NAME

apropos, whatis — search manual page databases

SYNOPSIS

apropos [−afk] [−C *file*] [−M *path*] [−m *path*] [−O *outkey*] [−S *arch*] [−s *section*]
expression ...

DESCRIPTION

The **apropos** and **whatis** utilities query manual page databases generated by *makewhatis*(8), evaluating *expression* for each file in each database. By default, they display the names, section numbers, and description lines of all matching manuals.

By default, **apropos** searches for *makewhatis*(8) databases in the default paths stipulated by *man*(1) and uses case-insensitive extended regular expression matching over manual names and descriptions (the Nm and Nd macro keys). Multiple terms imply pairwise −o.

whatis is a synonym for **apropos** −f.

The options are as follows:

−a
: Instead of showing only the title lines, show the complete manual pages, just like *man*(1) −a would. If the standard output is a terminal device and −c is not specified, use *less*(1) to paginate them. In −a mode, the options −IKOTW described in the *mandoc*(1) manual are also available.

−C *file*
: Specify an alternative configuration *file* in *man.conf*(5) format.

−f
: Search for all words in *expression* in manual page names only. The search is case-insensitive and matches whole words only. In this mode, macro keys, comparison operators, and logical operators are not available.

−k
: Support the full *expression* syntax. It is the default for **apropos**.

−M *path*
: Use the colon-separated path instead of the default list of paths searched for *makewhatis*(8) databases. Invalid paths, or paths without manual databases, are ignored.

−m *path*
: Prepend the colon-separated paths to the list of paths searched for *makewhatis*(8) databases. Invalid paths, or paths without manual databases, are ignored.

−O *outkey*
: Show the values associated with the key *outkey* instead of the manual descriptions.

−S *arch*
: Restrict the search to pages for the specified *machine*(1) architecture. *arch* is case-insensitive. By default, pages for all architectures are shown.

−s *section*
: Restrict the search to the specified section of the manual. By default, pages from all sections are shown. See *man*(1) for a listing of sections.

The options −chlw are also supported and are documented in *man*(1). The options −fkl are mutually exclusive and override each other.

An *expression* consists of search terms joined by logical operators −a (and) and −o (or). The −a operator has precedence over −o and both are evaluated left-to-right.

(*expr*)
: True if the subexpression *expr* is true.

expr1 −a *expr2*
: True if both *expr1* and *expr2* are true (logical ‘and’).

expr1 [−o] *expr2*
: True if *expr1* and/or *expr2* evaluate to true (logical ‘or’).

term
: True if *term* is satisfied. This has syntax [[*key*[,*key*...]](=|~)]*val*, where *key* is an *mdoc*(7) macro to query and *val* is its value. See “Macro Keys” for a list of available keys. Operator = evaluates a substring, while ~ evaluates a case-sensitive extended regular expression.

−i *term*
: If *term* is a regular expression, it is evaluated case-insensitively. Has no effect on substring terms.

Results are sorted first according to the section number in ascending numerical order, then by the page name in ascending *ascii*(7) alphabetical order, case-insensitive.

Each output line is formatted as

name[, name...](sec) − description

Where “name” is the manual’s name, “sec” is the manual section, and “description” is the manual’s short description. If an architecture is specified for the manual, it is displayed as

name(sec/arch) − description

Resulting manuals may be accessed as

```
$ man -s sec name
```

If an architecture is specified in the output, use

```
$ man -s sec -S arch name
```

Macro Keys

Queries evaluate over a subset of *mdoc*(7) macros indexed by *makewhatis*(8). In addition to the macro keys listed below, the special key `any` may be used to match any available macro key.

Names and description:

`Nm` manual name
`Nd` one-line manual description
`arch` machine architecture (case-insensitive)
`sec` manual section number

Sections and cross references:

`Sh` section header (excluding standard sections)
`Ss` subsection header
`Xr` cross reference to another manual page
`Rs` bibliographic reference

Semantic markup for command line utilities:

`Fl` command line options (flags)
`Cm` command modifier
`Ar` command argument
`Ic` internal or interactive command
`Ev` environmental variable
`Pa` file system path

Semantic markup for function libraries:

`Lb` function library name
`In` include file
`Ft` function return type
`Fn` function name
`Fa` function argument type and name
`Vt` variable type

`Va` variable name
`Dv` defined variable or preprocessor constant
`Er` error constant
`Ev` environmental variable

Various semantic markup:

`An` author name
`Lk` hyperlink
`Mt` “mailto” hyperlink
`Cd` kernel configuration declaration
`Ms` mathematical symbol
`Tn` tradename

Physical markup:

`Em` italic font or underline
`Sy` boldface font
`Li` typewriter font

Text production:

`St` reference to a standards document
`At` AT&T UNIX version reference
`Bx` BSD version reference
`Bsx` BSD/OS version reference
`Nx` NetBSD version reference
`Fx` FreeBSD version reference
`Ox` OpenBSD version reference
`Dx` DragonFly version reference

In general, macro keys are supposed to yield complete results without expecting the user to consider actual macro usage. For example, results include:

`Fa` function arguments appearing on **`Fn`** lines
`Fn` function names marked up with **`Fo`** macros
`In` include file names marked up with **`Fd`** macros
`Vt` types appearing as function return types and
types appearing in function arguments in the SYNOPSIS

ENVIRONMENT

MANPAGER
Any non-empty value of the environment variable *MANPAGER* is used instead of the standard pagination program, *less*(1); see *man*(1) for details. Only used if `-a` or `-l` is specified.

MANPATH A colon-separated list of directories to search for manual pages; see *man*(1) for details. Overridden by `-M`, ignored if `-l` is specified.

PAGER Specifies the pagination program to use when *MANPAGER* is not defined. If neither PAGER nor MANPAGER is defined, *less*(1) is used. Only used if `-a` or `-l` is specified.

FILES

mandoc.db name of the *makewhatis*(8) keyword database
/etc/man.conf default *man*(1) configuration file

EXIT STATUS

The **`apropos`** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Search for ".cf" as a substring of manual names and descriptions:

```
$ apropos =.cf
```

Include matches for ".cnf" and ".conf" as well:

```
$ apropos =.cf =.cnf =.conf
```

Search in names and descriptions using a case-sensitive regular expression:

```
$ apropos '~set.?[ug]id'
```

Search for all manual pages in a given section:

```
$ apropos -s 9 .
```

Search for manuals in the library section mentioning both the "optind" and the "optarg" variables:

```
$ apropos -s 3 Va=optind -a Va=optarg
```

Do exactly the same as calling **whatis** with the argument "ssh":

```
$ apropos -- -i 'Nm~[[:<:]]ssh[[:>:]]'
```

The following two invocations are equivalent:

```
$ apropos -S arch -s section expression
$ apropos \( expression \) -a arch~^(arch|any)$ -a sec~^section$
```

SEE ALSO

man(1), *re_format*(7), *makewhatis*(8)

STANDARDS

The **apropos** utility is compliant with the IEEE Std 1003.1-2008 (“POSIX.1”) specification of *man*(1) `-k`.

All options, the **whatis** command, support for logical operators, macro keys, substring matching, sorting of results, the environment variables *MANPAGER* and *MANPATH*, the database format, and the configuration file are extensions to that specification.

HISTORY

Part of the functionality of **whatis** was already provided by the former **manwhere** utility in 1BSD. The **apropos** and **whatis** utilities first appeared in 2BSD. They were rewritten from scratch for OpenBSD 5.6.

The `-M` option and the *MANPATH* variable first appeared in 4.3BSD; `-m` in 4.3BSD-Reno; `-C` in 4.4BSD Lite1; and `-S` and `-s` in OpenBSD 4.5 for **apropos** and in OpenBSD 5.6 for **whatis**. The options `-acfhIKklOTWw` appeared in OpenBSD 5.7.

AUTHORS

Bill Joy wrote **manwhere** in 1977 and the original BSD **apropos** and **whatis** in February 1979. The current version was written by Kristaps Dzonsons <kristaps@bsd.lv> and Ingo Schwarze <schwarze@openbsd.org>.

NAME

demandoc — emit only text of UNIX manuals

SYNOPSIS

demandoc [`-w`] [*`file ...`*]

DESCRIPTION

The **demandoc** utility emits only the text portions of well-formed *mdoc*(7) and *man*(7) Unix manual files.

By default, **demandoc** parses standard input and outputs only text nodes, preserving line and column position. Escape sequences are omitted from the output.

Its arguments are as follows:

`-w` Output a word list. This outputs each word of text on its own line. A "word", in this case, refers to whitespace-delimited terms beginning with at least two letters and not consisting of any escape sequences. Words have their leading and trailing punctuation (double-quotes, sentence punctuation, etc.) stripped.

`file ...`
The input files.

If a document is not well-formed, it is skipped.

The `-i`, `-k`, `-m`, and `-p` flags are silently discarded for calling compatibility with the historical deroff.

EXIT STATUS

The **demandoc** utility exits with one of the following values:

0 No errors occurred.

6 An operating system error occurred, for example memory exhaustion or an error accessing input files. Such errors cause **demandoc** to exit at once, possibly in the middle of parsing or formatting a file. The output databases are corrupt and should be removed .

EXAMPLES

The traditional usage of **demandoc** is for spell-checking manuals on BSD. This is accomplished as follows (assuming British spelling):

```
$ demandoc -w file.1 | spell -b
```

SEE ALSO

mandoc(1), *man*(7), *mdoc*(7)

HISTORY

demandoc replaces the historical deroff utility for handling modern *man*(7) and *mdoc*(7) documents.

AUTHORS

The **demandoc** utility was written by Kristaps Dzonsons <kristaps@bsd.lv>.

NAME

man — display manual pages

SYNOPSIS

man [−acfhklw] [−C *file*] [−M *path*] [−m *path*] [−S *subsection*] [[−s] *section*] *name* ...

DESCRIPTION

The **man** utility displays the manual page entitled *name*. Pages may be selected according to a specific category (*section*) or machine architecture (*subsection*).

The options are as follows:

−a
: Display all matching manual pages.

−C *file*
: Use the specified *file* instead of the default configuration file. This permits users to configure their own manual environment. See *man.conf*(5) for a description of the contents of this file.

−c
: Copy the manual page to the standard output instead of using *less*(1) to paginate it. This is done by default if the standard output is not a terminal device.

  When using −c, most terminal devices are unable to show the markup. To print the output of **man** to the terminal with markup but without using a pager, pipe it to *ul*(1). To remove the markup, pipe the output to *col*(1) −b instead.

−f
: A synonym for *whatis*(1). It searches for *name* in manual page names and displays the header lines from all matching pages. The search is case insensitive and matches whole words only.

−h
: Display only the SYNOPSIS lines of the requested manual pages. Implies −a and −c.

−k
: A synonym for *apropos*(1). Instead of *name*, an expression can be provided using the syntax described in the *apropos*(1) manual. By default, it displays the header lines of all matching pages.

−l
: A synonym for *mandoc*(1). The *name* arguments are interpreted as filenames. No search is done and *file*, *path*, *section*, *subsection*, and −w are ignored. This option implies −a.

−M *path*
: Override the list of directories to search for manual pages. The supplied *path* must be a colon (‘:’) separated list of directories. This option also overrides the environment variable *MANPATH* and any directories specified in the *man.conf*(5) file.

−m *path*
: Augment the list of directories to search for manual pages. The supplied *path* must be a colon (‘:’) separated list of directories. These directories will be searched before those specified using the −M option, the *MANPATH* environment variable, the *man.conf*(5) file, or the default directories.

−S *subsection*
: Only show pages for the specified *machine*(1) architecture. *subsection* is case insensitive.

  By default manual pages for all architectures are installed. Therefore this option can be used to view pages for one architecture whilst using another.

  This option overrides the *MACHINE* environment variable.

[−s] *section*
: Only select manuals from the specified *section*. The currently available sections are:

1	General commands (tools and utilities).
2	System calls and error numbers.
3	Library functions.

3p	*perl*(1) programmer's reference guide.
4	Device drivers.
5	File formats.
6	Games.
7	Miscellaneous information.
8	System maintenance and operation commands.
9	Kernel internals.

`-w` List the pathnames of all matching manual pages instead of displaying any of them. If no *name* is given, list the directories that would be searched.

The options `-IKOTW` are also supported and are documented in *mandoc*(1). The options `-fkl` are mutually exclusive and override each other.

The search starts with the `-m` argument if provided, then continues with the `-M` argument, the *MANPATH* variable, the **manpath** entries in the *man.conf*(5) file, or with */usr/share/man:/usr/X11R6/man:/usr/local/man* by default. Within each of these, directories are searched in the order provided. Within each directory, the search proceeds according to the following list of sections: 1, 8, 6, 2, 3, 5, 7, 4, 9, 3p. The first match found is shown.

The *mandoc.db*(5) database is used for looking up manual page entries. In cases where the database is absent, outdated, or corrupt, **man** falls back to looking for files called *name*.*section*. If both a formatted and an unformatted version of the same manual page, for example *cat1/foo.0* and *man1/foo.1*, exist in the same directory, only the unformatted version is used. The database is kept up to date with *makewhatis*(8), which is run by the *weekly*(8) maintenance script.

Guidelines for writing man pages can be found in *mdoc*(7).

ENVIRONMENT

MACHINE As some manual pages are intended only for specific architectures, **man** searches any subdirectories, with the same name as the current architecture, in every directory which it searches. Machine specific areas are checked before general areas. The current machine type may be overridden by setting the environment variable *MACHINE* to the name of a specific architecture, or with the `-S` option. *MACHINE* is case insensitive.

MANPAGER

Any non-empty value of the environment variable *MANPAGER* is used instead of the standard pagination program, *less*(1). If *less*(1) is used, the interactive **:t** command can be used to go to the definitions of various terms, for example command line options, command modifiers, internal commands, environment variables, function names, preprocessor macros, *errno*(2) values, and some other emphasized words. Some terms may have defining text at more than one place. In that case, the *less*(1) interactive commands **t** and **T** can be used to move to the next and to the previous place providing information about the term last searched for with **:t**. The `-O tag`[=*term*] option documented in the *mandoc*(1) manual opens a manual page at the definition of a specific *term* rather than at the beginning.

MANPATH Override the standard search path which is either specified in *man.conf*(5) or the default path. The format of *MANPATH* is a colon ('`:`') separated list of directories. Invalid directories are ignored. Overridden by `-M`, ignored if `-l` is specified.

If *MANPATH* begins with a colon, it is appended to the standard path; if it ends with a colon, it is prepended to the standard path; or if it contains two adjacent colons, the standard path is inserted between the colons.

PAGER Specifies the pagination program to use when *MANPAGER* is not defined. If neither PAGER nor MANPAGER is defined, *less*(1) is used.

FILES

/etc/man.conf default **man** configuration file

EXIT STATUS

The **man** utility exits 0 on success, and >0 if an error occurs. See *mandoc*(1) for details.

EXAMPLES

Format a page for pasting extracts into an email message — avoid printing any UTF-8 characters, reduce the width to ease quoting in replies, and remove markup:

```
$ man -T ascii -O width=65 pledge | col -b
```

Read a typeset page in a PDF viewer:

```
$ MANPAGER=mupdf man -T pdf lpd
```

SEE ALSO

apropos(1), *col*(1), *mandoc*(1), *ul*(1), *whereis*(1), *man.conf*(5), *mdoc*(7)

STANDARDS

The **man** utility is compliant with the IEEE Std 1003.1-2008 (“POSIX.1”) specification.

The flags [`-aCcfhIKlMmOSsTWw`], as well as the environment variables *MACHINE*, *MANPAGER*, and *MANPATH*, are extensions to that specification.

HISTORY

A **man** command first appeared in Version 2 AT&T UNIX.

The `-w` option first appeared in Version 7 AT&T UNIX; `-f` and `-k` in 4BSD; `-M` in 4.3BSD; `-a` in 4.3BSD−Tahoe; `-c` and `-m` in 4.3BSD−Reno; `-h` in 4.3BSD Net/2; `-C` in NetBSD 1.0; `-s` and `-S` in OpenBSD 2.3; and `-I`, `-K`, `-l`, `-O`, and `-W` in OpenBSD 5.7. The `-T` option first appeared in AT&T System III UNIX and was also added in OpenBSD 5.7.

NAME

man.options — assignment of option letters in manual page utilities

DESCRIPTION

This manual page lists option letters used in many different versions of the **man**, **apropos**, **whatis**, **mandoc**, **makewhatis**, **mandb**, **makemandb**, **catman**, and **manpath** utilities. Option letters used by **groff**, **nroff**, **troff**, and **roff** are also included because beginning with Version 7 AT&T UNIX, many versions of *man*(1) pass on unrecognized options to these programs.

For each option letter, information is first grouped into paragraphs, each paragraph describing similar functionality and starting with one line briefly summarizing that functionality.

For each program using the letter for that functionality, one line is provided, giving the name of the program, a colon, the system where this letter first appeared for this functionality in this program, optionally a comma and a list of other system versions introducing the same, a semicolon, and a list of current systems supporting it. If a system appears before the semicolon, it is not repeated afterwards.

Entries are sorted by historical precedence, except that obsolete options are moved to the end. Dates are commit dates where known, and release dates otherwise.

`-a` display all matching manual pages
man: 4.3BSD−Tahoe (June 1988), Eaton (before July 7, 1993; 1990/91?); OpenBSD, FreeBSD, NetBSD, man-db, man-1.6, illumos, Solaris 9-11
apropos, **whatis**, **mandoc**: OpenBSD 5.7 (August 27, 2014)

only display items that match all keywords
apropos: man-db (Aug 29, 2007)

use all directories and files for *mandoc.db*(5)
makewhatis: OpenBSD 5.6 (April 18, 2014)

[superseded by `-T ascii`] ASCII output mode
troff: Version 7 AT&T UNIX (January 1979)
groff: probably before groff-0.4 (before July 14, 1990)

`-B` use specified browser
man: man-1.6 (June 24, 2005)

`-b` print a backtrace with each warning or error message
groff: probably before groff-0.4 (before July 14, 1990)

[obsolete hardware] report whether the phototypesetter is busy
troff: Version 7 AT&T UNIX (January 1979)

`-C` alternate configuration file
apropos, **whatis**: 4.4BSD Lite1 (April 22, 1994), man-db (Feb 22, 2003); OpenBSD, NetBSD
man: NetBSD 1.0 (Oct 26, 1994), man-1.5e (not before 1993, not after 1998); OpenBSD
mandb, **catman**, **manpath**: man-db (Feb 22, 2003)
makemandb: NetBSD (Feb 7, 2012)
makewhatis: OpenBSD 5.6 (April 18, 2014)
mandoc: OpenBSD 5.7 (August 27, 2014)

[obsolete] enable compatibility mode
groff: before groff-0.5 (before August 3, 1990)

`-c` do not use a pager
man: 4.3BSD−Reno (June 1990); OpenBSD, NetBSD
apropos, **whatis**, **mandoc**: OpenBSD 5.7 (August 27, 2014)

process given catpath
makewhatis: (not before 1992, not after 1995)

recreate databases from scratch
mandb: man-db probably before 2.2a4 (before Nov 8, 1994)

produce a catpath as opposed to a manpath
manpath: man-db probably before 2.2a4 (before Nov 8, 1994)

internal option for use by *catman*(1)
man: man-db probably before 2.2a4 (before Nov 8, 1994)

reformat source page even if cat page exists
man: man-1.5e (not before 1993, not after 1998)

disable terminal color output in *grotty*(1)
groff: groff-1.18.0 (Oct 4, 2001)

recreate nroff versions from SGML sources
catman: Solaris 9-11

[obsolete] postprocess with *col*(1)
man: AT&T System III UNIX (June 1980), AT&T System V UNIX (January 1983)

`-D` reset whatever was set with *MANOPT*
man: man-db probably before 2.2a4 (before Nov 8, 1994)

print debugging info in addition to manual page
man: man-1.5e (not before 1993, not after 1998)

set default input encoding for *preconv*(1)
groff: groff-1.20 (August 20, 2008)

display all files added to *mandoc.db*(5)
makewhatis: OpenBSD 5.6 (April 18, 2014)

`-d` define a user-defined string
groff: probably before groff-0.4 (before July 14, 1990)

print debugging information
man: Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db, man-1.6, illumos, Solaris 9-11
manpath: Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db
apropos, **whatis**: man-db probably before 2.2a4 (before Nov 8, 1994); FreeBSD
mandb, **catman**: man-db probably before 2.2a4 (before Nov 8, 1994)

remove and re-add a file to *mandoc.db*(5)
makewhatis: OpenBSD 2.7 (Feb 3, 2000)

[superseded by `-l`] interpret arguments as file names
man: AT&T System III UNIX (June 1980), AT&T System V UNIX (January 1983)

`-E` inhibit all error messages
groff: probably before groff-0.4 (before July 14, 1990)

select output encoding
man: man-db (Dec 23, 2001)

