
SIGCONT	25	Ignore	Continued
SIGTTIN	26	Stop	Stopped (tty input) (see **termio**(7))
SIGTTOU	27	Stop	Stopped (tty output) (see **termio**(7))
SIGVTALRM	28	Exit	Virtual Timer Expired
SIGPROF	29	Exit	Profiling Timer Expired
SIGXCPU	30	Core	CPU time limit exceeded (see **getrlimit**(2))
SIGXFSZ	31	Core	File size limit exceeded (see **getrlimit**(2))
SIGWAITING	32	Ignore	Process's LWPs are blocked
SIGLWP	33	Ignore	Special signal used by thread library
SIGFREEZE	34	Ignore	Check point Freeze
SIGTHAW	35	Ignore	Check point Thaw
SIGRTMIN	*	Exit	First real time signal
(SIGRTMIN + 1)	*	Exit	Second real time signal
...			
(SIGRTMAX - 1)	*	Exit	Second-to-last real time signal
SIGRTMAX	*	Exit	Last real time signal

(The symbols **SIGRTMIN** through **SIGRTMAX** are evaluated dynamically in order to permit future configurability)

A process, using a **signal**(3C), **sigset**(3C) or **sigaction**(2) system call, may specify one of three dispositions for a signal: take the default action for the signal, ignore the signal, or catch the signal.

Default Action: SIG_DFL

A disposition of **SIG_DFL** specifies the default action. The default action for each signal is listed in the table above and is selected from the following:

Exit — When it gets the signal, the receiving process is to be terminated with all the consequences outlined in **exit**(2).

Core — When it gets the signal, the receiving process is to be terminated with all the consequences outlined in **exit**(2). In addition, a ''core image'' of the process is constructed in the current working directory.

Stop — When it gets the signal, the receiving process is to stop. When a process is stopped, all the threads and LWPs within the process also stop executing.

Ignore — When it gets the signal, the receiving process is to ignore it. This is identical to setting the disposition to **SIG_IGN**.

Ignore Signal: SIG_IGN

A disposition of **SIG_IGN** specifies that the signal is to be ignored. Setting a signal action to **SIG_IGN** for a signal that is pending causes the pending signal to be discarded, whether or not it is blocked. Any queued values pending are also discarded, and the resources used to queue them are released and made available to queue other signals.

Catch Signal: *function address*

A disposition that is a function address specifies that, when it gets the signal, the thread or LWP within the process that is selected to process the signal will execute the signal handler at the specified address. Normally, the signal handler is passed the signal number as its only argument; if the disposition was set with the **sigaction** function however, additional arguments may be requested (see **sigaction**(2)). When the signal handler returns, the receiving process resumes execution at the point it was interrupted, unless the signal handler makes other arrangements. If an invalid function address is specified, results are undefined.

If the disposition has been set with the **sigset** or **sigaction** function, the signal is automatically blocked in the thread or LWP while it is executing the signal catcher. If a **longjmp** (see **setjmp**(3C)) is used to leave the signal catcher, then the signal must be explicitly unblocked by the user (see **signal**(3C) and **sigprocmask**(2)).

If execution of the signal handler interrupts a blocked system call, the handler is executed and the interrupted system call returns a –1 to the calling process with **errno** set to **EINTR**. However, if the **SA_RESTART** flag is set the system call will be transparently restarted.

Some signal–generating functions, such as high resolution timer expiration, asynchronous I/O completion, inter-process message arrival, and the **sigqueue**(3R) function, support the specification of an application defined value, either explicitly as a parameter to the function, or in a **sigevent** structure parameter. The **sigevent** structure is defined by **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	**sigev_notify**	Notification type
int	**sigev_signo**	Signal number
union sigval	**sigev_value**	Signal value

The **sigval** union is defined by **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	**sival_int**	Integer signal value
void *	**sival_ptr**	Pointer signal value

sigev_notify specifies the notification mechanism to use when an asynchronous event occurs. **sigev_notify** may be defined with the following values:

SIGEV_NONE No asynchronous notification is delivered when the event of interest occurs.