`-e` preprocess with *eqn*(7)
man: Version 7 AT&T UNIX (January 1979)
groff: probably before groff-0.4 (before July 14, 1990)

adjust text to left and right margins
nroff: Version 7 AT&T UNIX (January 1979)

use exact matching
apropos, **whatis**: man-db probably before 2.2a4 (before Nov 8, 1994)

restrict search by section extension
man: man-db-2.3.5 (April 21, 1995)

`-F` use alternate font directory
troff: 4.2BSD (September 1983)
groff: probably before groff-0.4 (before July 14, 1990)

preformat only, do not display
man: man-1.5g (April 7, 1999)

force searching dirs, do not use index (default)
man: illumos, Solaris 9-11

`-f` *whatis*(1) mode
man: 4BSD (November 16, 1980), Eaton (before July 7, 1993; 1990/91?); OpenBSD, FreeBSD, man-db, man-1.6
apropos, **whatis**: man-db (Dec 2, 2010), OpenBSD 5.7 (August 27, 2014)
mandoc: OpenBSD 5.7 (August 27, 2014)

set the default font family
groff: probably before groff-0.4 (before July 14, 1990)

force formatting even if cat page is newer
catman: FreeBSD (March 15, 1995)

update only the entries for the given file
mandb: man-db (Feb 21, 2003)

force rebuilding the database from scratch
makemandb: NetBSD (Feb 7, 2012)

locate manual page related to given file name
man: illumos, Solaris 9-11

[obsolete hardware] do not feed out paper nor stop phototypesetter
troff: Version 7 AT&T UNIX (January 1979)

`-G` preprocess with *grap*(1)
groff: groff-1.16 (May 1, 2000)

`-g` produce a global manpath
manpath: man-db-2.2a7 (Nov 16, 1994)

preprocess with *grn*(1)
groff: groff-1.16 (Feb 20, 2000)

[obsolete hardware] output to a GCOS phototypesetter
troff: Version 7 AT&T UNIX (January 1979)

[obsolete hardware] output to a DASI 300 terminal in 12-pitch mode
man: PWB/UNIX 1.0 (July 1, 1977)

`-H` read hyphenation patterns from the given file
groff: probably before groff-0.4 (before July 14, 1990)

produce HTML output
man: man-db-1.3.12 to 1.3.17 (not before 1996, not after 2001)

use program to render HTML files as text
man: man-1.6 (June 24, 2005)

`-h` print a help message and exit
groff: probably before groff-0.4 (before July 14, 1990)
man: Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db, man-1.6
manpath: Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db

apropos, **whatis**, **mandb**, **catman**: man-db probably before 2.2a4 (before Nov 8, 1994)

display the SYNOPSIS lines only
man: 4.3BSD Net/2 (August 20, 1991); OpenBSD, NetBSD
apropos, **whatis**, **mandoc**: OpenBSD 5.7 (Sep 3, 2014)

turn on HTML formatting
apropos: NetBSD (Apr 2, 2013)

[obsolete] replace spaces by tabs in the output
roff, **nroff**: Version 7 AT&T UNIX (January 1979)

`-I` input file search path for *soelim*(1)
groff: groff-1.12 (Sep 11, 1999)

respect case when matching manual page names
man, **catman**: man-db (Apr 21, 2002)

input options, in particular default operating system name
mandoc: OpenBSD 5.2 (May 24, 2012)
man, **apropos**, **whatis**: OpenBSD 5.7 (August 27, 2014)

`-i` read standard input after the input files are exhausted
nroff, **troff**: Version 7 AT&T UNIX (January 1979)
groff: probably before groff-0.4 (before July 14, 1990)

ignore case when matching manual page names
man, **catman**: man-db (Apr 21, 2002)

turn on terminal escape code formatting
apropos: NetBSD (March 29, 2013)

`-J` preprocess with *gideal*(1)
groff: groff-1.22.3 (June 17, 2014)

`-j` preprocess with *chem*(1)
groff: groff-1.22 (Jan 22, 2011)

`-K` source code full text search
man: man-1.5e (not before 1993, not after 1998), man-db (June 28, 2009); Solaris 11

input encoding
groff: groff-1.20 (Dec 31, 2005)
man, **apropos**, **whatis**, **mandoc**: OpenBSD 5.7 (Oct 30, 2014)

`-k` *apropos*(1) mode
man: 4BSD (November 16, 1980), Eaton (before July 7, 1993; 1990/91?); POSIX, OpenBSD, FreeBSD, NetBSD, man-db, man-1.6, illumos, Solaris 9-11
apropos, **whatis**, **mandoc**: OpenBSD 5.7 (August 27, 2014)

ignore formatting errors
catman: NetBSD (April 26, 1994)

preprocess with *preconv*(1)
groff: groff-1.20 (Dec 31, 2005)

[obsolete hardware] display on a Tektronix 4014 terminal
man: Version 7 AT&T UNIX (January 1979)

`-L` pass argument to the spooler
groff: groff-0.6 (Sep 14, 1990)

use alternate *locale*(1)
man, **apropos**, **whatis**: before man-db-2.2a13 (before Dec 15, 1994)

print list of locales
manpath: FreeBSD (Nov 23, 1999)

use *locale*(1) specified in the environment
catman: FreeBSD (May 18, 2002)

`-l` spool the output
groff: probably before groff-0.4 (before July 14, 1990)

interpret arguments as file names
man: before man-2.2a7 (before Nov 16, 1994), OpenBSD 5.7 (Aug 30, 2014)
apropos, **whatis**, **mandoc**: OpenBSD 5.7 (Aug 30, 2014)

do not trim output to the terminal width
apropos, **whatis**: man-db (Aug 19, 2007)

only parse NAME sections
makemandb: NetBSD (Feb 7, 2012)

legacy mode: search Nm,Nd, no context or formatting
apropos: NetBSD (March 29, 2013)

list all manual pages matching name within the search path
man: illumos, Solaris 9-11

`-M` override manual page search path
man: 4.3BSD (June 1986), Eaton (before July 7, 1993; 1990/91?); OpenBSD, FreeBSD, NetBSD, man-db, man-1.6, illumos, Solaris 9-11
apropos, **whatis**: 4.3BSD (June 1986), before man-db-2.2a14 (before Dec 16, 1994); OpenBSD, illumos
catman: man-db probably before 2.2a4 (before Nov 8, 1994); NetBSD (July 27, 1993), Solaris 9-11
mandoc: OpenBSD 5.7 (August 27, 2014)

prepend to macro file search path
groff: probably before groff-0.4 (before July 14, 1990)

do not show the context of the match
apropos: NetBSD (May 22, 2016)

`-m` specify input macro language
nroff, **troff**: Version 7 AT&T UNIX (January 1979)
groff: probably before groff-0.4 (before July 14, 1990)
mandoc: OpenBSD 4.8 (April 6, 2009)

augment manual page search path
man, **apropos**, **whatis**: 4.3BSD−Reno (June 1990); OpenBSD, NetBSD
catman: NetBSD (Apr 4, 1999)
mandoc: OpenBSD 5.7 (August 27, 2014)

override operating system
man: Eaton (before July 7, 1993; 1990/91?); man-db, man-1.6
apropos, **whatis**, **manpath**: man-db probably before 2.2a4 (before Nov 8, 1994)

override architecture
man: FreeBSD (Jan 11, 2002)

show the context of the match
apropos: NetBSD (May 22, 2016)

`-N` do not allow newlines between *eqn*(7) delimiters
groff: groff-1.01 (Feb 21, 1991)

`-n` specify a page number for the first page
troff: Version 7 AT&T UNIX (January 1979)
groff: probably before groff-0.4 (before July 14, 1990)

nroff(1) output mode
man: Version 7 AT&T UNIX (January 1979)

do not create the *whatis*(1) database
catman: NetBSD (July 27, 1993)

print commands instead of executing them
catman: FreeBSD (May 18, 2002), Solaris 9-11

limit the number of results
apropos: NetBSD (Feb 7, 2012)

dry run simulating *mandoc.db*(5) creation
makewhatis: OpenBSD 5.6 (April 18, 2014)

`-O` output options
mandoc: OpenBSD 4.8 (Oct 27, 2009)
man, **apropos**, **whatis**: OpenBSD 5.7 (August 27, 2014)

`-o` select pages by numbers
nroff, **troff**: Version 7 AT&T UNIX (January 1979)
groff: probably before groff-0.4 (before July 14, 1990)

force use of non-localized manual pages
man: FreeBSD (June 7, 1999)

optimize index for speed and disk space
makemandb: NetBSD (Feb 7, 2012)

`-P` pass argument to postprocessor
groff: groff-0.6 (Sep 14, 1990)

use specified pager
man: Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db, man-1.6

turn on pager formatting
apropos: NetBSD (Apr 2, 2013)

`-p` preprocess with *pic*(1)
groff: probably before groff-0.4 (before July 14, 1990)

use the given list of preprocessors
man: Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db, man-1.6

dry run, display commands instead of executing them
catman: NetBSD (July 27, 1993), FreeBSD (March 15, 1995 to May 18, 2002), Solaris 9-11

print warnings when building *mandoc.db*(5)
makewhatis: OpenBSD 2.7 (April 23, 2000)

do not look for deleted manual pages
mandb: man-db (June 28, 2001)

print the search path for manual pages
man: NetBSD (June 14, 2011)

turn on pager formatting and pipe through pager
apropos: NetBSD (Feb 7, 2012)

[obsolete hardware] set phototypesetter point size
troff: Version 7 AT&T UNIX (January 1979)

`-Q` print only fatal error messages
makemandb: NetBSD (Aug 29, 2012)

quick mode of *mandoc.db*(5) creation
makewhatis: OpenBSD 5.6 (April 18, 2014)

`-q` invoke the simultaneous input-output mode of the .rd request
nroff, **troff**: Version 7 AT&T UNIX (January 1979)

issue no warnings
manpath: Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db
mandb: man-db probably before 2.2a4 (before Nov 8, 1994)

print only warnings and errors, no status updates
makemandb: NetBSD (Aug 29, 2012)

`-R` postprocess with *refer*(1)
groff: groff-1.02 (June 2, 1991)

recode to the specified encoding
man: man-db (Dec 31, 2007)

`-r` set number register
nroff, **troff**: Version 7 AT&T UNIX (January 1979)
groff: probably before groff-0.4 (before July 14, 1990)

scan for and remove junk files
catman: FreeBSD (March 31, 1995)

set *less*(1) prompt
man: man-db-2.3.5 (April 21, 1995)

use regular expression matching
apropos, **whatis**: man-db-2.3.5 (April 21, 1995)

turn off formatting
apropos: NetBSD (Feb 10, 2013)

check for formatting errors, do not display
man: illumos, Solaris 9-11

`-S` manual section search list
man: Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db, man-1.6

safer mode
groff: groff-1.10 (May 17, 1994)

restrict architecture
man: OpenBSD 2.3 (March 9, 1998), NetBSD (May 27, 2000)
apropos: OpenBSD 4.5 (Dec 24, 2008), NetBSD (May 8, 2009)
whatis: OpenBSD 5.6 (April 18, 2014)
mandoc: OpenBSD 5.7 (August 27, 2014)

`-s` preprocess with *soelim*(1)
groff: probably before groff-0.4 (before July 14, 1990)

silent mode, do not echo commands
catman: NetBSD (April 26, 1994)

restrict section
makewhatis: man-1.5g (not before 1993, not after 1999)
man: OpenBSD 2.3 (March 9, 1998), NetBSD (June 12, 2000); illumos, Solaris 9-11
apropos: man-db (Nov 16, 2003), OpenBSD 4.5 (Dec 24, 2008), NetBSD (May 8, 2009); illumos
whatis: man-db (Nov 16, 2003), OpenBSD 5.6 (April 18, 2014); illumos
mandoc: OpenBSD 5.7 (August 27, 2014)

do not look for stray cats
mandb: man-db probably before 2.2a4 (before Nov 8, 1994)

[SysV compat, recommends `-S`] manual section search list
man: man-db (Jan 1, 2008)

[superseded by `-h`] display the SYNOPSIS lines only
man: PWB/UNIX 1.0 (July 1, 1977)

[obsolete hardware] pause before each page for paper manipulation
roff: Version 7 AT&T UNIX (January 1979)

[obsolete hardware] *troff*(1) output mode, small format
man: AT&T System III UNIX (June 1980), AT&T System V UNIX (January 1983)

`-T` select terminal output format
nroff: Version 7 AT&T UNIX (January 1979)
man: AT&T System III UNIX (June 1980), AT&T System V UNIX (January 1983), man-db probably before 2.2a4 (before Nov 8, 1994), OpenBSD 5.7 (August 27, 2014)
groff: probably before groff-0.4 (before July 14, 1990)
mandoc: OpenBSD 4.8 (April 6, 2009)
apropos, **whatis**: OpenBSD 5.7 (August 27, 2014)

use UTF-8 for *mandoc.db*(5)
makewhatis: OpenBSD 5.6 (April 18, 2014)

[superseded by `-m`] use other macro package
man, **catman**: Solaris 9-11

`-t` *troff*(1) output mode
man: PWB/UNIX 1.0 (July 1, 1977), Version 7 AT&T UNIX (January 1979), 2BSD (May 10, 1979), AT&T System III UNIX (June 1980), AT&T System V UNIX (January 1983), Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db, man-1.6, illumos, Solaris 9-11
catman: Solaris 9-11

preprocess with *tbl*(7)
groff: probably before groff-0.4 (before July 14, 1990)

check manual pages in the hierarchy
mandb: man-db-1.3.12 to 1.3.17 (not before 1996, not after 2001)

check files for problems related to *mandoc.db*(5)
makewhatis: OpenBSD 2.7 (April 23, 2000)

`-U` unsafe mode
groff: groff-1.12 (Dec 13, 1999)

`-u` update database
makewhatis: (not before 1992, not after 1995)

create user databases only
mandb: man-db probably before 2.2a4 (before Nov 8, 1994)

update database cache (requires suid)
man: before man-db-2.2a10 (before Dec 6, 1994)

remove files from *mandoc.db*(5)
makewhatis: OpenBSD 3.4 (July 9, 2003)

`-V` print the pipeline on stdout instead of executing it
groff: groff-0.6 (Sep 2, 1990)

print version information
man, **apropos**, **whatis**, **mandb**, **catman**, **manpath**: man-db probably before 2.2a4 (before Nov 8, 1994)

`-v` print version number
groff: probably before groff-0.4 (before July 14, 1990)

verbose mode
catman: FreeBSD (March 15, 1995)
makewhatis: man-1.5g (not before 1993, not after 1999)
apropos, **whatis**: man-db (Dec 29, 2002)

print the name of every parsed file
makemandb: NetBSD (Feb 7, 2012)

[obsolete hardware] produce output on the Versatec printer
man: PWB/UNIX 1.0 (July 1, 1977)

`-W` disable the named warning
groff: groff-0.5 (August 14, 1990)

list pathnames without additional information
man: man-1.5e (not before 1993, not after 1998)

list pathnames of cat files
man: man-db (Aug 13, 2002)

minimum message level to display
mandoc: OpenBSD 4.8 (April 6, 2009)
man, **apropos**, **whatis**: OpenBSD 5.7 (August 27, 2014)

`-w` list pathnames
man: Version 7 AT&T UNIX (January 1979), AT&T System III UNIX (June 1980), AT&T System V UNIX (January 1983), Eaton (before July 7, 1993; 1990/91?); OpenBSD, FreeBSD, NetBSD, man-db, man-1.6
apropos, **whatis**, **mandoc**: OpenBSD 5.7 (August 27, 2014)

enable the named warning
groff: groff-0.5 (August 14, 1990)

only create the *whatis*(1) database
catman: NetBSD (July 27, 1993), Solaris 9-11

use wildcard matching
apropos, **whatis**: man-db-2.3.5 (April 21, 1995)

use manpath obtained from man --path
makewhatis: man-1.5g (not before 1993, not after 1999)

update the *whatis*(1) database
man: illumos

[obsolete hardware] wait until the phototypesetter is available
troff: Version 7 AT&T UNIX (January 1979)

`-X` display with *gxditview*(1)
groff: groff-1.06 (Sep 1, 1992)
man: man-db probably before 2.2a4 (before Nov 8, 1994)

`-y` use the non-compacted version of the macros
man: AT&T System III UNIX (June 1980), AT&T System V UNIX (January 1983)

`-Z` do not run preprocessors
groff: probably before groff-0.4 (before July 14, 1990)
man: man-db-2.2a5 (Nov 10, 1994)

`-z` suppress formatted output from *troff*(1), print only error messages
groff: probably before groff-0.4 (before July 14, 1990)

`-7` ASCII output mode
man: man-db-2.3.5 (April 21, 1995)

`-?` print a help message and exit
groff: probably before groff-0.4 (before July 14, 1990)
man, **manpath**: Eaton (before July 7, 1993; 1990/91?); FreeBSD, man-db
apropos, **whatis**, **mandb**, **catman**: man-db probably before 2.2a4 (before Nov 8, 1994)

Multi-letter options:

`-hp` [obsolete hardware] output to a Hewlett Packard terminal
man: PWB/UNIX 1.0 (July 1, 1977)

`-12` [obsolete hardware] use 12-pitch for certain terminals
man: AT&T System III UNIX (June 1980), AT&T System V UNIX (January 1983)

`-450` [obsolete hardware] output to a DASI 450 terminal
man: PWB/UNIX 1.0 (July 1, 1977)

In Version 3 AT&T UNIX, *man*(1) had no options.
The syntax was: **man** *name* [*section*]

In Version 4 AT&T UNIX,
the syntax changed to: **man** [*section*] [*name* ...]

AUTHORS

This information was assembled by Ingo Schwarze <schwarze@openbsd.org> using

- the Unix Archive of the Unix Heritage Society
- the CSRG Archive CD-ROMs
- the FreeBSD SVN repository
- the OpenBSD CVS repository
- the NetBSD CVS repository
- the GNU roff (groff) git repository
- the 4.3BSD-Net/2 groff CHANGES file (Oct 1990 to March 1991)
- the 4.3BSD-Net/2 groff ChangeLog file (July 1990 to March 1991)
- the man-db CVS and git repositories (since April 2001)
- the man-db NEWS file (April 1995 to Dec 2016)
- the man-db ChangeLog-2013 file (Nov 1994 to Dec 2013)
- release tarballs man-1.5g (July 1998) to man-1.5p (Jan 2005), man-1.6 (June 2005), and man-1.6a to man-1.6g (Dec 2010)
- a makewhatis release tarball without version number from 1995
- the illumos manual pages on the WWW
- and Solaris 11, SunOS 5.10, and SunOS 5.9 machines at opencsw.org.

NAME

mandoc — format manual pages

SYNOPSIS

mandoc [−ac] [−I os=*name*] [−K *encoding*] [−mdoc | −man] [−O *options*] [−T *output*] [−W *level*] [*file* ...]

DESCRIPTION

The **mandoc** utility formats manual pages for display.

By default, **mandoc** reads *mdoc*(7) or *man*(7) text from stdin and produces −T locale output.

The options are as follows:

−a
: If the standard output is a terminal device and −c is not specified, use *less*(1) to paginate the output, just like *man*(1) would.

−c
: Copy the formatted manual pages to the standard output without using *less*(1) to paginate them. This is the default. It can be specified to override −a.

−I os=*name*
: Override the default operating system *name* for the *mdoc*(7) **Os** and for the *man*(7) **TH** macro.

−K *encoding*
: Specify the input encoding. The supported *encoding* arguments are us-ascii, iso-8859-1, and utf-8. If not specified, autodetection uses the first match in the following list:

  1. If the first three bytes of the input file are the UTF-8 byte order mark (BOM, 0xefbbbf), input is interpreted as utf-8.
  2. If the first or second line of the input file matches the **emacs** mode line format

     .\" -*- [...;] coding: *encoding*; -*-

     then input is interpreted according to *encoding*.
  3. If the first non-ASCII byte in the file introduces a valid UTF-8 sequence, input is interpreted as utf-8.
  4. Otherwise, input is interpreted as iso-8859-1.

−mdoc | −man
: With −mdoc, all input files are interpreted as *mdoc*(7). With −man, all input files are interpreted as *man*(7). By default, the input language is automatically detected for each file: if the first macro is **Dd** or **Dt**, the *mdoc*(7) parser is used; otherwise, the *man*(7) parser is used. With other arguments, −m is silently ignored.

−O *options*
: Comma-separated output options. See the descriptions of the individual output formats for supported *options*.

−T *output*
: Select the output format. Supported values for the *output* argument are ascii, html, the default of locale, man, markdown, pdf, ps, tree, and utf8.

  The special −T lint mode only parses the input and produces no output. It implies −W all and redirects parser messages, which usually appear on standard error output, to standard output.

−W *level*
: Specify the minimum message *level* to be reported on the standard error output and to affect the exit status. The *level* can be base, style, warning, error, or unsupp. The base level automatically derives the operating system from the contents of the **Os** macro, from the −Ios command line option, or from the *uname*(3) return value. The levels openbsd and netbsd are variants of base that bypass autodetection and request validation of base system conventions for a particular operating system. The level all is an alias for base. By default, **mandoc** is silent. See “EXIT

STATUS” and “DIAGNOSTICS” for details.