SIGEV_SIGNAL A queued signal, with its value application-defined, is generated when the event of interest occurs.

Your implementation may define additional notification mechanisms.

sigev_signo specifies the signal to be generated.

sigev_value references the application defined value to be passed to the signal-catching function at the time of the signal delivery as the **si_value** member of the **siginfo_t** structure.

The **sival_int** member will be used when the application defined value is of type **int**; and the **sival_ptr** member will be used when the application defined value is a pointer.

When a signal is generated by **sigqueue**(3R) or any signal–generating function which supports the specification of an application defined value, the signal is marked pending and, if the **SA_SIGINFO** flag is set for that signal, the signal is queued to the process along with the application specified signal value. Multiple occurrences of signals so generated are queued in FIFO order. If the **SA_SIGINFO** flag is not set for that signal, later occurrences of that signal's generation, when a signal is already queued, are silently discarded.

NOTES

The dispositions of the **SIGKILL** and **SIGSTOP** signals cannot be altered from their default values. The system generates an error if this is attempted.

The **SIGKILL** and **SIGSTOP** signals cannot be blocked. The system silently enforces this restriction.

Whenever a process receives a **SIGSTOP**, **SIGTSTP**, **SIGTTIN**, or **SIGTTOU** signal, regardless of its disposition, any pending **SIGCONT** signal are discarded.

Whenever a process receives a **SIGCONT** signal, regardless of its disposition, any pending **SIGSTOP**, **SIGTSTP**, **SIGTTIN**, and **SIGTTOU** signals is discarded. In addition, if the process was stopped, it is continued.

SIGPOLL is issued when a file descriptor corresponding to a STREAMS (see **intro**(2)) file has a ''selectable'' event pending. A process must specifically request that this signal be sent using the **I_SETSIG ioctl** call. Otherwise, the process will never receive **SIGPOLL**.

If the disposition of the **SIGCHLD** signal has been set with **signal** or **sigset**, or with **sigaction** and the **SA_NOCLDSTOP** flag has been specified, it will only be sent to the calling process when its children exit; otherwise, it will also be sent when the calling process's children are stopped or continued due to job control.

The name **SIGCLD** is also defined in this header and identifies the same signal as **SIGCHLD**. **SIGCLD** is provided for backward compatibility, new applications should use **SIGCHLD**.

The disposition of signals that are inherited as **SIG_IGN** should not be changed.

SEE ALSO

intro(2), **exit**(2), **getrlimit**(2), **kill**(2), **pause**(2), **sigaction**(2), **sigaltstack**(2), **sigprocmask**(2), **sigsend**(2), **sigsuspend**(2), **wait**(2), **signal**(3C), **sigsetops**(3C), **sigqueue**(3R), **siginfo**(5), **ucontext**(5)

NAME

stat – data returned by stat system call

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
```

DESCRIPTION

The system calls **stat**, **lstat** and **fstat** return data in a **stat** structure, which is defined in **stat.h**.

The constants used in the **st_mode** field are also defined in this file:

```
#define   S_IFMT    /* type of file */
#define   S_IAMB    /* access mode bits */
#define   S_IFIFO   /* fifo */
#define   S_IFCHR   /* character special */
#define   S_IFDIR   /* directory */
#define   S_IFNAM   /* XENIX special named file */
#define   S_INSEM   /* XENIX semaphore subtype of IFNAM */
#define   S_INSHD   /* XENIX shared data subtype of IFNAM */
#define   S_IFBLK   /* block special */
#define   S_IFREG   /* regular */
#define   S_IFLNK   /* symbolic link */
#define   S_ISUID   /* set user id on execution */
#define   S_ISGID   /* set group id on execution */
#define   S_ISVTX   /* save swapped text even after use */
#define   S_IREAD   /* read permission, owner */
#define   S_IWRITE  /* write permission, owner */
#define   S_IEXEC   /* execute/search permission, owner */
#define   S_ENFMT   /* record locking enforcement flag */
#define   S_IRWXU   /* read, write, execute: owner */
#define   S_IRUSR   /* read permission: owner */
#define   S_IWUSR   /* write permission: owner */
#define   S_IXUSR   /* execute permission: owner */
#define   S_IRWXG   /* read, write, execute: group */
#define   S_IRGRP   /* read permission: group */
#define   S_IWGRP   /* write permission: group */
#define   S_IXGRP   /* execute permission: group */
#define   S_IRWXO   /* read, write, execute: other */
#define   S_IROTH   /* read permission: other */
#define   S_IWOTH   /* write permission: other */
#define   S_IXOTH   /* execute permission: other */
```

The following macros are for POSIX conformance:

```
#define S_ISBLK(mode)    block special file
#define S_ISCHR(mode)    character special file
#define S_ISDIR(mode)    directory file
#define S_ISFIFO(mode)   pipe or fifo file
#define S_ISREG(mode)    regular file
```

SEE ALSO **stat**(2), **types**(5)

NAME

stdarg – handle variable argument list

SYNOPSIS

```
#include <stdarg.h>

va_list pvar;

void va_start(va_list pvar, parmN);

type va_arg(va_list pvar, type);

void va_end(va_list pvar);
```

DESCRIPTION

This set of macros allows portable procedures that accept variable numbers of arguments of variable types to be written. Routines that have variable argument lists (such as **printf**) but do not use *stdarg* are inherently non-portable, as different machines use different argument-passing conventions.

va_list is a type defined for the variable used to traverse the list.

The **va_start()** macro is invoked before any access to the unnamed arguments and initializes **pvar** for subsequent use by **va_arg()** and **va_end()**. The parameter *parmN* is the identifier of the rightmost parameter in the variable parameter list in the function definition (the one just before the **, ...**). If this parameter is declared with the **register** storage class or with a function or array type, or with a type that is not compatible with the type that results after application of the default argument promotions, the behavior is undefined.

The parameter *parmN* is required under strict ANSI C compilation. In other compilation modes, *parmN* need not be supplied and the second parameter to the **va_start()** macro can be left empty (for example, **va_start(pvar,);**). This allows for routines that contain no parameters before the **...** in the variable parameter list.

The **va_arg()** macro expands to an expression that has the type and value of the next argument in the call. The parameter **pvar** should have been previously initialized by **va_start()**. Each invocation of **va_arg()** modifies **pvar** so that the values of successive arguments are returned in turn. The parameter *type* is the type name of the next argument to be returned. The type name must be specified in such a way so that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a * to *type*. If there is no actual next argument, or if *type* is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined.

The **va_end()** macro is used to clean up.

Multiple traversals, each bracketed by **va_start** and **va_end**, are possible.

EXAMPLE

This example gathers into an array a list of arguments that are pointers to strings (but not more than **MAXARGS** arguments) with function **f1**, then passes the array as a single argument to function **f2**. The number of pointers is specified by the first argument to **f1**.

```
#include <stdarg.h>
#define MAXARGS      31

void f1(int n_ptrs, ...)
{
        va_list ap;
        char *array[MAXARGS];
        int ptr_no = 0;

        if (n_ptrs > MAXARGS)
                n_ptrs = MAXARGS;
        va_start(ap, n_ptrs);
        while (ptr_no < n_ptrs)
                array[ptr_no++] = va_arg(ap, char*);
        va_end(ap);
        f2(n_ptrs, array);
}
```

Each call to **f1** shall have visible the definition of the function or a declaration such as