The special option `-W stop` tells **mandoc** to exit after parsing a file that causes warnings or errors of at least the requested level. No formatted output will be produced from that file. If both a *level* and `stop` are requested, they can be joined with a comma, for example `-W error,stop`.

file Read from the given input file. If multiple files are specified, they are processed in the given order. If unspecified, **mandoc** reads from standard input.

The options `-fhklw` are also supported and are documented in *man*(1). In `-f` and `-k` mode, **mandoc** also supports the options `-CMmOSs` described in the *apropos*(1) manual. The options `-fkl` are mutually exclusive and override each other.

ASCII Output

Use `-T ascii` to force text output in 7-bit ASCII character encoding documented in the *ascii*(7) manual page, ignoring the *locale*(1) set in the environment.

Font styles are applied by using back-spaced encoding such that an underlined character ‘c’ is rendered as ‘_\[bs]c’, where ‘\[bs]’ is the back-space character number 8. Emboldened characters are rendered as ‘c\[bs]c’. This markup is typically converted to appropriate terminal sequences by the pager or *ul*(1). To remove the markup, pipe the output to *col*(1) `-b` instead.

The special characters documented in *mandoc_char*(7) are rendered best-effort in an ASCII equivalent. In particular, opening and closing ‘single quotes’ are represented as characters number 0x60 and 0x27, respectively, which agrees with all ASCII standards from 1965 to the latest revision (2012) and which matches the traditional way in which *roff*(7) formatters represent single quotes in ASCII output. This correct ASCII rendering may look strange with modern Unicode-compatible fonts because contrary to ASCII, Unicode uses the code point U+0060 for the grave accent only, never for an opening quote.

The following `-O` arguments are accepted:

`indent`=*indent*
: The left margin for normal text is set to *indent* blank characters instead of the default of five for *mdoc*(7) and seven for *man*(7). Increasing this is not recommended; it may result in degraded formatting, for example overfull lines or ugly line breaks. When output is to a pager on a terminal that is less than 66 columns wide, the default is reduced to three columns.

`mdoc` Format *man*(7) input files in *mdoc*(7) output style. This prints the operating system name rather than the page title on the right side of the footer line, and it implies `-O indent`=5. One useful application is for checking that `-T man` output formats in the same way as the *mdoc*(7) source it was generated from.

`tag`[=*term*]
: If the formatted manual page is opened in a pager, go to the definition of the *term* rather than showing the manual page from the beginning. If no *term* is specified, reuse the first command line argument that is not a *section* number. If that argument is in *apropos*(1) *key*=*val* format, only the *val* is used rather than the argument as a whole. This is useful for commands like `man -akO tag Ic=ulimit` to search for a keyword and jump right to its definition in the matching manual pages.

`width`=*width*
: The output width is set to *width* instead of the default of 78. When output is to a pager on a terminal that is less than 79 columns wide, the default is reduced to one less than the terminal width. In any case, lines that are output in literal mode are never wrapped and may exceed the output width.

HTML Output

Output produced by `-T html` conforms to HTML5 using optional self-closing tags. Default styles use only CSS1. Equations rendered from *eqn*(7) blocks use MathML.

The file */usr/share/misc/mandoc.css* documents style-sheet classes available for customising output. If a style-sheet is not specified with `-O style`, `-T html` defaults to simple output (via an embedded style-sheet) readable in any graphical or text-based web browser.

Non-ASCII characters are rendered as hexadecimal Unicode character references.

The following `-O` arguments are accepted:

`fragment`
: Omit the <!DOCTYPE> declaration and the <html>, <head>, and <body> elements and only emit the subtree below the <body> element. The `style` argument will be ignored. This is useful when embedding manual content within existing documents.

`includes=`*`fmt`*
: The string *`fmt`*, for example, *`../src/%I.html`*, is used as a template for linked header files (usually via the **In** macro). Instances of '%I' are replaced with the include filename. The default is not to present a hyperlink.

`man=`*`fmt`*[`;`*`fmt`*]
: The string *`fmt`*, for example, *`../html%S/%N.%S.html`*, is used as a template for linked manuals (usually via the **Xr** macro). Instances of '%N' and '%S' are replaced with the linked manual's name and section, respectively. If no section is included, section 1 is assumed. The default is not to present a hyperlink. If two formats are given and a file *`%N.%S`* exists in the current directory, the first format is used; otherwise, the second format is used.

`style=`*`style.css`*
: The file *`style.css`* is used for an external style-sheet. This must be a valid absolute or relative URI.

`tag`[`=`*`term`*]
: Same syntax and semantics as for "ASCII Output". This is implemented by passing a **file://** URI ending in a fragment identifier to the pager rather than passing merely a file name. When using this argument, use a pager supporting such URIs, for example

```
MANPAGER='lynx -force_html' man -T html -O tag=MANPAGER man
MANPAGER='w3m -T text/html' man -T html -O tag=toc mandoc
```

  Consequently, for HTML output, this argument does not work with *more*(1) or *less*(1). For example, `MANPAGER=less man -T html -O tag=toc mandoc` does not work because *less*(1) does not support **file://** URIs.

`toc`
: If an input file contains at least two non-standard sections, print a table of contents near the beginning of the output.

Locale Output

By default, **mandoc** automatically selects UTF-8 or ASCII output according to the current *locale*(1). If any of the environment variables *LC_ALL*, *LC_CTYPE*, or *LANG* are set and the first one that is set selects the UTF-8 character encoding, it produces "UTF-8 Output"; otherwise, it falls back to "ASCII Output". This output mode can also be selected explicitly with `-T locale`.

Man Output

Use `-T man` to translate *mdoc*(7) input into *man*(7) output format. This is useful for distributing manual sources to legacy systems lacking *mdoc*(7) formatters. Embedded *eqn*(7) and *tbl*(7) code is not supported.

If the input format of a file is *man*(7), the input is copied to the output. The parser is also run, and as usual, the `-W` level controls which "DIAGNOSTICS" are displayed before copying the input to the output.

Markdown Output

Use `-T markdown` to translate *mdoc*(7) input to the markdown format conforming to John Gruber's 2004 specification. The output also almost conforms to the CommonMark specification.

The character set used for the markdown output is ASCII. Non-ASCII characters are encoded as HTML entities. Since that is not possible in literal font contexts, because these are rendered as code spans and code blocks in the markdown output, non-ASCII characters are transliterated to ASCII approximations in these contexts.

Markdown is a very weak markup language, so all semantic markup is lost, and even part of the presentational markup may be lost. Do not use this as an intermediate step in converting to HTML; instead, use `-T html` directly.

The *man*(7), *tbl*(7), and *eqn*(7) input languages are not supported by `-T markdown` output mode.

PDF Output

PDF-1.1 output may be generated by `-T pdf`. See "PostScript Output" for `-O` arguments and defaults.

PostScript Output

PostScript "Adobe-3.0" Level-2 pages may be generated by `-T ps`. Output pages default to letter sized and are rendered in the Times font family, 11-point. Margins are calculated as 1/9 the page length and width. Line-height is 1.4m.

Special characters are rendered as in "ASCII Output".

The following `-O` arguments are accepted:

`paper=`*`name`*

The paper size *`name`* may be one of *`a3`*, *`a4`*, *`a5`*, *`legal`*, or *`letter`*. You may also manually specify dimensions as *`NNxNN`*, width by height in millimetres. If an unknown value is encountered, *`letter`* is used.

UTF-8 Output

Use `-T utf8` to force text output in UTF-8 multi-byte character encoding, ignoring the *locale*(1) settings in the environment. See "ASCII Output" regarding font styles and `-O` arguments.

On operating systems lacking locale or wide character support, and on those where the internal character representation is not UCS-4, **mandoc** always falls back to "ASCII Output".

Syntax tree output

Use `-T tree` to show a human readable representation of the syntax tree. It is useful for debugging the source code of manual pages. The exact format is subject to change, so don't write parsers for it.

The first paragraph shows meta data found in the *mdoc*(7) prologue, on the *man*(7) **TH** line, or the fallbacks used.

In the tree dump, each output line shows one syntax tree node. Child nodes are indented with respect to their parent node. The columns are:

1. For macro nodes, the macro name; for text and *tbl*(7) nodes, the content. There is a special format for *eqn*(7) nodes.
2. Node type (text, elem, block, head, body, body-end, tail, tbl, eqn).
3. Flags:
   - An opening parenthesis if the node is an opening delimiter.
   - An asterisk if the node starts a new input line.
   - The input line number (starting at one).
   - A colon.
   - The input column number (starting at one).
   - A closing parenthesis if the node is a closing delimiter.
   - A full stop if the node ends a sentence.
   - BROKEN if the node is a block broken by another block.
   - NOSRC if the node is not in the input file, but automatically generated from macros.
   - NOPRT if the node is not supposed to generate output for any output format.

The following `-O` argument is accepted:

`noval` Skip validation and show the unvalidated syntax tree. This can help to find out whether a given behaviour is caused by the parser or by the validator. Meta data is not available in this case.

ENVIRONMENT

LC_CTYPE The character encoding *locale*(1). When "Locale Output" is selected, it decides whether to use ASCII or UTF-8 output format. It never affects the interpretation of input files.

MANPAGER
Any non-empty value of the environment variable *MANPAGER* is used instead of the standard pagination program, *less*(1); see *man*(1) for details. Only used if `-a` or `-l` is specified.

PAGER Specifies the pagination program to use when *MANPAGER* is not defined. If neither PAGER nor MANPAGER is defined, *less*(1) is used. Only used if `-a` or `-l` is specified.

EXIT STATUS

The **mandoc** utility exits with one of the following values, controlled by the message *level* associated with the `-W` option:

0 No base system convention violations, style suggestions, warnings, or errors occurred, or those that did were ignored because they were lower than the requested *level*.

1 At least one base system convention violation or style suggestion occurred, but no warning or error, and `-W base` or `-W style` was specified.

2 At least one warning occurred, but no error, and `-W warning` or a lower *level* was requested.

3 At least one parsing error occurred, but no unsupported feature was encountered, and `-W error` or a lower *level* was requested.

4 At least one unsupported feature was encountered, and `-W unsupp` or a lower *level* was requested.

5 Invalid command line arguments were specified. No input files have been read.

6 An operating system error occurred, for example exhaustion of memory, file descriptors, or process table entries. Such errors may cause **mandoc** to exit at once, possibly in the middle of parsing or formatting a file.

Note that selecting `-T lint` output mode implies `-W all`.

EXAMPLES

To page manuals to the terminal:

```
$ mandoc -l mandoc.1 man.1 apropos.1 makewhatis.8
```

To produce HTML manuals with */usr/share/misc/mandoc.css* as the style-sheet:

```
$ mandoc -T html -O style=/usr/share/misc/mandoc.css mdoc.7 >
mdoc.7.html
```

To check over a large set of manuals:

```
$ mandoc -T lint `find /usr/src -name \*\.[1-9]`
```

To produce a series of PostScript manuals for A4 paper:

```
$ mandoc -T ps -O paper=a4 mdoc.7 man.7 > manuals.ps
```

Convert a modern *mdoc*(7) manual to the older *man*(7) format, for use on systems lacking an *mdoc*(7) parser:

```
$ mandoc -T man foo.mdoc > foo.man
```

DIAGNOSTICS

Messages displayed by **mandoc** follow this format:

mandoc: *file*:*line*:*column*: *level*: *message*: *macro arguments* (*os*)

The first three fields identify the *file* name, *line* number, and *column* number of the input file where the message was triggered. The line and column numbers start at 1. Both are omitted for messages referring to an input file as a whole. All *level* and *message* strings are explained below. The name of the *macro* triggering the message and its *arguments* are omitted where meaningless. The *os* operating system specifier is omitted for messages that are relevant for all operating systems. Fatal messages about invalid command line arguments or operating system errors, for example when memory is exhausted, may also omit the *file* and *level* fields.

Message levels have the following meanings:

`syserr` An operating system error occurred. There isn't necessarily anything wrong with the input files. Output may all the same be missing or incomplete.

`badarg` Invalid command line arguments were specified. No input files have been read and no output is produced.

`unsupp` An input file uses unsupported low-level *roff*(7) features. The output may be incomplete and/or misformatted, so using GNU troff instead of **mandoc** to process the file may be preferable.

`error` Indicates a risk of information loss or severe misformatting, in most cases caused by serious syntax errors.

`warning` Indicates a risk that the information shown or its formatting may mismatch the author's intent in minor ways. Additionally, syntax errors are classified at least as warnings, even if they do not usually cause misformatting.

`style` An input file uses dubious or discouraged style. This is not a complaint about the syntax, and probably neither formatting nor portability are in danger. While great care is taken to avoid false positives on the higher message levels, the `style` level tries to reduce the probability that issues go unnoticed, so it may occasionally issue bogus suggestions. Please use your good judgement to decide whether any particular `style` suggestion really justifies a change to the input file.

`base` A convention used in the base system of a specific operating system is not adhered to. These are not markup mistakes, and neither the quality of formatting nor portability are in danger. Messages of the `base` level are printed with the more intuitive `style` *`level`* tag.

Messages of the `base`, `style`, `warning`, `error`, and `unsupp` levels are hidden unless their level, or a lower level, is requested using a `-W` option or `-T lint` output mode.

As indicated below, all `base` and some `style` checks are only performed if a specific operating system name occurs in the arguments of the `-W` command line option, of the **Os** macro, of the `-Ios` command line option, or, if neither are present, in the return value of the *uname*(3) function.

Conventions for base system manuals

Mdocdate found
(mdoc, NetBSD) The **Dd** macro uses CVS **Mdocdate** keyword substitution, which is not supported by the NetBSD base system. Consider using the conventional "Month dd, yyyy" format instead.

Mdocdate missing
(mdoc, OpenBSD) The **Dd** macro does not use CVS **Mdocdate** keyword substitution, but using it is conventionally expected in the OpenBSD base system.

unknown architecture
(mdoc, OpenBSD, NetBSD) The third argument of the **Dt** macro does not match any of the architectures this operating system is running on.

operating system explicitly specified
(mdoc, OpenBSD, NetBSD) The **Os** macro has an argument. In the base system, it is conventionally left blank.

RCS id missing
(OpenBSD, NetBSD) The manual page lacks the comment line with the RCS identifier generated by CVS **OpenBSD** or **NetBSD** keyword substitution as conventionally used in these operating systems.

Style suggestions

legacy man(7) date format
(mdoc) The **Dd** macro uses the legacy *man*(7) date format "yyyy-dd-mm". Consider using the conventional *mdoc*(7) date format "Month dd, yyyy" instead.

normalizing date format to: ...
(mdoc, man) The **Dd** or **TH** macro provides an abbreviated month name or a day number with a leading zero. In the formatted output, the month name is written out in full and the leading zero is omitted.

lower case character in document title
(mdoc, man) The title is still used as given in the **Dt** or **TH** macro.

duplicate RCS id
A single manual page contains two copies of the RCS identifier for the same operating system. Consider deleting the later instance and moving the first one up to the top of the page.

possible typo in section name
(mdoc) Fuzzy string matching revealed that the argument of an **Sh** macro is similar, but not identical to a standard section name.

unterminated quoted argument
(roff) Macro arguments can be enclosed in double quote characters such that space characters and macro names contained in the quoted argument need not be escaped. The closing quote of the last argument of a macro can be omitted. However, omitting it is not recommended because it makes the code harder to read.

useless macro
(mdoc) A **Bt**, **Tn**, or **Ud** macro was found. Simply delete it: it serves no useful purpose.

consider using OS macro
(mdoc) A string was found in plain text or in a **Bx** macro that could be represented using **Ox**, **Nx**, **Fx**, or **Dx**.

errnos out of order
(mdoc, NetBSD) The **Er** items in a **Bl** list are not in alphabetical order.

duplicate errno
(mdoc, NetBSD) A **Bl** list contains two consecutive **It** entries describing the same **Er** number.

referenced manual not found
(mdoc) An **Xr** macro references a manual page that was not found. When running with `-W base`, the search is restricted to the base system, by default to */usr/share/man*:*/usr/X11R6/man*. This path can be configured at compile time using the `MANPATH_BASE` preprocessor macro. When running with `-W style`, the search is done along the full search path as described in the *man*(1) manual page, respecting the `-m` and `-M` command line options, the *MANPATH* environment variable, the *man.conf*(5) file and falling back to the default of */usr/share/man*:*/usr/X11R6/man*:*/usr/local/man*, also configurable at compile time using the `MANPATH_DEFAULT` preprocessor macro.

trailing delimiter
(mdoc) The last argument of an **Ex**, **Fo**, **Nd**, **Nm**, **Os**, **Sh**, **Ss**, **St**, or **Sx** macro ends with a trailing delimiter. This is usually bad style and often indicates typos. Most likely, the delimiter can be removed.

no blank before trailing delimiter
(mdoc) The last argument of a macro that supports trailing delimiter arguments is longer than one byte and ends with a trailing delimiter. Consider inserting a blank such that the delimiter becomes a separate argument, thus moving it out of the scope of the macro.

fill mode already enabled, skipping
(man) A **fi** request occurs even though the document is still in fill mode, or already switched back to fill mode. It has no effect.

fill mode already disabled, skipping
(man) An **nf** request occurs even though the document already switched to no-fill mode and did not switch back to fill mode yet. It has no effect.

input text line longer than 80 bytes
Consider breaking the input text line at one of the blank characters before column 80.

verbatim "--", maybe consider using \(em
(mdoc) Even though the ASCII output device renders an em-dash as "−−", that is not a good way to write it in an input file because it renders poorly on all other output devices.

function name without markup
(mdoc) A word followed by an empty pair of parentheses occurs on a text line. Consider using an **Fn** or **Xr** macro.

whitespace at end of input line
(mdoc, man, roff) Whitespace at the end of input lines is almost never semantically significant — but in the odd case where it might be, it is extremely confusing when reviewing and maintaining documents.

bad comment style
(roff) Comment lines start with a dot, a backslash, and a double-quote character. The **mandoc** utility treats the line as a comment line even without the backslash, but leaving out the backslash might not be portable.

Warnings related to the document prologue

missing manual title, using UNTITLED
(mdoc) A **Dt** macro has no arguments, or there is no **Dt** macro before the first non-prologue macro.

missing manual title, using ""
(man) There is no **TH** macro, or it has no arguments.

missing manual section, using ""
(mdoc, man) A **Dt** or **TH** macro lacks the mandatory section argument.

unknown manual section
(mdoc) The section number in a **Dt** line is invalid, but still used.

filename/section mismatch
(mdoc, man) The name of the input file being processed is known and its file name extension starts with a non-zero digit, but the **Dt** or **TH** macro contains a *section* argument that starts with a different non-zero digit. The *section* argument is used as provided anyway. Consider checking whether the file name or the argument need a correction.

missing date, using ""
(mdoc, man) The document was parsed as *mdoc*(7) and it has no **Dd** macro, or the **Dd** macro has no arguments or only empty arguments; or the document was parsed as *man*(7) and it has no **TH** macro, or the **TH** macro has less than three arguments or its third argument is empty.

cannot parse date, using it verbatim
(mdoc, man) The date given in a **Dd** or **TH** macro does not follow the conventional format.

date in the future, using it anyway
(mdoc, man) The date given in a **Dd** or **TH** macro is more than a day ahead of the current system *time*(3).

missing Os macro, using ""
(mdoc) The default or current system is not shown in this case.

late prologue macro
(mdoc) A **Dd** or **Os** macro occurs after some non-prologue macro, but still takes effect.

prologue macros out of order
(mdoc) The prologue macros are not given in the conventional order **Dd**, **Dt**, **Os**. All three macros are used even when given in another order.

Warnings regarding document structure

.so is fragile, better use ln(1)
(roff) Including files only works when the parser program runs with the correct current working directory.

no document body
(mdoc, man) The document body contains neither text nor macros. An empty document is shown, consisting only of a header and a footer line.

content before first section header
(mdoc, man) Some macros or text precede the first **Sh** or **SH** section header. The offending macros and text are parsed and added to the top level of the syntax tree, outside any section block.

first section is not NAME
(mdoc) The argument of the first **Sh** macro is not 'NAME'. This may confuse *makewhatis*(8) and *apropos*(1).

NAME section without Nm before Nd
(mdoc) The NAME section does not contain any **Nm** child macro before the first **Nd** macro.