```
void f1(int, ...)
```

SEE ALSO

vprintf(3S)

NOTES

It is up to the calling routine to specify in some manner how many arguments there are, since it is not always possible to determine the number of arguments from the stack frame. For example, **execl** is passed a zero pointer to signal the end of the list. **printf** can tell how many arguments there are by the format. It is non-portable to specify a second argument of **char**, **short**, or **float** to **va_arg**, because arguments seen by the called function are not **char**, **short**, or **float**. C converts **char** and **short** arguments to **int** and converts **float** arguments to **double** before passing them to a function.

NAME

term – conventional names for terminals

DESCRIPTION

Terminal names are maintained as part of the shell environment in the environment variable **TERM** (see **sh**(1), **profile**(4), and **environ**(5)). These names are used by certain commands (for example, **tabs**, **tput**, and **vi**) and certain functions (for example, see **curses**(3X)).

Files under **/usr/share/lib/terminfo** are used to name terminals and describe their capabilities. These files are in the format described in **terminfo**(4). Entries in **terminfo** source files consist of a number of comma-separated fields. To print a description of a terminal *term*, use the command **infocmp –I** *term* (see **infocmp**(1M)). White space after each comma is ignored. The first line of each terminal description in the **terminfo** database gives the names by which **terminfo** knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal (this is the one to use to set the environment variable **TERMINFO** in **$HOME/.profile**; see **profile**(4)), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, for the AT&T 4425 terminal, **att4425**. This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Up to 8 characters, chosen from the set **a** through **z** and **0** through **9**, make up a basic terminal name. Names should generally be based on original vendors rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name. Terminal sub-models, operational modes that the hardware can be in, or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, an AT&T 4425 terminal in 132 column mode is **att4425–w**. The following suffixes should be used where possible:

Suffix	Meaning	Example
–w	Wide mode (more than 80 columns)	**att4425–w**
–am	With auto. margins (usually default)	**vt100–am**
–nam	Without automatic margins	**vt100–nam**
–n	Number of lines on the screen	**aaa–60**
–na	No arrow keys (leave them in local)	**c100–na**
–np	Number of pages of memory	**c100–4p**
–rv	Reverse video	**att4415–rv**

To avoid conflicts with the naming conventions used in describing the different modes of a terminal (for example, **–w**), it is recommended that a terminal's root name not contain hyphens. Further, it is good practice to make all terminal names used in the **terminfo**(4) database unique. Terminal entries that are present only for inclusion in other entries via the **use=** facilities should have a '**+**' in their name, as in **4415+nl**.

Here are some of the known terminal names: (For a complete list, enter the command **ls -C /usr/share/lib/terminfo/?**).

2621,hp2621	Hewlett-Packard 2621 series
2631	Hewlett-Packard 2631 line printer
2631–c	Hewlett-Packard 2631 line printer, compressed mode
2631–e	Hewlett-Packard 2631 line printer, expanded mode
2640,hp2640	Hewlett-Packard 2640 series
2645,hp2645	Hewlett-Packard 2645 series
3270	IBM Model 3270
33,tty33	AT&T Teletype Model 33 KSR
35,tty35	AT&T Teletype Model 35 KSR
37,tty37	AT&T Teletype Model 37 KSR
4000a	Trendata 4000a
4014,tek4014	TEKTRONIX 4014
40,tty40	AT&T Teletype Dataspeed 40/2
43,tty43	AT&T Teletype Model 43 KSR
4410,5410	AT&T 4410/5410 in 80-column mode, version 2
4410–nfk,5410–nfk	AT&T 4410/5410 without function keys, version 1
4410–nsl,5410–nsl	AT&T 4410/5410 without pln defined
4410–w,5410–w	AT&T 4410/5410 in 132-column mode
4410v1,5410v1	AT&T 4410/5410 in 80-column mode, version 1
4410v1–w,5410v1–w	AT&T 4410/5410 in 132-column mode, version 1
4415,5420	AT&T 4415/5420 in 80-column mode
4415–nl,5420–nl	AT&T 4415/5420 without changing labels
4415–rv,5420–rv	AT&T 4415/5420 80 columns in reverse video
4415–rv–nl,5420–rv–nl	AT&T 4415/5420 reverse video without changing labels
4415–w,5420–w	AT&T 4415/5420 in 132-column mode
4415–w–nl,5420–w–nl	AT&T 4415/5420 in 132-column mode without changing labels
4415–w–rv,5420–w–rv	AT&T 4415/5420 132 columns in reverse video
4418,5418	AT&T 5418 in 80-column mode
4418–w,5418–w	AT&T 5418 in 132-column mode
4420	AT&T Teletype Model 4420
4424	AT&T Teletype Model 4424
4424-2	AT&T Teletype Model 4424 in display function group ii
4425,5425	AT&T 4425/5425
4425–fk,5425–fk	AT&T 4425/5425 without function keys

4425–nl,5425–nl	AT&T 4425/5425 without changing labels in 80-column mode
4425–w,5425–w	AT&T 4425/5425 in 132-column mode
4425–w–fk,5425–w–fk	AT&T 4425/5425 without function keys in 132-column mode
4425–nl–w,5425–nl–w	AT&T 4425/5425 without changing labels in 132-column mode
4426	AT&T Teletype Model 4426S
450	DASI 450 (same as Diablo 1620)
450–12	DASI 450 in 12-pitch mode
500,att500	AT&T-IS 500 terminal
510,510a	AT&T 510/510a in 80-column mode
513bct,att513	AT&T 513 bct terminal
5320	AT&T 5320 hardcopy terminal
5420_2	AT&T 5420 model 2 in 80-column mode
5420_2–w	AT&T 5420 model 2 in 132-column mode
5620,dmd	AT&T 5620 terminal 88 columns
5620–24,dmd–24	AT&T Teletype Model DMD 5620 in a 24x80 layer
5620–34,dmd–34	AT&T Teletype Model DMD 5620 in a 34x80 layer
610,610bct	AT&T 610 bct terminal in 80-column mode
610–w,610bct–w	AT&T 610 bct terminal in 132-column mode
630,630MTG	AT&T 630 Multi-Tasking Graphics terminal
7300,pc7300,unix_pc	AT&T UNIX PC Model 7300
735,ti	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
pt505	AT&T Personal Terminal 505 (22 lines)
pt505–24	AT&T Personal Terminal 505 (24-line mode)
sync	generic name for synchronous Teletype Model 4540-compatible terminals

Commands whose behavior depends on the type of terminal should accept arguments of the form **–T***term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **TERM**, which, in turn, should contain *term*.

FILES

/usr/share/lib/terminfo/?/*
compiled terminal description database

SEE ALSO

sh(1), **stty**(1), **tabs**(1), **tput**(1), **vi**(1), **infocmp**(1M), **curses**(3X), **profile**(4), **terminfo**(4), **environ**(5)

NAME

types – primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in **types.h** are used in UNIX System code. Some data of these types are accessible to user code:

```
typedef  struct { int r[1]; } *physadr;
typedef  long               clock_t;
typedef  long               daddr_t;
typedef  char *             caddr_t;
typedef  unsigned char      unchar;
typedef  unsigned short     ushort;
typedef  unsigned int       uint;
typedef  unsigned long      ulong;
typedef  unsigned long      ino_t;
typedef  long               uid_t;
typedef  long               gid_t;
typedef  ulong              nlink_t;
typedef  ulong              mode_t;
typedef  short              cnt_t;
typedef  long               time_t;
typedef  int                label_t[10];
typedef  ulong              dev_t;
typedef  long               off_t;
typedef  long               pid_t;
typedef  long               paddr_t;
typedef  int                key_t;
typedef  unsigned char      use_t;
typedef  short              sysid_t;
typedef  short              index_t;
typedef  short              lock_t;
typedef  unsigned int       size_t;
typedef  long               clock_t;
typedef  long               pid_t;
```

The form **daddr_t** is used for disk addresses except in an inode on disk. Times are encoded in seconds since 00:00:00 UTC, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The **label_t** variables are used to save the processor state while another process is running.

NAME

ucontext – user context

SYNOPSIS

```
#include <ucontext.h>
```

DESCRIPTION

The **ucontext** structure defines the context of a thread of control within an executing process.

This structure includes at least the following members:

```
ucontext_t  uc_link
sigset_t    uc_sigmask
stack_t     uc_stack
mcontext_t  uc_mcontext
```

uc_link is a pointer to the context that to be resumed when this context returns. If **uc_link** is equal to 0, then this context is the main context, and the process exits when this context returns.

uc_sigmask defines the set of signals that are blocked when this context is active [see **sigprocmask**(2)].

uc_stack defines the stack used by this context [see **sigaltstack**(2)].

uc_mcontext contains the saved set of machine registers and any implementation specific context data. Portable applications should not modify or access **uc_mcontext**.

SEE ALSO

getcontext(2), **sigaction**(2), **sigaltstack**(2), **sigprocmask**(2), **makecontext**(3C)

NAME

values – machine-dependent values

SYNOPSIS

#include <values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS(*type***)** The number of bits in a specified type (for example, **int**).

HIBITS The value of a short integer with only the high-order bit set.

HIBITL The value of a long integer with only the high-order bit set.

HIBITI The value of a regular integer with only the high-order bit set.

MAXSHORT The maximum value of a signed short integer.

MAXLONG The maximum value of a signed long integer.

MAXINT The maximum value of a signed regular integer.

MAXFLOAT, **LN_MAXFLOAT**
The maximum value of a single-precision floating-point number, and its natural logarithm.

MAXDOUBLE, **LN_MAXDOUBLE**
The maximum value of a double-precision floating-point number, and its natural logarithm.

MINFLOAT, **LN_MINFLOAT**
The minimum positive value of a single-precision floating-point number, and its natural logarithm.

MINDOUBLE, **LN_MINDOUBLE**
The minimum positive value of a double-precision floating-point number, and its natural logarithm.

FSIGNIF The number of significant bits in the mantissa of a single-precision floating-point number.

DSIGNIF The number of significant bits in the mantissa of a double-precision floating-point number.

SEE ALSO

intro(3)**, math**(5)

NAME

varargs – handle variable argument list

SYNOPSIS

#include <varargs.h>

va_alist

va_dcl

va_list pvar;

void va_start(va_list pvar);

type **va_arg(va_list pvar,** *type***);**

void va_end(va_list pvar);

DESCRIPTION

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists (such as **printf**(3S)) but do not use **varargs** are inherently non-portable, as different machines use different argument-passing conventions.

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for **va_alist**. No semicolon should follow **va_dcl**.

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize **pvar** to the beginning of the list.

va_arg will return the next argument in the list pointed to by **pvar**. *type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by **va_start** and **va_end**, are possible.

EXAMPLE

This example is a possible implementation of **execl** (see **exec**(2)).

```
#include <unistd.h>
#include <varargs.h>
#define MAXARGS      100

/*      execl is called by
                execl(file, arg1, arg2, ..., (char *)0);
*/
execl(va_alist)
va_dcl
{
        va_list ap;
        char *file;
        char *args[MAXARGS];                    /* assumed big enough*/
        int argno = 0;