NAME section without description
(mdoc) The NAME section lacks the mandatory **Nd** child macro.

description not at the end of NAME
(mdoc) The NAME section does contain an **Nd** child macro, but other content follows it.

bad NAME section content
(mdoc) The NAME section contains plain text or macros other than **Nm** and **Nd**.

missing comma before name
(mdoc) The NAME section contains an **Nm** macro that is neither the first one nor preceded by a comma.

missing description line, using ""
(mdoc) The **Nd** macro lacks the required argument. The title line of the manual will end after the dash.

description line outside NAME section
(mdoc) An **Nd** macro appears outside the NAME section. The arguments are printed anyway and the following text is used for *apropos*(1), but none of that behaviour is portable.

sections out of conventional order
(mdoc) A standard section occurs after another section it usually precedes. All section titles are used as given, and the order of sections is not changed.

duplicate section title
(mdoc) The same standard section title occurs more than once.

unexpected section
(mdoc) A standard section header occurs in a section of the manual where it normally isn't useful.

cross reference to self
(mdoc) An **Xr** macro refers to a name and section matching the section of the present manual page and a name mentioned in an **Nm** macro in the NAME or SYNOPSIS section, or in an **Fn** or **Fo** macro in the SYNOPSIS. Consider using **Nm** or **Fn** instead of **Xr**.

unusual Xr order
(mdoc) In the SEE ALSO section, an **Xr** macro with a lower section number follows one with a higher number, or two **Xr** macros referring to the same section are out of alphabetical order.

unusual Xr punctuation
(mdoc) In the SEE ALSO section, punctuation between two **Xr** macros differs from a single comma, or there is trailing punctuation after the last **Xr** macro.

AUTHORS section without An macro
(mdoc) An AUTHORS sections contains no **An** macros, or only empty ones. Probably, there are author names lacking markup.

Warnings related to macros and nesting

obsolete macro
(mdoc) See the *mdoc*(7) manual for replacements.

macro neither callable nor escaped
(mdoc) The name of a macro that is not callable appears on a macro line. It is printed verbatim. If the intention is to call it, move it to its own input line; otherwise, escape it by prepending '\&'.

skipping paragraph macro
In *mdoc*(7) documents, this happens
- at the beginning and end of sections and subsections
- right before non-compact lists and displays
- at the end of items in non-column, non-compact lists
- and for multiple consecutive paragraph macros.

In *man*(7) documents, it happens
- for empty **P**, **PP**, and **LP** macros
- for **IP** macros having neither head nor body arguments
- for **br** or **sp** right after **SH** or **SS**

moving paragraph macro out of list
(mdoc) A list item in a **Bl** list contains a trailing paragraph macro. The paragraph macro is moved after the end of the list.

skipping no-space macro
(mdoc) An input line begins with an **Ns** macro, or the next argument after an **Ns** macro is an isolated closing delimiter. The macro is ignored.

blocks badly nested
(mdoc) If two blocks intersect, one should completely contain the other. Otherwise, rendered output is likely to look strange in any output format, and rendering in SGML-based output formats is likely to be outright wrong because such languages do not support badly nested blocks at all. Typical examples of badly nested blocks are "**Ao Bo Ac Bc**" and "**Ao Bq Ac**". In these examples, **Ac** breaks **Bo** and **Bq**, respectively.

nested displays are not portable
(mdoc) A **Bd**, **D1**, or **Dl** display occurs nested inside another **Bd** display. This works with **mandoc**, but fails with most other implementations.

moving content out of list
(mdoc) A **Bl** list block contains text or macros before the first **It** macro. The offending children are moved before the beginning of the list.

first macro on line
Inside a **Bl** `-column` list, a **Ta** macro occurs as the first macro on a line, which is not portable.

line scope broken
(man) While parsing the next-line scope of the previous macro, another macro is found that prematurely terminates the previous one. The previous, interrupted macro is deleted from the parse tree.

Warnings related to missing arguments

skipping empty request
(roff, eqn) The macro name is missing from a macro definition request, or an *eqn*(7) control statement or operation keyword lacks its required argument.

conditional request controls empty scope
(roff) A conditional request is only useful if any of the following follows it on the same logical input line:
- The '\{' keyword to open a multi-line scope.
- A request or macro or some text, resulting in a single-line scope.

– The immediate end of the logical line without any intervening whitespace, resulting in next-line scope.
Here, a conditional request is followed by trailing whitespace only, and there is no other content on its logical input line. Note that it doesn't matter whether the logical input line is split across multiple physical input lines using '\' line continuation characters. This is one of the rare cases where trailing whitespace is syntactically significant. The conditional request controls a scope containing whitespace only, so it is unlikely to have a significant effect, except that it may control a following **el** clause.

skipping empty macro
(mdoc) The indicated macro has no arguments and hence no effect.

empty block
(mdoc, man) A **Bd**, **Bk**, **Bl**, **D1**, **Dl**, **MT**, **RS**, or **UR** block contains nothing in its body and will produce no output.

empty argument, using 0n
(mdoc) The required width is missing after **Bd** or **Bl** `-offset` or `-width`.

missing display type, using -ragged
(mdoc) The **Bd** macro is invoked without the required display type.

list type is not the first argument
(mdoc) In a **Bl** macro, at least one other argument precedes the type argument. The **mandoc** utility copes with any argument order, but some other *mdoc*(7) implementations do not.

missing -width in -tag list, using 8n
(mdoc) Every **Bl** macro having the `-tag` argument requires `-width`, too.

missing utility name, using ""
(mdoc) The **Ex** `-std` macro is called without an argument before **Nm** has first been called with an argument.

missing function name, using ""
(mdoc) The **Fo** macro is called without an argument. No function name is printed.

empty head in list item
(mdoc) In a **Bl** `-diag`, `-hang`, `-inset`, `-ohang`, or `-tag` list, an **It** macro lacks the required argument. The item head is left empty.

empty list item
(mdoc) In a **Bl** `-bullet`, `-dash`, `-enum`, or `-hyphen` list, an **It** block is empty. An empty list item is shown.

missing argument, using next line
(mdoc) An **It** macro in a **Bd** `-column` list has no arguments. While **mandoc** uses the text or macros of the following line, if any, for the cell, other formatters may misformat the list.

missing font type, using \fR
(mdoc) A **Bf** macro has no argument. It switches to the default font.

unknown font type, using \fR
(mdoc) The **Bf** argument is invalid. The default font is used instead.

nothing follows prefix
(mdoc) A **Pf** macro has no argument, or only one argument and no macro follows on the same input line. This defeats its purpose; in particular, spacing is not suppressed before the text or macros following on the next input line.

empty reference block
(mdoc) An **Rs** macro is immediately followed by an **Re** macro on the next input line. Such an empty block does not produce any output.

missing section argument
(mdoc) An **Xr** macro lacks its second, section number argument. The first argument, i.e. the name, is printed, but without subsequent parentheses.

missing -std argument, adding it
(mdoc) An **Ex** or **Rv** macro lacks the required `-std` argument. The **mandoc** utility assumes `-std` even when it is not specified, but other implementations may not.

missing option string, using ""
(man) The **OP** macro is invoked without any argument. An empty pair of square brackets is shown.

missing resource identifier, using ""
(man) The **MT** or **UR** macro is invoked without any argument. An empty pair of angle brackets is shown.

missing eqn box, using ""
(eqn) A diacritic mark or a binary operator is found, but there is nothing to the left of it. An empty box is inserted.

Warnings related to bad macro arguments

duplicate argument
(mdoc) A **Bd** or **Bl** macro has more than one `-compact`, more than one `-offset`, or more than one `-width` argument. All but the last instances of these arguments are ignored.

skipping duplicate argument
(mdoc) An **An** macro has more than one `-split` or `-nosplit` argument. All but the first of these arguments are ignored.

skipping duplicate display type
(mdoc) A **Bd** macro has more than one type argument; the first one is used.

skipping duplicate list type
(mdoc) A **Bl** macro has more than one type argument; the first one is used.

skipping -width argument
(mdoc) A **Bl** `-column`, `-diag`, `-ohang`, `-inset`, or `-item` list has a `-width` argument. That has no effect.

wrong number of cells
In a line of a **Bl** `-column` list, the number of tabs or **Ta** macros is less than the number expected from the list header line or exceeds the expected number by more than one. Missing cells remain empty, and all cells exceeding the number of columns are joined into one single cell.

unknown AT&T UNIX version
(mdoc) An **At** macro has an invalid argument. It is used verbatim, with "AT&T UNIX " prefixed to it.

comma in function argument
(mdoc) An argument of an **Fa** or **Fn** macro contains a comma; it should probably be split into two arguments.

parenthesis in function name
(mdoc) The first argument of an **Fc** or **Fn** macro contains an opening or closing parenthesis; that's probably wrong, parentheses are added automatically.

unknown library name
(mdoc, not on OpenBSD) An **Lb** macro has an unknown name argument and will be rendered as "library "*name*"".

invalid content in Rs block
(mdoc) An **Rs** block contains plain text or non-% macros. The bogus content is left in the syntax tree. Formatting may be poor.

invalid Boolean argument
(mdoc) An **Sm** macro has an argument other than `on` or `off`. The invalid argument is moved out of the macro, which leaves the macro empty, causing it to toggle the spacing mode.

argument contains two font escapes
(roff) The second argument of a **char** request contains more than one font escape sequence. A wrong font may remain active after using the character.

unknown font, skipping request
(man, tbl) A *roff*(7) **ft** request or a *tbl*(7) **f** layout modifier has an unknown *font* argument.

odd number of characters in request
(roff) A **tr** request contains an odd number of characters. The last character is mapped to the blank character.

Warnings related to plain text

blank line in fill mode, using .sp
(mdoc) The meaning of blank input lines is only well-defined in non-fill mode: In fill mode, line breaks of text input lines are not supposed to be significant. However, for compatibility with groff, blank lines in fill mode are formatted like **sp** requests. To request a paragraph break, use **Pp** instead of a blank line.

tab in filled text
(mdoc, man) The meaning of tab characters is only well-defined in non-fill mode: In fill mode, whitespace is not supposed to be significant on text input lines. As an implementation dependent choice, tab characters on text lines are passed through to the formatters in any case. Given that the text before the tab character will be filled, it is hard to predict which tab stop position the tab will advance to.

new sentence, new line
(mdoc) A new sentence starts in the middle of a text line. Start it on a new input line to help formatters produce correct spacing.

invalid escape sequence
(roff) An escape sequence has an invalid opening argument delimiter, lacks the closing argument delimiter, the argument is of an invalid form, or it is a character escape sequence with an invalid name. If the argument is incomplete, ***** and **\n** expand to an empty string, **\B** to the digit '0', and **\w** to the length of the incomplete argument. All other invalid escape sequences are ignored.

undefined escape, printing literally
(roff) In an escape sequence, the first character right after the leading backslash is invalid. That character is printed literally, which is equivalent to ignoring the backslash.

undefined string, using ""
(roff) If a string is used without being defined before, its value is implicitly set to the empty string. However, defining strings explicitly before use keeps the code more readable.

Warnings related to tables

tbl line starts with span
(tbl) The first cell in a table layout line is a horizontal span ('s'). Data provided for this cell is ignored, and nothing is printed in the cell.

tbl column starts with span
(tbl) The first line of a table layout specification requests a vertical span ('^'). Data provided for this cell is ignored, and nothing is printed in the cell.

skipping vertical bar in tbl layout
(tbl) A table layout specification contains more than two consecutive vertical bars. A double bar is printed, all additional bars are discarded.

Errors related to tables

non-alphabetic character in tbl options
(tbl) The table options line contains a character other than a letter, blank, or comma where the beginning of an option name is expected. The character is ignored.

skipping unknown tbl option
(tbl) The table options line contains a string of letters that does not match any known option name. The word is ignored.

missing tbl option argument
(tbl) A table option that requires an argument is not followed by an opening parenthesis, or the opening parenthesis is immediately followed by a closing parenthesis. The option is ignored.

wrong tbl option argument size
(tbl) A table option argument contains an invalid number of characters. Both the option and the argument are ignored.

empty tbl layout
(tbl) A table layout specification is completely empty, specifying zero lines and zero columns. As a fallback, a single left-justified column is used.

invalid character in tbl layout
(tbl) A table layout specification contains a character that can neither be interpreted as a layout key character nor as a layout modifier, or a modifier precedes the first key. The invalid character is discarded.

unmatched parenthesis in tbl layout
(tbl) A table layout specification contains an opening parenthesis, but no matching closing parenthesis. The rest of the input line, starting from the parenthesis, has no effect.

ignoring excessive spacing in tbl layout
(tbl) A spacing modifier in a table layout is unreasonably large. The default spacing of 3n is used instead.

tbl without any data cells
(tbl) A table does not contain any data cells. It will probably produce no output.

ignoring data in spanned tbl cell
(tbl) A table cell is marked as a horizontal span (‘`s`’) or vertical span (‘`^`’) in the table layout, but it contains data. The data is ignored.

ignoring extra tbl data cells
(tbl) A data line contains more cells than the corresponding layout line. The data in the extra cells is ignored.

data block open at end of tbl
(tbl) A data block is opened with `T{`, but never closed with a matching `T}`. The remaining data lines of the table are all put into one cell, and any remaining cells stay empty.

Errors related to roff, mdoc, and man code

duplicate prologue macro
(mdoc) One of the prologue macros occurs more than once. The last instance overrides all previous ones.

skipping late title macro
(mdoc) The **`Dt`** macro appears after the first non-prologue macro. Traditional formatters cannot handle this because they write the page header before parsing the document body. Even though this technical restriction does not apply to **`mandoc`**, traditional semantics is preserved. The late macro is discarded including its arguments.

input stack limit exceeded, infinite loop?
(roff) Explicit recursion limits are implemented for the following features, in order to prevent infinite loops:
- expansion of nested escape sequences including expansion of strings and number registers,

- expansion of nested user-defined macros,
- and **so** file inclusion.

When a limit is hit, the output is incorrect, typically losing some content, but the parser can continue.

skipping bad character
(mdoc, man, roff) The input file contains a byte that is not a printable *ascii*(7) character. The message mentions the character number. The offending byte is replaced with a question mark ('?'). Consider editing the input file to replace the byte with an ASCII transliteration of the intended character.

skipping unknown macro
(mdoc, man, roff) The first identifier on a request or macro line is neither recognized as a *roff*(7) request, nor as a user-defined macro, nor, respectively, as an *mdoc*(7) or *man*(7) macro. It may be mistyped or unsupported. The request or macro is discarded including its arguments.

skipping request outside macro
(roff) A **shift** or **return** request occurs outside any macro definition and has no effect.

skipping insecure request
(roff) An input file attempted to run a shell command or to read or write an external file. Such attempts are denied for security reasons.

skipping item outside list
(mdoc, eqn) An **It** macro occurs outside any **Bl** list, or an *eqn*(7) **above** delimiter occurs outside any pile. It is discarded including its arguments.

skipping column outside column list
(mdoc) A **Ta** macro occurs outside any **Bl** `-column` block. It is discarded including its arguments.

skipping end of block that is not open
(mdoc, man, eqn, tbl, roff) Various syntax elements can only be used to explicitly close blocks that have previously been opened. An *mdoc*(7) block closing macro, a *man*(7) **ME**, **RE** or **UE** macro, an *eqn*(7) right delimiter or closing brace, or the end of an equation, table, or *roff*(7) conditional request is encountered but no matching block is open. The offending request or macro is discarded.

fewer RS blocks open, skipping
(man) The **RE** macro is invoked with an argument, but less than the specified number of **RS** blocks is open. The **RE** macro is discarded.

inserting missing end of block
(mdoc, tbl) Various *mdoc*(7) macros as well as tables require explicit closing by dedicated macros. A block that doesn't support bad nesting ends before all of its children are properly closed. The open child nodes are closed implicitly.

appending missing end of block
(mdoc, man, eqn, tbl, roff) At the end of the document, an explicit *mdoc*(7) block, a *man*(7) next-line scope or **MT**, **RS** or **UR** block, an equation, table, or *roff*(7) conditional or ignore block is still open. The open block is closed implicitly.

escaped character not allowed in a name
(roff) Macro, string and register identifiers consist of printable, non-whitespace ASCII characters. Escape sequences and characters and strings expressed in terms of them cannot form part of a name. The first argument of an **am**, **as**, **de**, **ds**, **nr**, or **rr** request, or any argument of an **rm** request, or the name of a request or user defined macro being called, is terminated by an escape sequence. In the cases of **as**, **ds**, and **nr**, the request has no effect at all. In the cases of **am**, **de**, **rr**, and **rm**, what was parsed up to this point is used as the arguments to the request, and the rest of the input line is discarded including the escape sequence. When parsing for a request or a user-defined macro name to be called, only the escape sequence is discarded. The characters preceding it are used as the request or macro name, the characters following it are used as the arguments to the request or macro.

using macro argument outside macro
(roff) The escape sequence \$ occurs outside any macro definition and expands to the empty string.

argument number is not numeric
(roff) The argument of the escape sequence \$ is not a digit; the escape sequence expands to the empty string.

NOT IMPLEMENTED: Bd -file
(mdoc) For security reasons, the **Bd** macro does not support the `-file` argument. By requesting the inclusion of a sensitive file, a malicious document might otherwise trick a privileged user into inadvertently displaying the file on the screen, revealing the file content to bystanders. The argument is ignored including the file name following it.

skipping display without arguments
(mdoc) A **Bd** block macro does not have any arguments. The block is discarded, and the block content is displayed in whatever mode was active before the block.

missing list type, using -item
(mdoc) A **Bl** macro fails to specify the list type.

argument is not numeric, using 1
(roff) The argument of a **ce** request is not a number.

argument is not a character
(roff) The first argument of a **char** request is neither a single ASCII character nor a single character escape sequence. The request is ignored including all its arguments.

missing manual name, using ""
(mdoc) The first call to **Nm**, or any call in the NAME section, lacks the required argument.

uname(3) system call failed, using UNKNOWN
(mdoc) The **Os** macro is called without arguments, and the *uname*(3) system call failed. As a workaround, **mandoc** can be compiled with `-DOSNAME="\"`*string*`\""`.

unknown standard specifier
(mdoc) An **St** macro has an unknown argument and is discarded.

skipping request without numeric argument
(roff, eqn) An **it** request or an *eqn*(7) **size** or **gsize** statement has a non-numeric or negative argument or no argument at all. The invalid request or statement is ignored.

excessive shift
(roff) The argument of a **shift** request is larger than the number of arguments of the macro that is currently being executed. All macro arguments are deleted and \n(.$ is set to zero.

NOT IMPLEMENTED: .so with absolute path or ".."
(roff) For security reasons, **mandoc** allows **so** file inclusion requests only with relative paths and only without ascending to any parent directory. By requesting the inclusion of a sensitive file, a malicious document might otherwise trick a privileged user into inadvertently displaying the file on the screen, revealing the file content to bystanders. **mandoc** only shows the path as it appears behind **so**.

.so request failed
(roff) Servicing a **so** request requires reading an external file, but the file could not be opened. **mandoc** only shows the path as it appears behind **so**.

skipping all arguments
(mdoc, man, eqn, roff) An *mdoc*(7) **Bt**, **Ed**, **Ef**, **Ek**, **El**, **Lp**, **Pp**, **Re**, **Rs**, or **Ud** macro, an **It** macro in a list that don't support item heads, a *man*(7) **LP**, **P**, or **PP** macro, an *eqn*(7) **EQ** or **EN** macro, or a *roff*(7) **br**, **fi**, or **nf** request or '..' block closing request is invoked with at least one argument. All arguments are ignored.

skipping excess arguments
(mdoc, man, roff) A macro or request is invoked with too many arguments:
- **Fo**, **MT**, **PD**, **RS**, **UR**, **ft**, or **sp** with more than one argument
- **An** with another argument after `-split` or `-nosplit`
- **RE** with more than one argument or with a non-integer argument
- **OP** or a request of the **de** family with more than two arguments
- **Dt** with more than three arguments
- **TH** with more than five arguments
- **Bd**, **Bk**, or **Bl** with invalid arguments

The excess arguments are ignored.

Unsupported features

input too large
(mdoc, man) Currently, **mandoc** cannot handle input files larger than its arbitrary size limit of 2^31 bytes (2 Gigabytes). Since useful manuals are always small, this is not a problem in practice. Parsing is aborted as soon as the condition is detected.

unsupported control character
(roff) An ASCII control character supported by other *roff*(7) implementations but not by **mandoc** was found in an input file. It is replaced by a question mark.

unsupported escape sequence
(roff) An input file contains an escape sequence supported by GNU troff or Heirloom troff but not by **mandoc**, and it is likely that this will cause information loss or considerable misformatting.

unsupported roff request
(roff) An input file contains a *roff*(7) request supported by GNU troff or Heirloom troff but not by **mandoc**, and it is likely that this will cause information loss or considerable misformatting.

eqn delim option in tbl
(eqn, tbl) The options line of a table defines equation delimiters. Any equation source code contained in the table will be printed unformatted.

unsupported table layout modifier
(tbl) A table layout specification contains an ‘`m`’ modifier. The modifier is discarded.

ignoring macro in table
(tbl, mdoc, man) A table contains an invocation of an *mdoc*(7) or *man*(7) macro or of an undefined macro. The macro is ignored, and its arguments are handled as if they were a text line.

skipping tbl in -Tman mode
(mdoc, tbl) An input file contains the **TS** macro. This message is only generated in `-T man` output mode, where *tbl*(7) input is not supported.

skipping eqn in -Tman mode
(mdoc, eqn) An input file contains the **EQ** macro. This message is only generated in `-T man` output mode, where *eqn*(7) input is not supported.

Bad command line arguments

bad command line argument
The argument following one of the `-IKMmOTW` command line options is invalid, or a *file* given as a command line argument cannot be opened.

duplicate command line argument
The `-I` command line option was specified twice.

option has a superfluous value
An argument to the `-O` option has a value but does not accept one.

missing option value
An argument to the `-O` option has no argument but requires one.

bad option value
An argument to the `-O indent` or `width` option has an invalid value.

duplicate option value
The same `-O` option is specified more than once.

no such tag
The `-O tag` option was specified but the tag was not found in any of the displayed manual pages.

-Tmarkdown unsupported for man(7) input
(man) The `-T markdown` option was specified but an input file uses the *man*(7) language. No output is produced for that input file.

SEE ALSO

apropos(1), *man*(1), *eqn*(7), *man*(7), *mandoc_char*(7), *mdoc*(7), *roff*(7), *tbl*(7)

HISTORY

The **mandoc** utility first appeared in OpenBSD 4.8. The option `-I` appeared in OpenBSD 5.2, and `-aCcfhKklMSsw` in OpenBSD 5.7.

AUTHORS

The **mandoc** utility was written by Kristaps Dzonsons <kristaps@bsd.lv> and is maintained by Ingo Schwarze <schwarze@openbsd.org>.

NAME

soelim — interpret .so requests in manpages

SYNOPSIS

soelim [−Crtv] [−I *dir*] [*files* ...]

DESCRIPTION

soelim reads *files* lines by lines.

If a line starts by: ".so anotherfile" it replace the line by processing "anotherfile". Otherwise the line is printed to stdout.

−C Recognise *.so* when not followed by a space character.

−r Compatibility with GNU groff's **soelim** (does nothing).

−t Compatibility with GNU groff's **soelim** (does nothing).

−v Compatibility with GNU groff's **soelim** (does nothing).

−I *dir*
: This option specify directories where **soelim** searches for files (both those on the command line and those named in ".so" directive.) This options may be specified multiple times. The directories will be searched in the order specified.

The files are always searched first in the current directory.

A file specified with an absolute path will be opened directly without performing a search.

SEE ALSO

mandoc(1)

AUTHORS

This version of the **soelim** utility was written by Baptiste Daroussin <bapt@freebsd.org>.

NAME

man.cgi — internals of the CGI program to search and display manual pages

DESCRIPTION

The source code of *man.cgi*(8) is organized in four levels:

1. "Top level"
2. "Page generators"
3. "Result generators"
4. "Utility routines"

Top level

The top level of *man.cgi*(8) consists of the **main**() program and a few parser routines.

int **main**(*void*)
The main program
- limits execution time;
- changes to MAN_DIR, the data directory containing all the manual trees;
- calls **parse_manpath_conf**();
- if *PATH_INFO* is empty, calls **parse_query_string**(); otherwise, calls **parse_path_info**();
- validates the manpath and the architecture;
- calls the appropriate one among the "Page generators".

void **parse_manpath_conf**(*struct req *req*)
Parses and validates *manpath.conf* and fills *req->p* and *req->psz*.

void **parse_path_info**(*struct req *req*, *const char *path*)
Parses and validates *PATH_INFO*, clears *req->isquery*, and fills *req->q*.

void **parse_query_string**(*struct req *req*, *const char *qs*)
Parses and validates *QUERY_STRING*, sets *req->isquery*, and fills *req->q*. This function is the only user of the utility functions **http_decode**() and **set_query_attr**().

Page generators

The purpose of each page generator is to print a complete HTML page, starting with the HTTP headers and continuing to the page footer. Before starting HTML output with **resp_begin_html**(), some page generators do some preparatory work, for example to decide which page to show. Each page generator ends with a call to **resp_end_html**().

void **pg_show**(*struct req *req*, *const char *fullpath*)
This page generator is used when *PATH_INFO* contains the complete path to a manual page including manpath, section directory, optional architecture subdirectory, manual name and section number suffix. It validates the manpath, changes into it, validate the filename, and then calls **resp_begin_html**(), **resp_searchform**(), **resp_show**(), and **resp_end_html**() in that order.

void **pg_search**(*const struct req *req*)
This page generator is used when *PATH_INFO* contains a search query in short format or when *PATH_INFO* is empty and a *QUERY_STRING* is provided. If possible, requests using *QUERY_STRING* are redirected to URIs using *PATH_INFO* by calling **pg_redirect**(). Otherwise, it changes into the manpath and calls *mansearch*(3). Depending on the result, it calls either **pg_noresult**() or **pg_searchres**().

void **pg_redirect**(*const struct req *req*, *const char *name*)
This function is special in so far as it does not print an HTML page, but only an HTTP 303 response with a Location: of the form: http://*host*/[*scriptname*/][*manpath*/][*arch*/]*name*[.*sec*]

void **pg_noresult**(*const struct req *req*, *const char *msg*)
This function calls **resp_begin_html**(), **resp_searchform**(), prints the *msg* passed to it, and calls **resp_end_html**().

void **pg_searchres**(*const struct req *req*, *struct manpage *r*, *size_t sz*)
This function first validates the filenames found. If *QUERY_STRING* was used and there is exactly one result, it writes an HTTP redirect to that result. Otherwise, it writes an HTML result page beginning with **resp_begin_html**() and **resp_searchform**(). If there is more than one result, it writes a list of links to all the results. If it was a *man*(1) rather than an *apropos*(1) query or if there is only one single result, it calls **resp_show**(). Finally, it calls **resp_end_html**().

void **pg_index**(*const struct req *req*)
This page generator is used when *PATH_INFO* and *QUERY_STRING* are both empty. It calls **resp_begin_html**() and **resp_searchform**(), writes links to help pages, and calls **resp_end_html**().

void **pg_error_badrequest**(*const char *msg*)
This page generator is used when **main**() or **pg_show**() detect an invalid URI. It calls **resp_begin_html**(), prints the *msg* provided, and calls **resp_end_html**().

void **pg_error_internal**(*void*)
This page generator is used by various functions when errors are detected in the *manpath.conf* configuration file, in *mandoc.db*(5) databases, in the *mandoc*(3) parser, in file system permissions, or when setting up timeouts. It calls **resp_begin_html**(), prints "Internal Server Error", and calls **resp_end_html**(). Before calling **pg_error_internal**(), call *warn*(3) or *warnx*(3) to log the reason of the error to the *httpd*(8) server log file.

Result generators
The purpose of result generators is to print a chunk of HTML code. When they print untrusted strings or characters, **html_print**() and **html_putchar**() are used. The highest level result generators are:

void **resp_begin_html**(*int code*, *const char *msg*, *const char *file*)
This generator calls **resp_begin_http**() to print the HTTP headers, then prints the HTML header up to the opening tag of the <body> element, then copies the file *header.html* to the output, if it exists and is readable. If *file* is not NULL, it is used for the <title> element.

void **resp_searchform**(*const struct req *req*, *enum focus focus*)
This generator prints a search form, filling it with data from the provided request object. If the *focus* argument is FOCUS_QUERY, it sets the document's autofocus to the query input box.

void **resp_show**(*const struct req *req*, *const char *file*)
This wrapper dispatches to either **resp_catman**() or **resp_format**(), depending on whether *file* starts with *cat* or *man*, respectively.

void **resp_catman**(*const struct req *req*, *const char *file*)
This generator translates a preformatted, backspace-encoded manual page to HTML and prints it to the output.

void **resp_format**(*const struct req *req*, *const char *file*)
This generator formats a manual page on the standard output, using the functions documented in *mchars_alloc*(3) and *mandoc*(3).

void **resp_end_html**(*void*)
This generator copies the file *footer.html* to the output, if it exists and is readable, and closes the <body> and <html> elements.

Utility routines
These functions take a string and return 1 if it is valid, or 0 otherwise.

int **validate_urifrag**(*const char *frag*)
Checks that the string only contains alphanumeric ASCII characters, dashes, dots, slashes, and underscores.

int **validate_manpath**(*const struct req *req*, *const char* manpath*)
Checks that the string is either "mandoc" or one of the manpaths configured in *manpath.conf*.

int **validate_filename**(*const char *file*)
Checks that the string starts with "man" or "cat" and does not ascend to parent directories.

SEE ALSO
mandoc(3), *mansearch*(3), *mchars_alloc*(3), *mandoc.db*(5), *man.cgi*(8)

NAME

mandoc, deroff, mparse_alloc, mparse_copy, mparse_free, mparse_open, mparse_readfd, mparse_reset, mparse_result — mandoc macro compiler library

SYNOPSIS

```
#include <sys/types.h>
#include <stdio.h>
#include <mandoc.h>

#define ASCII_NBRSP
#define ASCII_HYPH
#define ASCII_BREAK

struct mparse *
mparse_alloc(int options, enum mandoc_os oe_e, char *os_s);

void
mparse_free(struct mparse *parse);

void
mparse_copy(const struct mparse *parse);

int
mparse_open(struct mparse *parse, const char *fname);

void
mparse_readfd(struct mparse *parse, int fd, const char *fname);

void
mparse_reset(struct mparse *parse);

struct roff_meta *
mparse_result(struct mparse *parse);

#include <roff.h>

void
deroff(char **dest, const struct roff_node *node);

#include <sys/types.h>
#include <mandoc.h>
#include <mdoc.h>

extern const char * const * mdoc_argnames;
extern const char * const * mdoc_macronames;

#include <sys/types.h>
#include <mandoc.h>
#include <man.h>

extern const char * const * man_macronames;
```

DESCRIPTION

The **mandoc** library parses a Unix manual into an abstract syntax tree (AST). Unix manuals are composed of *mdoc*(7) or *man*(7), and may be mixed with *roff*(7), *tbl*(7), and *eqn*(7) invocations.

The following describes a general parse sequence:

1. initiate a parsing sequence with *mchars_alloc*(3) and **mparse_alloc**();
2. open a file with *open*(2) or **mparse_open**();
3. parse it with **mparse_readfd**();

4. close it with *close*(2);
5. retrieve the syntax tree with **mparse_result**();
6. if information about the validity of the input is needed, fetch it with **mparse_updaterc**();
7. iterate over parse nodes with starting from the *first* member of the returned *struct roff_meta*;
8. free all allocated memory with **mparse_free**() and *mchars_free*(3), or invoke **mparse_reset**() and go back to step 2 to parse new files.

REFERENCE

This section documents the functions, types, and variables available via <*mandoc.h*>, with the exception of those documented in *mandoc_escape*(3) and *mchars_alloc*(3).

Types

enum mandocerr
An error or warning message during parsing.

enum mandoclevel
A classification of an *enum mandocerr* as regards system operation. See the DIAGNOSTICS section in *mandoc*(1) regarding the meanings of the levels.

struct mparse
An opaque pointer to a running parse sequence. Created with **mparse_alloc**() and freed with **mparse_free**(). This may be used across parsed input if **mparse_reset**() is called between parses.

Functions

deroff()
Obtain a text-only representation of a *struct roff_node*, including text contained in its child nodes. To be used on children of the *first* member of *struct roff_meta*. When it is no longer needed, the pointer returned from **deroff**() can be passed to *free*(3).

mparse_alloc()
Allocate a parser. The arguments have the following effect:

options When the MPARSE_MDOC or MPARSE_MAN bit is set, only that parser is used. Otherwise, the document type is automatically detected.

When the MPARSE_SO bit is set, *roff*(7) **so** file inclusion requests are always honoured. Otherwise, if the request is the only content in an input file, only the file name is remembered, to be returned in the *sodest* field of *struct roff_meta*.

When the MPARSE_QUICK bit is set, parsing is aborted after the NAME section. This is for example useful in *makewhatis*(8) -Q to quickly build minimal databases.

When the MARSE_VALIDATE bit is set, **mparse_result**() runs the validation functions before returning the syntax tree. This is almost always required, except in certain debugging scenarios, for example to dump unvalidated syntax trees.

os_e Operating system to check base system conventions for. If MANDOC_OS_OTHER, the system is automatically detected from **Os**, -Ios, or *uname*(3).

os_s A default string for the *mdoc*(7) **Os** macro, overriding the OSNAME preprocessor definition and the results of *uname*(3). Passing NULL sets no default.

The same parser may be used for multiple files so long as **mparse_reset**() is called between parses. **mparse_free**() must be called to free the memory allocated by this function. Declared in <*mandoc.h*>, implemented in *read.c*.

mparse_free()
Free all memory allocated by **mparse_alloc**(). Declared in <*mandoc.h*>, implemented in *read.c*.

mparse_copy()
Dump a copy of the input to the standard output; used for `-man -Tman`. Declared in <*mandoc.h*>, implemented in *read.c*.

mparse_open()
Open the file for reading. If that fails and *fname* does not already end in `.gz`, try again after appending `.gz`. Save the information whether the file is zipped or not. Return a file descriptor open for reading or -1 on failure. It can be passed to **mparse_readfd**() or used directly. Declared in <*mandoc.h*>, implemented in *read.c*.

mparse_readfd()
Parse a file descriptor opened with *open*(2) or **mparse_open**(). Pass the associated filename in *fname*. This function may be called multiple times with different parameters; however, *close*(2) and **mparse_reset**() should be invoked between parses. Declared in <*mandoc.h*>, implemented in *read.c*.

mparse_reset()
Reset a parser so that **mparse_readfd**() may be used again. Declared in <*mandoc.h*>, implemented in *read.c*.

mparse_result()
Obtain the result of a parse. Declared in <*mandoc.h*>, implemented in *read.c*.

Variables

man_macronames
The string representation of a *man*(7) macro as indexed by `enum mant`.

mdoc_argnames
The string representation of an *mdoc*(7) macro argument as indexed by `enum mdocargt`.

mdoc_macronames
The string representation of an *mdoc*(7) macro as indexed by `enum mdoct`.

IMPLEMENTATION NOTES

This section consists of structural documentation for *mdoc*(7) and *man*(7) syntax trees and strings.

Man and Mdoc Strings

Strings may be extracted from mdoc and man meta-data, or from text nodes (MDOC_TEXT and MAN_TEXT, respectively). These strings have special non-printing formatting cues embedded in the text itself, as well as *roff*(7) escapes preserved from input. Implementing systems will need to handle both situations to produce human-readable text. In general, strings may be assumed to consist of 7-bit ASCII characters.

The following non-printing characters may be embedded in text strings:

`ASCII_NBRSP`
: A non-breaking space character.

`ASCII_HYPH`
: A soft hyphen.

`ASCII_BREAK`
: A breakable zero-width space.

Escape characters are also passed verbatim into text strings. An escape character is a sequence of characters beginning with the backslash (‘\’). To construct human-readable text, these should be intercepted with *mandoc_escape*(3) and converted with one the functions described in *mchars_alloc*(3).

Man Abstract Syntax Tree

This AST is governed by the ontological rules dictated in *man*(7) and derives its terminology accordingly.

The AST is composed of `struct roff_node` nodes with element, root and text types as declared by the *type* field. Each node also provides its parse point (the *line*, *pos*, and *sec* fields), its position in the tree (the *parent*, *child*, *next* and *prev* fields) and some type-specific data.

The tree itself is arranged according to the following normal form, where capitalised non-terminals represent nodes.

ROOT	← mnode+
mnode	← ELEMENT | TEXT | BLOCK
BLOCK	← HEAD BODY
HEAD	← mnode∗
BODY	← mnode∗
ELEMENT	← ELEMENT | TEXT∗
TEXT	← [[:ascii:]]∗

The only elements capable of nesting other elements are those with next-line scope as documented in *man*(7).

Mdoc Abstract Syntax Tree

This AST is governed by the ontological rules dictated in *mdoc*(7) and derives its terminology accordingly. "In-line" elements described in *mdoc*(7) are described simply as "elements".

The AST is composed of `struct roff_node` nodes with block, head, body, element, root and text types as declared by the *type* field. Each node also provides its parse point (the *line*, *pos*, and *sec* fields), its position in the tree (the *parent*, *child*, *last*, *next* and *prev* fields) and some type-specific data, in particular, for nodes generated from macros, the generating macro in the *tok* field.

The tree itself is arranged according to the following normal form, where capitalised non-terminals represent nodes.

ROOT	← mnode+
mnode	← BLOCK | ELEMENT | TEXT
BLOCK	← HEAD [TEXT] (BODY [TEXT])+ [TAIL [TEXT]]
ELEMENT	← TEXT∗
HEAD	← mnode∗
BODY	← mnode∗ [ENDBODY mnode∗]
TAIL	← mnode∗
TEXT	← [[:ascii:]]∗

Of note are the TEXT nodes following the HEAD, BODY and TAIL nodes of the BLOCK production: these refer to punctuation marks. Furthermore, although a TEXT node will generally have a non-zero-length string, in the specific case of '.Bd −literal', an empty line will produce a zero-length string. Multiple body parts are only found in invocations of 'Bl −column', where a new body introduces a new phrase.

The *mdoc*(7) syntax tree accommodates for broken block structures as well. The ENDBODY node is available to end the formatting associated with a given block before the physical end of that block. It has a non-null *end* field, is of the BODY *type*, has the same *tok* as the BLOCK it is ending, and has a *pending* field pointing to that BLOCK's BODY node. It is an indirect child of that BODY node and has no children of its own.

An ENDBODY node is generated when a block ends while one of its child blocks is still open, like in the following example:

```
.Ao ao
.Bo bo ac
.Ac bc
.Bc end
```

This example results in the following block structure:

```
BLOCK Ao
    HEAD Ao
    BODY Ao
        TEXT ao
        BLOCK Bo, pending -> Ao
            HEAD Bo
            BODY Bo
```

```
                    TEXT bo
                    TEXT ac
                    ENDBODY Ao, pending -> Ao
                    TEXT bc
TEXT end
```

Here, the formatting of the **Ao** block extends from TEXT ao to TEXT ac, while the formatting of the **Bo** block extends from TEXT bo to TEXT bc. It renders as follows in `-Tascii` mode:

```
<ao [bo ac> bc] end
```

Support for badly-nested blocks is only provided for backward compatibility with some older *mdoc*(7) implementations. Using badly-nested blocks is *strongly discouraged*; for example, the `-Thtml` front-end to *mandoc*(1) is unable to render them in any meaningful way. Furthermore, behaviour when encountering badly-nested blocks is not consistent across troff implementations, especially when using multiple levels of badly-nested blocks.

SEE ALSO

mandoc(1), *man.cgi*(3), *mandoc_escape*(3), *mandoc_headers*(3), *mandoc_malloc*(3), *mansearch*(3), *mchars_alloc*(3), *tbl*(3), *eqn*(7), *man*(7), *mandoc_char*(7), *mdoc*(7), *roff*(7), *tbl*(7)

AUTHORS

The **mandoc** library was written by Kristaps Dzonsons <kristaps@bsd.lv> and is maintained by Ingo Schwarze <schwarze@openbsd.org>.