        va_start(ap);
```

```
        file = va_arg(ap, char *);
        while ((args[argno++] = va_arg(ap, char *)) != 0)
                ;
        va_end(ap);
        return execv(file, args);
}
```

SEE ALSO

exec(2), **printf**(3S), **vprintf**(3S), **stdarg**(5)

NOTES

It is up to the calling routine to specify in some manner how many arguments there are, since it is not always possible to determine the number of arguments from the stack frame. For example, **execl** is passed a zero pointer to signal the end of the list. **printf** can tell how many arguments are there by the format.

It is non-portable to specify a second argument of **char**, **short**, or **float** to **va_arg**, since arguments seen by the called function are not **char**, **short**, or **float**. C converts **char** and **short** arguments to **int** and converts **float** arguments to **double** before passing them to a function.

stdarg is the preferred interface.

NAME

wstat – wait status

SYNOPSIS

#include <sys/wait.h>

DESCRIPTION

When a process waits for status from its children via either the **wait** or **waitpid** function, the status returned may be evaluated with the following macros, defined in **<sys/wait.h>**. These macros evaluate to integral expressions. The *stat* argument to these macros is the integer value returned from **wait** or **waitpid**.

WIFEXITED(*stat***)** Evaluates to a non-zero value if status was returned for a child process that terminated normally.

WEXITSTATUS(*stat***)** If the value of **WIFEXITED(***stat***)** is non-zero, this macro evaluates to the exit code that the child process passed to **_exit( )** (see **exit**(2)) or **exit**(3C), or the value that the child process returned from **main**.

WIFSIGNALED(*stat***)** Evaluates to a non-zero value if status was returned for a child process that terminated due to the receipt of a signal.

WTERMSIG(*stat***)** If the value of **WIFSIGNALED(***stat***)** is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.

WIFSTOPPED(*stat***)** Evaluates to a non-zero value if status was returned for a child process that is currently stopped.

WSTOPSIG(*stat***)** If the value of **WIFSTOPPED(***stat***)** is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.

WIFCONTINUED(*stat*) Evaluates to a non-zero value if status was returned for a child process that has continued.

WCOREDUMP(*stat*) If the value of **WIFSIGNALED** (*stat*) is non-zero, this macro evaluates to a non-zero value if a core image of the terminated child was created.

SEE ALSO

exit(2), **wait**(2), **waitpid**(2), **exit**(3C)

Index

A

ASCII character set
— `ascii`, 5-6

C

character definitions for equations — `eqnchar`, 5-12
code set conversion tables
— `iconv`, 5-24

D

data types, primitive system
— `types`, 5-74
document production
`man` — macros to format manual pages, 5-37
`mansun` — macros to format manual pages, 5-41
`me` — macros to format technical papers, 5-46
`ms` — macros to format articles, theses and books, 5-49
special character definitions for equations — `eqnchar`, 5-12

E

`environ` — user environment, 5-7
environment variables
HOME, 5-7
LANG, 5-7
environment variables, *continued*
LC_COLLATE, 5-7
LC_CTYPE, 5-7
LC_MESSAGES, 5-7
LC_MONETARY, 5-7
LC_NUMERIC, 5-7
LC_TIME, 5-7
MSGVERB, 5-7
NETPATH, 5-7
PATH, 5-7
SEV_LEVEL, 5-7
TERM, 5-7
TZ, 5-7
`eqnchar` — special character definitions for equations, 5-12

F

file control options
— `fcntl`, 5-13
`filesystem` — file system layout, 5-16
/export File System, 5-21
/usr File System, 5-18
Root File System, 5-16
`floatingpoint` — IEEE floating point definitions, 5-22

I

iconv — code set conversion tables, 5-24
IEEE arithmetic
    floating point definitions — floatingpoint, 5-22

L

language data types, native — nl_types, 5-53
language information constants — langinfo, 5-35

M

machine-dependent values
    — values, 5-76
macros
    to format articles, theses and books — ms, 5-49
    to format Manual pages — man, 5-37, 5-41
    to format technical papers — me, 5-46
man — macros to format manual pages, 5-37
mansun — macros to format manual pages, 5-41
math — math functions and constants, 5-45
math functions and constants — math, 5-45
me — macros to format technical papers, 5-46
ms — macros to format articles, theses and books, 5-49

N

nl_types — native language data types, 5-53

P

processes
    base signals — signal, 5-62
    signal generation information — siginfo, 5-59
    wait status — wstat, 5-79
profiling utilities
    profile within a function — prof, 5-54

R

regular expression compile and match routines
    — advance, 5-55
    — compile, 5-55
    — regexp, 5-55
    — step, 5-55

S

shell environment
    conventional names for terminals — term, 5-71
signal — base signals, 5-62
signal generation information
    — siginfo, 5-59
special character definitions for equations — eqnchar, 5-12
stat — data returned by stat system call, 5-67
system calls
    — stat, 5-67

T

term — conventional names for terminals, 5-71
terminals
    conventional names — term, 5-71

U

UNIX System Code
    data types — types, 5-74
user context
    — ucontext, 5-75
user environment
    — environ, 5-7

V

values — machine-dependent values, 5-76
variable arguments
    handle list — stdarg, 5-69, 5-77

W

wait status
    — wstat, 5-79