NAME

mandoc_escape — parse roff escape sequences

SYNOPSIS

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

enum mandoc_esc
mandoc_escape(const char **end, const char **start, int *sz);
```

DESCRIPTION

This function scans a *roff*(7) escape sequence.

An escape sequence consists of

- an initial backslash character (‘\’),
- a single ASCII character called the escape sequence identifier,
- and, with only a few exceptions, an argument.

Arguments can be given in the following forms; some escape sequence identifiers only accept some of these forms as specified below. The first three forms are called the standard forms.

In brackets: **[***argument***]**
: The argument starts after the initial ‘[’, ends before the final ‘]’, and the escape sequence ends with the final ‘]’.

Two-character argument short form: **(***ar*
: This form can only be used for arguments consisting of exactly two characters. It has the same effect as **[***ar***]**.

One-character argument short form: ***a***
: This form can only be used for arguments consisting of exactly one character. It has the same effect as **[***a***]**.

Delimited form: ***C****argument****C***
: The argument starts after the initial delimiter character ***C***, ends before the next occurrence of the delimiter character ***C***, and the escape sequence ends with that second ***C***. Some escape sequences allow arbitrary characters ***C*** as quoting characters, some restrict the range of characters that can be used as quoting characters.

Upon function entry, ***end*** is expected to point to the escape sequence identifier. The values passed in as ***start*** and ***sz*** are ignored and overwritten.

By design, this function cannot handle those *roff*(7) escape sequences that require in-place expansion, in particular user-defined strings *****, number registers **\n**, width measurements **\w**, and numerical expression control **\B**. These are handled by **roff_res**(), a private preprocessor function called from **roff_parseln**(), see the file *roff.c*.

The function **mandoc_escape**() is used

- recursively by itself, because some escape sequence arguments can in turn contain other escape sequences,
- for error detection internally by the *roff*(7) parser part of the *mandoc*(3) library, see the file *roff.c*,
- above all externally by the *mandoc*(1) formatting modules, in particular `-Tascii` and `-Thtml`, for formatting purposes, see the files *term.c* and *html.c*,
- and rarely externally by high-level utilities using the mandoc library, for example *makewhatis*(8), to purge escape sequences from text.

RETURN VALUES

Upon function return, the pointer ***end*** is set to the character after the end of the escape sequence, such that the calling higher-level parser can easily continue.

For escape sequences taking an argument, the pointer ***start*** is set to the beginning of the argument and ***sz*** is set to the length of the argument. For escape sequences not taking an argument, ***start*** is set to the character after the end of the sequence and ***sz*** is set to 0. Both ***start*** and ***sz*** may be `NULL`; in that case, the

argument and the length are not returned.

For sequences taking an argument, the function **mandoc_escape**() returns one of the following values:

ESCAPE_FONT
: The escape sequence **\f** taking an argument in standard form: **\f[**, **\f(**, **\f***a*. Two-character arguments starting with the character 'C' are reduced to one-character arguments by skipping the 'C'. More specific values are returned for the most commonly used arguments:

argument	return value
R or 1	ESCAPE_FONTROMAN
I or 2	ESCAPE_FONTITALIC
B or 3	ESCAPE_FONTBOLD
P	ESCAPE_FONTPREV
BI	ESCAPE_FONTBI

ESCAPE_SPECIAL
: The escape sequence **\C** taking an argument delimited with the single quote character and, as a special exception, the escape sequences *not* having an identifier, that is, those where the argument, in standard form, directly follows the initial backslash: **\C'**, **\[**, **\(**, *a*. Note that the one-character argument short form can only be used for argument characters that do not clash with escape sequence identifiers.

  If the argument matches one of the forms described below under ESCAPE_UNICODE, that value is returned instead.

  The ESCAPE_SPECIAL special character escape sequences can be rendered using the functions **mchars_spec2cp**() and **mchars_spec2str**() described in the *mchars_alloc*(3) manual.

ESCAPE_UNICODE
: Escape sequences of the same format as described above under ESCAPE_SPECIAL, but with an argument of the forms **u***XXXX*, **u***YXXXX*, or **u10***XXXX* where *X* and *Y* are hexadecimal digits and *Y* is not zero: **\C'u**, **\[u**. As a special exception, *start* is set to the character after the **u**, and the *sz* return value does not include the **u** either.

  Such Unicode character escape sequences can be rendered using the function **mchars_num2uc**() described in the *mchars_alloc*(3) manual.

ESCAPE_NUMBERED
: The escape sequence **\N** followed by a delimited argument. The delimiter character is arbitrary except that digits cannot be used. If a digit is encountered instead of the opening delimiter, that digit is considered to be the argument and the end of the sequence, and ESCAPE_IGNORE is returned.

  Such ASCII character escape sequences can be rendered using the function **mchars_num2char**() described in the *mchars_alloc*(3) manual.

ESCAPE_OVERSTRIKE
: The escape sequence **\o** followed by an argument delimited by an arbitrary character.

ESCAPE_IGNORE

- The escape sequence **\s** followed by an argument in standard form or by an argument delimited by the single quote character: **\s'**, **\s[**, **\s(**, **\s***a*. As a special exception, an optional '+' or '−' character is allowed after the 's' for all forms.
- The escape sequences **\F**, **\g**, **\k**, **\M**, **\m**, **\n**, **\V**, and **\Y** followed by an argument in standard form.
- The escape sequences **\A**, **\b**, **\D**, **\R**, **\X**, and **\Z** followed by an argument delimited by an arbitrary character.
- The escape sequences **\H**, **\h**, **\L**, **\l**, **\S**, **\v**, and **\x** followed by an argument delimited by a character that cannot occur in numerical expressions. However, if any character that can occur in numerical expressions is found instead of a delimiter, the sequence is considered to end with that character, and ESCAPE_ERROR is returned.

`ESCAPE_ERROR`
: Escape sequences taking an argument but not matching any of the above patterns. In particular, that happens if the end of the logical input line is reached before the end of the argument.

For sequences that do not take an argument, the function **mandoc_escape**() returns one of the following values:

`ESCAPE_SKIPCHAR`
: The escape sequence "\z".

`ESCAPE_NOSPACE`
: The escape sequence "\c".

`ESCAPE_IGNORE`
: The escape sequences "\d" and "\u".

FILES

This function is implemented in *mandoc.c*.

SEE ALSO

mchars_alloc(3), *mandoc_char*(7), *roff*(7)

HISTORY

This function has been available since mandoc 1.11.2.

AUTHORS

Kristaps Dzonsons <kristaps@bsd.lv>
Ingo Schwarze <schwarze@openbsd.org>

BUGS

The function doesn't cleanly distinguish between sequences that are valid and supported, valid and ignored, valid and unsupported, syntactically invalid, or undefined. For sequences that are ignored or unsupported, it doesn't tell whether that deficiency is likely to cause major formatting problems and/or loss of document content. The function is already rather complicated and still parses some sequences incorrectly.

NAME

mandoc_headers — ordering of mandoc include files

DESCRIPTION

To support a cleaner coding style, the mandoc header files do not contain any include directives and do not guard against multiple inclusion. The application developer has to make sure that the headers are included in a proper order, and that no header is included more than once.

The headers and functions form three major groups: “Parser interface”, “Parser internals”, and “Formatter interface”.

Various rules are given below prohibiting the inclusion of certain combinations of headers into the same file. The intention is to keep the following functional components separate from each other:

- *roff*(7) parser
- *mdoc*(7) parser
- *man*(7) parser
- *tbl*(7) parser
- *eqn*(7) parser
- terminal formatters
- HTML formatters
- search tools
- main programs

Note that mere usage of an opaque struct type does *not* require inclusion of the header where that type is defined.

Parser interface

Each of the following headers can be included without including any other mandoc header. These headers should be included before any other mandoc headers.

"*mandoc_aux.h*"

Memory allocation utility functions; can be used everywhere.

Requires <*sys/types.h*> for `size_t`.

Provides the functions documented in *mandoc_malloc*(3).

"*mandoc_ohash.h*"

Hashing utility functions; can be used everywhere.

Requires <*stddef.h*> for `ptrdiff_t` and <*stdint.h*> for `uint32_t`.

Includes <*ohash.h*> and provides **mandoc_ohash_init**().

"*mandoc.h*"

Error handling, escape sequence, and character utilities; can be used everywhere.

Requires <*sys/types.h*> for `size_t` and <*stdio.h*> for `FILE`.

Provides `enum mandoc_esc`, `enum mandocerr`, `enum mandoclevel`, the function *mandoc_escape*(3), the functions described in *mchars_alloc*(3), and the **mandoc_msg**∗() functions.

"*roff.h*" Common data types for all syntax trees and related functions; can be used everywhere.

Provides `enum mandoc_os`, `enum mdoc_endbody`, `enum roff_macroset`, `enum roff_sec`, `enum roff_tok`, `enum roff_type`, `struct roff_man`, `struct roff_meta`, `struct roff_node`, the constant array *roff_name* and the function **deroff**().

Uses pointers to the types `struct ohash` from "*mandoc_ohash.h*", `struct mdoc_arg` and `union mdoc_data` from "*mdoc.h*", `struct tbl_span` from "*tbl.h*", and `struct eqn_box` from "*eqn.h*" as opaque struct members.

"*tbl.h*" Data structures for the *tbl*(7) parse tree; can be used everywhere.

Requires <*sys/types.h*> for `size_t` and "*mandoc.h*" for `enum mandoc_esc`.

Provides `enum tbl_cellt`, `enum tbl_datt`, `enum tbl_spant`, `struct tbl_opts`, `struct tbl_cell`, `struct tbl_row`, `struct tbl_dat`, and `struct tbl_span`.

"*eqn.h*"
Data structures for the *eqn*(7) parse tree; can be used everywhere.

Requires <*sys/types.h*> for `size_t`.

Provides `enum eqn_boxt`, `enum eqn_fontt`, `enum eqn_post`, and `struct eqn_box`.

"*mandoc_parse.h*"
Top level parser interface, for use in the main program and in the main parser, but not in formatters.

Requires "*mandoc.h*" for `enum mandocerr` and `enum mandoclevel` and "*roff.h*" for `enum mandoc_os`.

Uses the opaque type `struct mparse` from *read.c* for function prototypes. Uses `struct roff_meta` from "*roff.h*" as an opaque type for function prototypes.

"*mandoc_xr.h*"
Cross reference validation; intended for use in the main program and in parsers, but not in formatters.

Provides `struct mandoc_xr` and the functions **mandoc_xr_reset**(), **mandoc_xr_add**(), **mandoc_xr_get**(), and **mandoc_xr_free**().

"*tag.h*" Internal interfaces to tag syntax tree nodes, for use by validation modules only.

Requires <*limits.h*> for `INT_MAX`.

Provides the functions **tag_alloc**(), **tag_put**(), **tag_check**(), and **tag_free**() and some `TAG_*` constants.

Uses the type `struct roff_node` from "*roff.h*" as an opaque type for function prototypes.

The following two require "*roff.h*" but no other mandoc headers. Afterwards, any other mandoc headers can be included as needed.

"*mdoc.h*"
Requires <*sys/types.h*> for `size_t`.

Provides `enum mdocargt`, `enum mdoc_auth`, `enum mdoc_disp`, `enum mdoc_font`, `enum mdoc_list`, `struct mdoc_argv`, `struct mdoc_arg`, `struct mdoc_an`, `struct mdoc_bd`, `struct mdoc_bf`, `struct mdoc_bl`, `struct mdoc_rs`, `union mdoc_data`, and the functions **mdoc**_*() described in *mandoc*(3).

Uses the types `struct roff_node` from "*roff.h*" and `struct roff_man` from "*roff_int.h*" as opaque types for function prototypes.

When this header is included, the same file should not include internals of different parsers.

"*man.h*"
Provides the functions **man**_*() described in *mandoc*(3).

Uses the type `struct roff_man` from "*roff.h*" as an opaque type for function prototypes.

When this header is included, the same file should not include internals of different parsers.

Parser internals

Most of the following headers require inclusion of a parser interface header before they can be included. All parser interface headers should precede all parser internal headers. When any parser internal headers are included, the same file should not include any formatter headers.

"*libmandoc.h*"

Requires <*sys/types.h*> for `size_t` and "*mandoc.h*" for `enum mandocerr`.

Provides `struct buf`, utility functions needed by multiple parsers, and the top-level functions to call the parsers.

Uses the opaque type `struct roff` from *roff.c* for function prototypes. Uses the type `struct roff_man` from "*roff.h*" as an opaque type for function prototypes.

"*roff_int.h*"

Parser internals shared by multiple parsers. Can be used in all parsers, but not in main programs or formatters.

Requires "*roff.h*" for `enum roff_type` and `enum roff_tok`.

Provides `enum roff_next`, `struct roff_man`, functions named **roff_**∗() to handle roff nodes, **roffhash_alloc**(), **roffhash_find**(), **roffhash_free**(), and **roff_validate**(), and the two special functions **man_breakscope**() and **mdoc_argv_free**() because the latter two are needed by *roff.c*.

Uses the types `struct ohash` from "*mandoc_ohash.h*", `struct roff_node` and `struct roff_meta` from "*roff.h*", `struct roff` from *roff.c*, and `struct mdoc_arg` from "*mdoc.h*" as opaque types for function prototypes.

"*libmdoc.h*"

Requires "*roff.h*" for `enum roff_tok` and `enum roff_sec`.

Provides `enum margserr`, `enum mdelim`, `struct mdoc_macro`, and many functions internal to the *mdoc*(7) parser.

Uses the types `struct roff_node` from "*roff.h*", `struct roff_man` from "*roff_int.h*", and `struct mdoc_arg` from "*mdoc.h*" as opaque types for function prototypes.

When this header is included, the same file should not include interfaces of different parsers.

"*libman.h*"

Requires "*roff.h*" for `enum roff_tok`.

Provides `struct man_macro` and some functions internal to the *man*(7) parser.

Uses the types `struct roff_node` from "*roff.h*" and `struct roff_man` from "*roff_int.h*" as opaque types for function prototypes.

When this header is included, the same file should not include interfaces of different parsers.

"*eqn_parse.h*"

External interface of the *eqn*(7) parser, for use in the *roff*(7) and *eqn*(7) parsers only.

Requires <*sys/types.h*> for `size_t`.

Provides `struct eqn_node` and the functions **eqn_alloc**(), **eqn_box_new**(), **eqn_box_free**(), **eqn_free**(), **eqn_parse**(), **eqn_read**(), and **eqn_reset**().

Uses the type `struct eqn_box` from "*mandoc.h*" as an opaque type for function prototypes. Uses the types `struct roff_node` from "*roff.h*" and `struct eqn_def` from *eqn.c* as opaque struct members.

When this header is included, the same file should not include internals of different parsers.

"*tbl_parse.h*"

External interface of the *tbl*(7) parser, for use in the *roff*(7) and *tbl*(7) parsers only.

Provides the functions documented in *tbl*(3).

Uses the types `struct tbl_span` from "*tbl.h*" and `struct tbl_node` from "*tbl_int.h*" as opaque types for function prototypes.

When this header is included, the same file should not include internals of different parsers.

"*tbl_int.h*"

Internal interfaces of the *tbl*(7) parser, for use inside the *tbl*(7) parser only.

Requires "*tbl.h*" for `struct tbl_opts`.

Provides `enum tbl_part`, `struct tbl_node`, and the functions **tbl_option**(), **tbl_layout**(), **tbl_data**(), **tbl_cdata**(), and **tbl_reset**().

When this header is included, the same file should not include interfaces of different parsers.

Formatter interface

These headers should be included after any parser interface headers. No parser internal headers should be included by the same file.

"*out.h*" Requires <*sys/types.h*> for `size_t`.

Provides `enum roffscale`, `struct roffcol`, `struct roffsu`, `struct rofftbl`, **a2roffsu**(), and **tblcalc**().

Uses `struct tbl_span` from "*mandoc.h*" as an opaque type for function prototypes.

When this header is included, the same file should not include "*mansearch.h*".

"*term.h*"

Requires <*sys/types.h*> for `size_t` and "*out.h*" for `struct roffsu` and `struct rofftbl`.

Provides `enum termenc`, `enum termfont`, `enum termtype`, `struct termp_tbl`, `struct termp`, **roff_term_pre**(), and many terminal formatting functions.

Uses the opaque type `struct termp_ps` from *term_ps.c*. Uses `struct tbl_span` and `struct eqn_box` from "*mandoc.h*" and `struct roff_meta` and `struct roff_node` from "*roff.h*" as opaque types for function prototypes.

When this header is included, the same file should not include "*html.h*" or "*mansearch.h*".

"*tag_term.h*"

Requires <*sys/types.h*> for `size_t` and <*stdio.h*> for `FILE`.

Provides an interface to generate *ctags*(1) files for the **:t** functionality mentioned in *man*(1).

Uses the type `struct roff_node` from "*roff.h*" as an opaque type for function prototypes.

When this header is included, the same file should not include "*html.h*" or "*mansearch.h*".

"*html.h*"

Requires <*sys/types.h*> for `size_t`, "*mandoc.h*" for `enum mandoc_esc`, "*roff.h*" for `enum roff_tok`, and "*out.h*" for `struct roffsu` and `struct rofftbl`.

Provides `enum htmltag`, `enum htmlattr`, `enum htmlfont`, `struct tag`, `struct tagq`, `struct htmlpair`, `struct html`, **roff_html_pre**(), and many HTML formatting functions.

Uses `struct tbl_span` and `struct eqn_box` from "*mandoc.h*" and `struct roff_node` from "*roff.h*" as opaque types for function prototypes.

When this header is included, the same file should not include "*term.h*", "*tab_term.h*", or "*mansearch.h*".

"*main.h*"

Provides the top level steering functions for all formatters.

Uses the type `struct roff_meta` from "*roff.h*" as an opaque type for function prototypes.

"*manconf.h*"

Requires <*sys/types.h*> for `size_t`.

Provides *struct manconf*, *struct manpaths*, *struct manoutput*, and the functions **manconf_parse**(), **manconf_output**(), **manconf_free**(), and **manpath_base**().

"*mansearch.h*"

Requires <*sys/types.h*> for *size_t* and <*stdint.h*> for *uint64_t*.

Provides *enum argmode*, *struct manpage*, *struct mansearch*, and the functions **mansearch**() and **mansearch_free**().

Uses *struct manpaths* from "*manconf.h*" as an opaque type for function prototypes.

When this header is included, the same file should not include "*out.h*", "*term.h*", "*tab_term.h*", or "*html.h*".

NAME

mandoc_html — internals of the mandoc HTML formatter

SYNOPSIS

```
#include <sys/types.h>
#include "mandoc.h"
#include "roff.h"
#include "out.h"
#include "html.h"

void
print_gen_decls(struct html *h);

void
print_gen_comment(struct html *h, struct roff_node *n);

void
print_gen_head(struct html *h);

struct tag *
print_otag(struct html *h,enum htmltag tag,const char *fmt, ...);

void
print_tagq(struct html *h,const struct tag *until);

void
print_stagq(struct html *h,const struct tag *suntil);

void
html_close_paragraph(struct html *h);

enum roff_tok
html_fillmode(struct html *h,enum roff_tok tok);

int
html_setfont(struct html *h,enum mandoc_esc font);

void
print_text(struct html *h,const char *word);

void
print_tagged_text(struct html *h,const char *word,struct roff_node *n);

char *
html_make_id(const struct roff_node *n,int unique);

struct tag *
print_otag_id(struct html *h,enum htmltag tag,const char *cattr,
    struct roff_node *n);

void
print_endline(struct html *h);
```

DESCRIPTION

The mandoc HTML formatter is not a formal library. However, as it is compiled into more than one program, in particular *mandoc*(1) and *man.cgi*(8), and because it may be security-critical in some contexts, some documentation is useful to help to use it correctly and to prevent XSS vulnerabilities.

The formatter produces HTML output on the standard output. Since proper escaping is usually required and best taken care of at one central place, the language-specific formatters (**_html.c*, see “FILES”) are not supposed to print directly to `stdout` using functions like *printf*(3), *putc*(3), *puts*(3), or *write*(2). Instead, they are expected to use the output functions declared in *html.h* and implemented as part of the main HTML formatting engine in *html.c*.

Data structures

These structures are declared in *html.h*.

`struct html`
: Internal state of the HTML formatter.

`struct tag`
: One entry for the LIFO stack of HTML elements. Members include `enum htmltag tag` and `struct tag *next`.

Private interface functions

The function **print_gen_decls**() prints the opening ⟨!**DOCTYPE**⟩ declaration.

The function **print_gen_comment**() prints the leading comments, usually containing a Copyright notice and license, as an HTML comment. It is intended to be called right after opening the ⟨**HTML**⟩ element. Pass the first `ROFFT_COMMENT` node in *n*.

The function **print_gen_head**() prints the opening ⟨**META**⟩ and ⟨**LINK**⟩ elements for the document ⟨**HEAD**⟩, using the *style* member of *h* unless that is `NULL`. It uses **print_otag**() which takes care of properly encoding attributes, which is relevant for the *style* link in particular.

The function **print_otag**() prints the start tag of an HTML element with the name *tag*, optionally including the attributes specified by *fmt*. If *fmt* is the empty string, no attributes are written. Each letter of *fmt* specifies one attribute to write. Most attributes require one *char* * argument which becomes the value of the attribute. The arguments have to be given in the same order as the attribute letters. If an argument is `NULL`, the respective attribute is not written.

`c`
: Print a `class` attribute.

`h`
: Print a `href` attribute. This attribute letter can optionally be followed by a modifier letter. If followed by `R`, it formats the link as a local one by prefixing a '#' character. If followed by `I`, it interpretes the argument as a header file name and generates a link using the *mandoc*(1) `-O includes` option. If followed by `M`, it takes two arguments instead of one, a manual page name and section, and formats them as a link to a manual page using the *mandoc*(1) `-O man` option.

`i`
: Print an `id` attribute.

`?`
: Print an arbitrary attribute. This format letter requires two *char* * arguments, the attribute name and the value. The name must not be `NULL`.

`s`
: Print a `style` attribute. If present, it must be the last format letter. It requires two *char* * arguments. The first is the name of the style property, the second its value. The name must not be `NULL`. The `s` *fmt* letter can be repeated, each repetition requiring an additional pair of *char* * arguments.

print_otag() uses the private function **print_encode**() to take care of HTML encoding. If required by the element type, it remembers in *h* that the element is open. The function **print_tagq**() is used to close out all open elements up to and including *until*; **print_stagq**() is a variant to close out all open elements up to but excluding *suntil*. The function **html_close_paragraph**() closes all open elements that establish phrasing context, thus returning to the innermost flow context.

The function **html_fillmode**() switches to fill mode if *want* is `ROFF_fi` or to no-fill mode if *want* is `ROFF_nf`. Switching from fill mode to no-fill mode closes the current paragraph and opens a ⟨**PRE**⟩ element. Switching in the opposite direction closes the ⟨**PRE**⟩ element, but does not open a new paragraph. If *want* matches the mode that is already active, no elements are closed nor opened. If *want* is `TOKEN_NONE`, the mode remains as it is.

The function **html_setfont**() selects the *font*, which can be ESCAPE_FONTROMAN, ESCAPE_FONTBOLD, ESCAPE_FONTITALIC, ESCAPE_FONTBI, or ESCAPE_FONTCW, for future text output and internally remembers the font that was active before the change. If the *font* argument is ESCAPE_FONTPREV, the current and the previous font are exchanged. This function only changes the internal state of the *h* object; no HTML elements are written yet. Subsequent text output will write font elements when needed.

The function **print_text**() prints HTML element content. It uses the private function **print_encode**() to take care of HTML encoding. If the document has requested a non-standard font, for example using a *roff*(7) **\f** font escape sequence, **print_text**() wraps *word* in an HTML font selection element using the **print_otag**() and **print_tagq**() functions.

The function **print_tagged_text**() is a variant of **print_text**() that wraps *word* in an ⟨**A**⟩ element of class "permalink" if *n* is not NULL and yields a segment identifier when passed to **html_make_id**().

The function **html_make_id**() allocates a string to be used for the id attribute of an HTML element and/or as a segment identifier for a URI in an ⟨**A**⟩ element. If *n* contains a *tag* attribute, it is used; otherwise, child nodes are used. If *n* is an **Sh**, **Ss**, **Sx**, **SH**, or **SS** node, the resulting string is the concatenation of the child strings; for other node types, only the first child is used. Bytes not permitted in URI-fragment strings are replaced by underscores. If any of the children to be used is not a text node, no string is generated and NULL is returned instead. If the *unique* argument is non-zero, deduplication is performed by appending an underscore and a decimal integer, if necessary. If the *unique* argument is 1, this is assumed to be the first call for this tag at this location, typically for use by NODE_ID, so the integer is incremented before use. If the *unique* argument is 2, this is ssumed to be the second call for this tag at this location, typically for use by NODE_HREF, so the existing integer, if any, is used without incrementing it.

The function **print_otag_id**() opens a *tag* element of class *cattr* for the node *n*. If the flag NODE_ID is set in *n*, it attempts to generate an id attribute with **html_make_id**(). If the flag NODE_HREF is set in *n*, an ⟨**A**⟩ element of class "permalink" is added: outside if *n* generates an element that can only occur in phrasing context, or inside otherwise. This function is a wrapper around **html_make_id**() and **print_otag**(), automatically chosing the *unique* argument appropriately and setting the *fmt* arguments to "chR" and "ci", respectively.

The function **print_endline**() makes sure subsequent output starts on a new HTML output line. If nothing was printed on the current output line yet, it has no effect. Otherwise, it appends any buffered text to the current output line, ends the line, and updates the internal state of the *h* object.

The functions **print_eqn**(), **print_tbl**(), and **print_tblclose**() are not yet documented.

RETURN VALUES

The functions **print_otag**() and **print_otag_id**() return a pointer to a new element on the stack of HTML elements. When **print_otag_id**() opens two elements, a pointer to the outer one is returned. The memory pointed to is owned by the library and is automatically *free*(3)d when **print_tagq**() is called on it or when **print_stagq**() is called on a parent element.

The function **html_fillmode**() returns ROFF_fi if fill mode was active before the call or ROFF_nf otherwise.

The function **html_make_id**() returns a newly allocated string or NULL if *n* lacks text data to create the attribute from. The caller is responsible for *free*(3)ing the returned string after using it.

In case of *malloc*(3) failure, these functions do not return but call *err*(3).

FILES

main.h	declarations of public functions for use by the main program, not yet documented
html.h	declarations of data types and private functions for use by language-specific HTML formatters
html.c	main HTML formatting engine and utility functions

mdoc_html.c	*mdoc*(7) HTML formatter
man_html.c	*man*(7) HTML formatter
tbl_html.c	*tbl*(7) HTML formatter
eqn_html.c	*eqn*(7) HTML formatter
roff_html.c	*roff*(7) HTML formatter, handling requests like **br**, **ce**, **fi**, **ft**, **nf**, **rj**, and **sp**.
out.h	declarations of data types and private functions for shared use by all mandoc formatters, not yet documented
out.c	private functions for shared use by all mandoc formatters
mandoc_aux.h	declarations of common mandoc utility functions, see *mandoc*(3)
mandoc_aux.c	implementation of common mandoc utility functions

SEE ALSO

mandoc(1), *mandoc*(3), *man.cgi*(8)

AUTHORS

The mandoc HTML formatter was written by Kristaps Dzonsons <kristaps@bsd.lv>. It is maintained by Ingo Schwarze <schwarze@openbsd.org>, who also wrote this manual.

NAME

mandoc_malloc, mandoc_realloc, mandoc_reallocarray, mandoc_calloc, mandoc_recallocarray, mandoc_strdup, mandoc_strndup, mandoc_asprintf — memory allocation function wrappers used in the mandoc library

SYNOPSIS

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

void *
mandoc_malloc(size_t size);

void *
mandoc_realloc(void *ptr, size_t size);

void *
mandoc_reallocarray(void *ptr, size_t nmemb, size_t size);

void *
mandoc_calloc(size_t nmemb, size_t size);

void *
mandoc_recallocarray(void *ptr, size_t oldnmemb, size_t nmemb, size_t size);

char *
mandoc_strdup(const char *s);

char *
mandoc_strndup(const char *s, size_t maxlen);

int
mandoc_asprintf(char **ret, const char *format, ...);
```

DESCRIPTION

These functions call the libc functions of the same names, passing through their return values when successful. In case of failure, they do not return, but instead call *err*(3). They can be used both internally by any code in the mandoc libraries and externally by programs using that library, for example *mandoc*(1), *man*(1), *apropos*(1), *makewhatis*(8), and *man.cgi*(8).

The function **mandoc_malloc**() allocates one new object, leaving the memory uninitialized. The functions **mandoc_realloc**(), **mandoc_reallocarray**(), and **mandoc_recallocarray**() change the size of an existing object or array, possibly moving it. When shrinking the size, existing data is truncated; when growing, only **mandoc_recallocarray**() initializes the new elements to zero. The function **mandoc_calloc**() allocates a new array, initializing it to zero.

The argument *size* is the size of each object. The argument *nmemb* is the new number of objects in the array. The argument *oldnmemb* is the number of objects in the array before the call. The argument *ptr* is a pointer to the existing object or array to be resized; if it is NULL, a new object or array is allocated.

The functions **mandoc_strdup**() and **mandoc_strndup**() copy a string into newly allocated memory. For **mandoc_strdup**(), the string pointed to by *s* needs to be NUL-terminated. For **mandoc_strndup**(), at most *maxlen* bytes are copied. The function **mandoc_asprintf**() writes output formatted according to *format* into newly allocated memory and returns a pointer to the result in *ret*. For all three string functions, the result is always NUL-terminated.

When the objects and strings are no longer needed, the pointers returned by these functions can be passed to *free*(3).

RETURN VALUES

The function **mandoc_asprintf**() always returns the number of characters written, excluding the final NUL byte. It never returns -1.

The other functions always return a valid pointer; they never return `NULL`.

FILES

These functions are implemented in *mandoc_aux.c*.

SEE ALSO

asprintf(3), *err*(3), *malloc*(3), *strdup*(3)

STANDARDS

The functions **malloc**(), **realloc**(), and **calloc**() are required by ANSI X3.159-1989 (“ANSI C89”). The functions **strdup**() and **strndup**() are required by IEEE Std 1003.1-2008 (“POSIX.1”). The function **asprintf**() is a widespread extension that first appeared in the GNU C library.

The function **reallocarray**() is an extension that first appeared in OpenBSD 5.6, and **recallocarray**() in OpenBSD 6.1. If these two are not provided by the operating system, the mandoc build system uses bundled portable implementations.

HISTORY

The functions **mandoc_malloc**(), **mandoc_realloc**(), **mandoc_calloc**(), and **mandoc_strdup**() have been available since mandoc 1.9.12, **mandoc_strndup**() since 1.11.5, **mandoc_asprintf**() since 1.12.4, **mandoc_reallocarray**() since 1.13.0, and **mandoc_recallocarray**() since 1.14.2.

AUTHORS

Kristaps Dzonsons <kristaps@bsd.lv>
Ingo Schwarze <schwarze@openbsd.org>

NAME

mansearch — search manual page databases

SYNOPSIS

```
#include <stdint.h>
#include <manconf.h>
#include <mansearch.h>

int
mansearch(const struct mansearch *search, const struct manpaths *paths,
    int argc, char *argv[], struct manpage **res, size_t *sz);
```

DESCRIPTION

The **mansearch**() function returns information about manuals matching a search query from a *mandoc.db*(5) database.

The query arguments are as follows:

*const struct mansearch *search*
: Search options, defined in <*mansearch.h*>.

*const struct manpaths *paths*
: Directories to be searched, defined in <*manconf.h*>.

int argc, *char *argv[]*
: Search criteria, usually taken from the command line.

The output arguments are as follows:

*struct manpage **res*
: Returns a pointer to an array of result structures defined in <*mansearch.h*>. The user is expected to call *free*(3) on the *file*, *names*, and *output* fields of all structures, as well as the *res* array itself.

*size_t *sz*
: Returns the number of result structures contained in *res*.

IMPLEMENTATION NOTES

For each manual page tree, the search is done in two steps. In the first step, a list of pages matching the search criteria is built. In the second step, the requested information about these pages is retrieved from the database and assembled into the *res* array.

All function mentioned here are defined in the file *mansearch.c*.

Finding matches

Command line parsing is done by the function **exprcomp**() building a singly linked list of *expr* structures, using the helper functions **expr_and**() and **exprterm**().

Assembling the results

The names, sections, and architectures of the manuals found are assembled into the *names* field of the result structure by the function **buildnames**().

FILES

mandoc.db The manual page database.

SEE ALSO

apropos(1), *mandoc.db*(5), *makewhatis*(8)

HISTORY

The **mansearch**() subsystem first appeared in OpenBSD 5.6.

AUTHORS

A module to search manual page databases was first written by Kristaps Dzonsons <kristaps@bsd.lv> in 2011, at first using the Berkeley DB; he rewrote it for SQLite3 in 2012, and Ingo Schwarze <schwarze@openbsd.org> removed the dependency on SQLite3 in 2016.

NAME

mchars_alloc, mchars_free, mchars_num2char, mchars_num2uc, mchars_spec2cp, mchars_spec2str, mchars_uc2str — character table for mandoc

SYNOPSIS

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

void
mchars_alloc(void);

void
mchars_free(void);

char
mchars_num2char(const char *decimal, size_t sz);

int
mchars_num2uc(const char *hexadecimal, size_t sz);

int
mchars_spec2cp(const char *name, size_t sz);

const char *
mchars_spec2str(const char *name, size_t sz, size_t *rsz);

const char *
mchars_uc2str(int codepoint);
```

DESCRIPTION

These functions translate Unicode character numbers and *roff*(7) character names into glyphs. See *mandoc_char*(7) for a list of *roff*(7) special characters. These functions are intended for external use by programs formatting *mdoc*(7) and *man*(7) pages for output, for example the *mandoc*(1) output formatter modules and *makewhatis*(8). The *decimal*, *hexadecimal*, *name*, and *size* input arguments are usually obtained from the *mandoc_escape*(3) parser function.

The function **mchars_num2char**() converts a *decimal* string representation of a character number consisting of *sz* digits into a printable ASCII character. If the input string is non-numeric or does not represent a printable ASCII character, the NUL character ('\0') is returned. For example, the *mandoc*(1) `-Tascii`, `-Tutf8`, and `-Thtml` output modules use this function to render *roff*(7) **\N** escape sequences.

The function **mchars_num2uc**() converts a *hexadecimal* string representation of a Unicode codepoint consisting of *sz* digits into an integer representation. If the input string is non-numeric or represents an ASCII character, the NUL character ('\0') is returned. For example, the *mandoc*(1) `-Tutf8` and `-Thtml` output modules use this function to render *roff*(7) **\[u***XXXX***]** and **\C'u***XXXX***'** escape sequences.

The function **mchars_alloc**() initializes a static *struct ohash* object for subsequent use by the following two lookup functions. When no longer needed, this object can be destroyed with **mchars_free**().

The function **mchars_spec2cp**() looks up a *roff*(7) special character *name* consisting of *sz* characters and returns the corresponding Unicode codepoint. If the *name* is not recognized, −1 is returned. For example, the *mandoc*(1) `-Tutf8` and `-Thtml` output modules use this function to render *roff*(7) **\[***name***]** and **\C'***name***'** escape sequences.

The function **mchars_spec2str**() looks up a *roff*(7) special character *name* consisting of *sz* characters and returns an ASCII string representation. The length of the representation is returned in *rsz*. In many cases, the meaning of such ASCII representations is not quite obvious, so using *roff*(7) special characters in documents intended for ASCII rendering is usually a bad idea. If the *name* is not recognized, `NULL` is returned. For example, *makewhatis*(8) and the *mandoc*(1) `-Tascii` output module use this function to render *roff*(7) **\[***name***]** and **\C'***name***'** escape sequences.

The function **mchars_uc2str**() performs a reverse lookup of the Unicode *codepoint* and returns an ASCII string representation, or the string "<?>" if none is available.

FILES

These funtions are implemented in the file *chars.c*.

SEE ALSO

mandoc(1), *mandoc_escape*(3), *ohash_init*(3), *mandoc_char*(7), *roff*(7)

HISTORY

These functions and their predecessors have been available since the following mandoc versions:

function	since	**predecessor**	since
mchars_alloc()	1.11.3	**ascii2htab**()	1.5.3
mchars_free()	1.11.2	**asciifree**()	1.6.0
mchars_num2char()	1.11.2	**chars_num2char**()	1.10.10
mchars_num2uc()	1.11.3	—	—
mchars_spec2cp()	1.11.2	**chars_spec2cp**()	1.10.5
mchars_spec2str()	1.11.2	**a2ascii**()	1.5.3
mchars_uc2str()	1.13.2	—	—

AUTHORS

Kristaps Dzonsons <kristaps@bsd.lv>
Ingo Schwarze <schwarze@openbsd.org>

NAME

tbl_alloc, tbl_read, tbl_restart, tbl_span, tbl_end, tbl_free — roff table parser library for mandoc

SYNOPSIS

```
#include <sys/types.h>
#include <tbl.h>
#include <tbl_parse.h>

struct tbl_node *
tbl_alloc(int pos, int line);

void
tbl_read(struct tbl_node *tbl, int ln, const char *p, int offs);

void
tbl_restart(int line, int pos, struct tbl_node *tbl);

const struct tbl_span *
tbl_span(struct tbl_node *tbl);

void
tbl_end(struct tbl_node **tblp);

void
tbl_free(struct tbl_node *tbl);
```

DESCRIPTION

This library is tightly integrated into the *mandoc*(1) utility and not designed for stand-alone use. The present manual is intended as a reference for developers working on *mandoc*(1).

Data structures

Unless otherwise noted, all of the following data structures are declared in <*tbl.h*> and are deleted in **tbl_free**().

struct tbl_node

This structure describes a complete table. It is declared in <*tbl_int.h*>, created in **tbl_alloc**(), and stored in the members *first_tbl*, *last_tbl*, and *tbl* of *struct roff* [*roff.c*].

The *first_span*, *current_span*, *last_span*, and *next* members may be NULL. The *first_row* and *last_row* members may be NULL, but if there is a span, the function **tbl_layout**() guarantees that these pointers are not NULL.

struct tbl_opts

This structure describes the options of one table. It is used as a substructure of *struct tbl_node* and thus created and deleted together with it. It is filled in **tbl_options**().

struct tbl_row

This structure describes one layout line in a table by maintaining a list of all the cells in that line. It is allocated and filled in **row**() [*tbl_layout.c*] and referenced from the *layout* member of *struct tbl_node*.

The *next* member may be NULL. The function **tbl_layout**() guarantees that the *first* and *last* members are not NULL.

struct tbl_cell

This structure describes one layout cell in a table, in particular its alignment, membership in spans, and usage for lines. It is allocated and filled in **cell_alloc**() [*tbl_layout.c*] and referenced from the *first* and *last* members of *struct tbl_row*.

The *next* member may be NULL.

struct tbl_span

This structure describes one data line in a table by maintaining a list of all data cells in that line or by specifying that it is a horizontal line. It is allocated and filled in **newspan**() [*tbl_data.c*] which is

called from **tbl_data**() and referenced from the *first_span*, *current_span*, and *last_span* members of *struct tbl_node*, and from the *span* members of *struct man_node* and *struct mdoc_node* from <*man.h*> and <*mdoc.h*>.

The *first*, *last*, *prev*, and *next* members may be NULL. The function **newspan**() [*tbl_data.c*] guarantees that the *opts* and *layout* members are not NULL.

struct tbl_dat

This structure describes one data cell in a table by specifying whether it contains a line or data, whether it spans additional layout cells, and by storing the data. It is allocated and filled in **tbl_data**() and referenced from the *first* and *last* members of *struct tbl_span*.

The *string* and *next* members may be NULL. The function **getdata**() guarantees that the *layout* member is not NULL.

Interface functions

The following functions are implemented in *tbl.c*, and all callers are in *roff.c*.

tbl_alloc()

Allocates, initializes, and returns a new *struct tbl_node*. Called from **roff_TS**().

tbl_read()

Dispatches to **tbl_option**(), **tbl_layout**(), **tbl_cdata**(), and **tbl_data**(), see below. Called from **roff_parseln**().

tbl_restart()

Resets the *part* member of *struct tbl_node* to TBL_PART_LAYOUT. Called from **roff_T_**().

tbl_span()

On the first call, return the first *struct tbl_span*; for later calls, return the next one or NULL. Called from **roff_span**().

tbl_end()

Flags the last span as TBL_SPAN_LAST and clears the pointer passed as an argment. Called from **roff_TE**() and **roff_endparse**().

tbl_free()

Frees the specified *struct tbl_node* and all the tbl_row, tbl_cell, tbl_span, and tbl_dat structures referenced from it. Called from **roff_free**() and **roff_reset**().

Private functions

The following functions are declared in <*tbl_int.h*>.

int **tbl_options**(*struct tbl_node *tbl*, *int ln*, *const char *p*)

Parses the options line into *struct tbl_opts*. Implemented in *tbl_opts.c*, called from **tbl_read**().

int **tbl_layout**(*struct tbl_node *tbl*, *int ln*, *const char *p*)

Allocates and fills one *struct tbl_row* for each layout line and one *struct tbl_cell* for each layout cell. Implemented in *tbl_layout.c*, called from **tbl_read**().

int **tbl_data**(*struct tbl_node *tbl*, *int ln*, *const char *p*)

Allocates one *struct tbl_span* for each data line and calls **getdata**() for each data cell. Implemented in *tbl_data.c*, called from **tbl_read**().

int **tbl_cdata**(*struct tbl_node *tbl*, *int ln*, *const char *p*)

Continues parsing a data line: When finding 'T}', switches back to TBL_PART_DATA mode and calls **getdata**() if there are more data cells on the line. Otherwise, appends the data to the current data cell. Implemented in *tbl_data.c*, called from **tbl_read**().

int **getdata**(*struct tbl_node* ∗*tbl*, *struct tbl_span* ∗*dp*, *int ln*, *const char* ∗*p*, *int* ∗*pos*)
Parses one data cell into one *struct tbl_dat*. Implemented in *tbl_data.c*, called from **tbl_data**() and **tbl_cdata**().

SEE ALSO

mandoc(1), *mandoc*(3), *tbl*(7)

AUTHORS

The **tbl** library was written by Kristaps Dzonsons <kristaps@bsd.lv> with contributions from Ingo Schwarze <schwarze@openbsd.org>.

NAME

man.conf — configuration file for man

DESCRIPTION

This is the configuration file for the *man*(1), *apropos*(1), and *makewhatis*(8) utilities. Its presence, and all directives, are optional.

This file is an ASCII text file. Leading whitespace on lines, lines starting with '#', and blank lines are ignored. Words are separated by whitespace. The first word on each line is the name of a configuration directive.

The following directives are supported:

manpath *path*

Override the default search *path* for *man*(1), *apropos*(1), and *makewhatis*(8). It can be used multiple times to specify multiple paths, with the order determining the manual page search order.

Each path is a tree containing subdirectories whose names consist of the strings 'man' and/or 'cat' followed by the names of sections, usually single digits. The former are supposed to contain unformatted manual pages in *mdoc*(7) and/or *man*(7) format; file names should end with the name of the section preceded by a dot. The latter should contain preformatted manual pages; file names should end with '.0'.

Creating a *mandoc.db*(5) database with *makewhatis*(8) in each directory configured with **manpath** is recommended and necessary for *apropos*(1) to work, and also for *man*(1) on operating systems like OpenBSD that install each manual page with only one file name in the file system, even if it documents multiple utilities or functions.

output *option* [*value*]

Configure the default value of an output option. These directives are overridden by the -O command line options of the same names. For details, see the *mandoc*(1) manual.

option	*value*	used by -T	purpose
fragment	none	html	print only body
includes	string	html	path to header files
indent	integer	ascii, utf8	left margin
man	string	html	path for Xr links
paper	string	ps, pdf	paper size
style	string	html	CSS file
toc	none	html	print table of contents
width	integer	ascii, utf8	right margin

FILES

/etc/man.conf

EXAMPLES

The following configuration file reproduces the defaults: installing it is equivalent to not having a **man.conf** file at all.

```
manpath /usr/share/man
manpath /usr/X11R6/man
manpath /usr/local/man
```

SEE ALSO

apropos(1), *man*(1), *makewhatis*(8)

HISTORY

A relatively complicated **man.conf** file format first appeared in 4.3BSD−Reno. For OpenBSD 5.8, it was redesigned from scratch, aiming for simplicity.

AUTHORS

Ingo Schwarze <schwarze@openbsd.org>

NAME

mandoc.db — manual page database

DESCRIPTION

The **mandoc.db** file format is used to store information about installed manual pages to facilitate semantic searching for manuals. Each manual page tree contains its own **mandoc.db** file; see "FILES" for examples.

Such database files are generated by *makewhatis*(8) and used by *man*(1), *apropos*(1) and *whatis*(1).

The file format uses three datatypes:

- 32-bit signed integer numbers in big endian (network) byte ordering
- NUL-terminated strings
- lists of NUL-terminated strings, terminated by a second NUL character

Numbers are aligned to four-byte boundaries; where they follow strings or lists of strings, padding with additional NUL characters occurs. Some, but not all, numbers point to positions in the file. These pointers are measured in bytes, and the first byte of the file is considered to be byte 0.

Each file consists of:

- One magic number, 0x3a7d0cdb.
- One version number, currently 1.
- One pointer to the macros table.
- One pointer to the final magic number.
- The pages table (variable length).
- The macros table (variable length).
- The magic number once again, 0x3a7d0cdb.

The pages table contains one entry for each physical manual page file, no matter how many hard and soft links it may have in the file system. The pages table consists of:

- The number of pages in the database.
- For each page:
  - One pointer to the list of names.
  - One pointer to the list of sections.
  - One pointer to the list of architectures or 0 if the page is machine-independent.
  - One pointer to the one-line description string.
  - One pointer to the list of filenames.
- For each page, the list of names. Each name is preceded by a single byte indicating the sources of the name. The meaning of the bits is:
  - 0x10: The name appears in a filename.
  - 0x08: The name appears in a header line, i.e. in a .Dt or .TH macro.
  - 0x04: The name is the first one in the title line, i.e. it appears in the first .Nm macro in the NAME section.
  - 0x02: The name appears in any .Nm macro in the NAME section.
  - 0x01: The name appears in an .Nm block in the SYNOPSIS section.
- For each page, the list of sections. Each section is given as a string, not as a number.
- For each architecture-dependent page, the list of architectures.
- For each page, the one-line description string taken from the .Nd macro.
- For each page, the list of filenames relative to the root of the respective manpath. This list includes hard links, soft links, and links simulated with .so *roff*(7) requests. The first filename is preceded by a single byte having the following significance:
  - `FORM_SRC` = 0x01: The file format is *mdoc*(7) or *man*(7).
  - `FORM_CAT` = 0x02: The manual page is preformatted.
- Zero to three NUL bytes for padding.

The macros table consists of:

- The number of different macro keys, currently 36. The ordering of macros is defined in <*mansearch.h*> and the significance of the macro keys is documented in *apropos*(1).
- For each macro key, one pointer to the respective macro table.
- For each macro key, the macro table (variable length).

Each macro table consists of:

- The number of entries in the table.
- For each entry:
  - One pointer to the value of the macro key. Each value is a string of text taken from some macro invocation.
  - One pointer to the list of pages.
- For each entry, the value of the macro key.
- Zero to three NUL bytes for padding.
- For each entry, one or more pointers to pages in the pages table, pointing to the pointer to the list of names, followed by the number 0.

FILES

/usr/share/man/mandoc.db	The manual page database for the base system.
/usr/X11R6/man/mandoc.db	The same for the *X*(7) Window System.
/usr/local/man/mandoc.db	The same for *packages*(7).

A program to dump **mandoc.db** files in a human-readable format suitable for *diff*(1) is provided in the directory */usr/src/regress/usr.bin/mandoc/db/dbm_dump/*.

SEE ALSO

apropos(1), *man*(1), *whatis*(1), *makewhatis*(8)

HISTORY

A manual page database */usr/lib/whatis* first appeared in 2BSD. The present format first appeared in OpenBSD 6.1.

AUTHORS

The original version of *makewhatis*(8) was written by Bill Joy in 1979. The present database format was designed by Ingo Schwarze <schwarze@openbsd.org> in 2016.

NAME

eqn — eqn language reference for mandoc

DESCRIPTION

The **eqn** language is an equation-formatting language. It is used within *mdoc*(7) and *man*(7) Unix manual pages. It describes the *structure* of an equation, not its mathematical meaning. This manual describes the **eqn** language accepted by the *mandoc*(1) utility, which corresponds to the Second Edition **eqn** specification (see "SEE ALSO" for references).

An equation starts with an input line containing exactly the characters '.EQ', may contain multiple input lines, and ends with an input line containing exactly the characters '.EN'. Equivalently, an equation can be given in the middle of a single text input line by surrounding it with the equation delimiters defined with the `delim` statement.

The equation grammar is as follows, where quoted strings are case-sensitive literals in the input:

```
eqn     : box | eqn box
box     : text
        | "{" eqn "}"
        | "define" text text
        | "ndefine" text text
        | "tdefine" text text
        | "gfont" text
        | "gsize" text
        | "set" text text
        | "undef" text
        | "sqrt" box
        | box pos box
        | box mark
        | "matrix" "{" [col "{" list "}"]* "}"
        | pile "{" list "}"
        | font box
        | "size" text box
        | "left" text eqn ["right" text]
col     : "lcol" | "rcol" | "ccol" | "col"
text    : [^space\"]+ | \".*\"
pile    : "lpile" | "cpile" | "rpile" | "pile"
pos     : "over" | "sup" | "sub" | "to" | "from"
mark    : "dot" | "dotdot" | "hat" | "tilde" | "vec"
        | "dyad" | "bar" | "under"
font    : "roman" | "italic" | "bold" | "fat"
list    : eqn
        | list "above" eqn
space   : [\^~ \t]
```

White-space consists of the space, tab, circumflex, and tilde characters. It is required to delimit tokens consisting of alphabetic characters and it is ignored at other places. Braces and quotes also delimit tokens. If within a quoted string, these space characters are retained. Quoted strings are also not scanned for keywords, glyph names, and expansion of definitions. To print a literal quote character, it can be prepended with a backslash or expressed with the \(dq escape sequence.

Subequations can be enclosed in braces to pass them as arguments to operation keywords, overriding standard operation precedence. Braces can be nested. To set a brace verbatim, it needs to be enclosed in quotes.

The following text terms are translated into a rendered glyph, if available: alpha, beta, chi, delta, epsilon, eta, gamma, iota, kappa, lambda, mu, nu, omega, omicron, phi, pi, psi, rho, sigma, tau, theta, upsilon, xi, zeta, DELTA, GAMMA, LAMBDA, OMEGA, PHI, PI, PSI, SIGMA, THETA, UPSILON, XI, inter (intersection), union (union), prod (product), int (integral), sum (summation), grad (gradient), del (vector differential), times

(multiply), cdot (center-dot), nothing (zero-width space), approx (approximately equals), prime (prime), half (one-half), partial (partial differential), inf (infinity), >> (much greater), << (much less), <− (left arrow), −> (right arrow), +− (plus-minus), != (not equal), == (equivalence), <= (less-than-equal), and >= (more-than-equal). The character escape sequences documented in *mandoc_char*(7) can be used, too.

The following control statements are available:

`define`

Replace all occurrences of a key with a value. Its syntax is as follows:

```
define key cvalc
```

The first character of the value string, *c*, is used as the delimiter for the value *val*. This allows for arbitrary enclosure of terms (not just quotes), such as

```
define foo 'bar baz'
define foo cbar bazc
```

It is an error to have an empty *key* or *val*. Note that a quoted *key* causes errors in some **eqn** implementations and should not be considered portable. It is not expanded for replacements. Definitions may refer to other definitions; these are evaluated recursively when text replacement occurs and not when the definition is created.

Definitions can create arbitrary strings, for example, the following is a legal construction.

```
define foo 'define'
foo bar 'baz'
```

Self-referencing definitions will raise an error. The `ndefine` statement is a synonym for `define`, while `tdefine` is discarded.

`delim` This statement takes a string argument consisting of two bytes, to be used as the opening and closing delimiters for equations in the middle of text input lines. Conventionally, the dollar sign is used for both delimiters, as follows:

```
.EQ
delim $$
.EN
An equation like $sin pi = 0$ can now be entered
in the middle of a text input line.
```

The special statement `delim off` temporarily disables previously declared delimiters and `delim on` reenables them.

`gfont` Set the default font of subsequent output. Its syntax is as follows:

```
gfont font
```

In mandoc, this value is discarded.

`gsize` Set the default size of subsequent output. Its syntax is as follows:

```
gsize [+|−]size
```

The *size* value should be an integer. If prepended by a sign, the font size is changed relative to the current size.

`set` Set an equation mode. In mandoc, both arguments are thrown away. Its syntax is as follows:

```
set key val
```

The *key* and *val* are not expanded for replacements. This statement is a GNU extension.

`undef` Unset a previously-defined key. Its syntax is as follows:

`define` *`key`*

Once invoked, the definition for *`key`* is discarded. The *`key`* is not expanded for replacements. This statement is a GNU extension.

Operation keywords have the following semantics:

`above` See `pile`.

`bar` Draw a line over the preceding box.

`bold` Set the following box using bold font.

`ccol` Like `cpile`, but for use in `matrix`.

`cpile` Like `pile`, but with slightly increased vertical spacing.

`dot` Set a single dot over the preceding box.

`dotdot`
Set two dots (dieresis) over the preceding box.

`dyad` Set a dyad symbol (left-right arrow) over the preceding box.

`fat` A synonym for `bold`.

`font` Set the second argument using the font specified by the first argument; currently not recognized by the *mandoc*(1) **eqn** parser.

`from` Set the following box below the preceding box, using a slightly smaller font. Used for sums, integrals, limits, and the like.

`hat` Set a hat (circumflex) over the preceding box.

`italic`
Set the following box using italic font.

`lcol` Like `lpile`, but for use in `matrix`.

`left` Set the first argument as a big left delimiter before the second argument. As an optional third argument, `right` can follow. In that case, the fourth argument is set as a big right delimiter after the second argument.

`lpile` Like `cpile`, but subequations are left-justified.

`matrix`
Followed by a list of columns enclosed in braces. All columns need to have the same number of subequations. The columns are set as a matrix. The difference compared to multiple subsequent `pile` operators is that in a `matrix`, corresponding subequations in all columns line up horizontally, while each `pile` does vertical spacing independently.

`over` Set a fraction. The preceding box is the numerator, the following box is the denominator.

`pile` Followed by a list of subequations enclosed in braces, the subequations being separated by `above` keywords. Sets the subequations one above the other, each of them centered. Typically used to represent vectors in coordinate representation.

`rcol` Like `rpile`, but for use in `matrix`.

`right` See `left`; `right` cannot be used without `left`. To set a big right delimiter without a big left delimiter, the following construction can be used:

`left "" `*`box`* `right` *`delimiter`*

`roman` Set the following box using the default font.

`rpile` Like `cpile`, but subequations are right-justified.

`size` Set the second argument with the font size specified by the first argument; currently ignored by *mandoc*(1). By prepending a plus or minus sign to the first argument, the font size can be selected relative to the current size.

`sqrt` Set the square root of the following box.

`sub` Set the following box as a subscript to the preceding box.

`sup` Set the following box as a superscript to the preceding box. As a special case, if a `sup` clause immediately follows a `sub` clause as in

mainbox `sub` *subbox* `sup` *supbox*

both are set with respect to the same *mainbox*, that is, *supbox* is set above *subbox*.

`tilde` Set a tilde over the preceding box.

`to` Set the following box above the preceding box, using a slightly smaller font. Used for sums and integrals and the like. As a special case, if a `to` clause immediately follows a `from` clause as in

mainbox `from` *frombox* `to` *tobox*

both are set below and above the same *mainbox*.

`under` Underline the preceding box.

`vec` Set a vector symbol (right arrow) over the preceding box.

The binary operations `from`, `to`, `sub`, and `sup` group to the right, that is,

mainbox `sup` *supbox* `sub` *subbox*

is the same as

mainbox `sup` {*supbox* `sub` *subbox*}

and different from

{*mainbox* `sup` *supbox*} `sub` *subbox*.

By contrast, `over` groups to the left.

In the following list, earlier operations bind more tightly than later operations:

1. `dyad`, `vec`, `under`, `bar`, `tilde`, `hat`, `dot`, `dotdot`
2. `fat`, `roman`, `italic`, `bold`, `size`
3. `sub`, `sup`
4. `sqrt`
5. `over`
6. `from`, `to`

COMPATIBILITY

This section documents the compatibility of mandoc **eqn** and the troff **eqn** implementation (including GNU troff).

- The text string ‘\"’ is interpreted as a literal quote in troff. In mandoc, this is interpreted as a comment.
- In troff, The circumflex and tilde white-space symbols map to fixed-width spaces. In mandoc, these characters are synonyms for the space character.
- The troff implementation of **eqn** allows for equation alignment with the `mark` and `lineup` tokens. mandoc discards these tokens. The `back` *n*, `fwd` *n*, `up` *n*, and `down` *n* commands are also ignored.

SEE ALSO

mandoc(1), *man*(7), *mandoc_char*(7), *mdoc*(7), *roff*(7)

Brian W. Kernighan and Lorinda L. Cherry, “System for Typesetting Mathematics”, *Communications of the ACM*, 18, pp. 151–157, March, 1975.

Brian W. Kernighan and Lorinda L. Cherry, *Typesetting Mathematics, User's Guide*, 1976.

Brian W. Kernighan and Lorinda L. Cherry, *Typesetting Mathematics, User's Guide (Second Edition)*, 1978.

HISTORY

The eqn utility, a preprocessor for troff, was originally written by Brian W. Kernighan and Lorinda L. Cherry in 1975. The GNU reimplementation of eqn, part of the GNU troff package, was released in 1989 by James Clark. The eqn component of *mandoc*(1) was added in 2011.

AUTHORS

This **eqn** reference was written by Kristaps Dzonsons <kristaps@bsd.lv>.

NAME

man — legacy formatting language for manual pages

DESCRIPTION

The **man** language was the standard formatting language for AT&T UNIX manual pages from 1979 to 1989. Do not use it to write new manual pages: it is a purely presentational language and lacks support for semantic markup. Use the *mdoc*(7) language, instead.

In a **man** document, lines beginning with the control character '.' are called "macro lines". The first word is the macro name. It usually consists of two capital letters. For a list of portable macros, see "MACRO OVERVIEW". The words following the macro name are arguments to the macro.

Lines not beginning with the control character are called "text lines". They provide free-form text to be printed; the formatting of the text depends on the respective processing context:

```
.SH Macro lines change control state.
Text lines are interpreted within the current state.
```

Many aspects of the basic syntax of the **man** language are based on the *roff*(7) language; see the *LANGUAGE SYNTAX* and *MACRO SYNTAX* sections in the *roff*(7) manual for details, in particular regarding comments, escape sequences, whitespace, and quoting.

Each **man** document starts with the **TH** macro specifying the document's name and section, followed by the "NAME" section formatted as follows:

```
.TH PROGNAME 1 1979-01-10
.SH NAME
\fBprogname\fR \(en one line about what it does
```

MACRO OVERVIEW

This overview is sorted such that macros of similar purpose are listed together. Deprecated and non-portable macros are not included in the overview, but can be found in the alphabetical reference below.

Page header and footer meta-data

TH set the title: *name section date* [*source* [*volume*]]
AT display AT&T UNIX version in the page footer (<= 1 argument)
UC display BSD version in the page footer (<= 1 argument)

Sections and paragraphs

SH section header (one line)
SS subsection header (one line)
PP start an undecorated paragraph (no arguments)
RS, **RE** reset the left margin: [*width*]
IP indented paragraph: [*head* [*width*]]
TP tagged paragraph: [*width*]
PD set vertical paragraph distance: [*height*]
in additional indent: [*width*]

Physical markup

B boldface font
I italic font
SB small boldface font
SM small roman font
BI alternate between boldface and italic fonts
BR alternate between boldface and roman fonts
IB alternate between italic and boldface fonts
IR alternate between italic and roman fonts
RB alternate between roman and boldface fonts

RI alternate between roman and italic fonts

MACRO REFERENCE

This section is a canonical reference to all macros, arranged alphabetically. For the scoping of individual macros, see "MACRO SYNTAX".

AT Sets the volume for the footer for compatibility with man pages from AT&T UNIX releases. The optional arguments specify which release it is from. This macro is an extension that first appeared in 4.3BSD.

B Text is rendered in bold face.

BI Text is rendered alternately in bold face and italic. Thus, '.BI this word and that' causes 'this' and 'and' to render in bold face, while 'word' and 'that' render in italics. Whitespace between arguments is omitted in output.

Example:

```
.BI bold italic bold italic
```

BR Text is rendered alternately in bold face and roman (the default font). Whitespace between arguments is omitted in output. See also **BI**.

DT Restore the default tabulator positions. They are at intervals of 0.5 inches. This has no effect unless the tabulator positions were changed with the *roff*(7) **ta** request.

EE This is a non-standard Version 9 AT&T UNIX extension later adopted by GNU. In *mandoc*(1), it does the same as the *roff*(7) **fi** request (switch to fill mode).

EX This is a non-standard Version 9 AT&T UNIX extension later adopted by GNU. In *mandoc*(1), it does the same as the *roff*(7) **nf** request (switch to no-fill mode).

HP Begin a paragraph whose initial output line is left-justified, but subsequent output lines are indented, with the following syntax:

.**HP** [*width*]

The *width* argument is a *roff*(7) scaling width. If specified, it's saved for later paragraph left margins; if unspecified, the saved or default width is used.

This macro is portable, but deprecated because it has no good representation in HTML output, usually ending up indistinguishable from **PP**.

I Text is rendered in italics.

IB Text is rendered alternately in italics and bold face. Whitespace between arguments is omitted in output. See also **BI**.

IP Begin an indented paragraph with the following syntax:

.**IP** [*head* [*width*]]

The *width* argument is a *roff*(7) scaling width defining the left margin. It's saved for later paragraph left-margins; if unspecified, the saved or default width is used.

The *head* argument is used as a leading term, flushed to the left margin. This is useful for bulleted paragraphs and so on.

IR Text is rendered alternately in italics and roman (the default font). Whitespace between arguments is omitted in output. See also **BI**.

LP A synonym for **PP**.

ME End a mailto block started with **MT**. This is a non-standard GNU extension.

MT Begin a mailto block. This is a non-standard GNU extension. It has the following syntax:

```
.MT address
link description to be shown
.ME
```

OP Optional command-line argument. This is a non-standard DWB extension. It has the following syntax:

```
.OP key [value]
```

The *key* is usually a command-line flag and *value* its argument.

P This synonym for **PP** is an AT&T System III UNIX extension later adopted by 4.3BSD.

PD Specify the vertical space to be inserted before each new paragraph.
The syntax is as follows:

```
.PD [height]
```

The *height* argument is a *roff*(7) scaling width. It defaults to `1v`. If the unit is omitted, `v` is assumed.

This macro affects the spacing before any subsequent instances of **HP**, **IP**, **LP**, **P**, **PP**, **SH**, **SS**, **SY**, and **TP**.

PP Begin an undecorated paragraph. The scope of a paragraph is closed by a subsequent paragraph, sub-section, section, or end of file. The saved paragraph left-margin width is reset to the default.

RB Text is rendered alternately in roman (the default font) and bold face. Whitespace between arguments is omitted in output. See also **BI**.

RE Explicitly close out the scope of a prior **RS**. The default left margin is restored to the state before that **RS** invocation.

The syntax is as follows:

```
.RE [level]
```

Without an argument, the most recent **RS** block is closed out. If *level* is 1, all open **RS** blocks are closed out. Otherwise, *level* − 1 nested **RS** blocks remain open.

RI Text is rendered alternately in roman (the default font) and italics. Whitespace between arguments is omitted in output. See also **BI**.

RS Temporarily reset the default left margin. This has the following syntax:

```
.RS [width]
```

The *width* argument is a *roff*(7) scaling width. If not specified, the saved or default width is used.

See also **RE**.

SB Text is rendered in small size (one point smaller than the default font) bold face. This macro is an extension that probably first appeared in SunOS 4.0 and was later adopted by GNU and by 4.4BSD.

SH Begin a section. The scope of a section is only closed by another section or the end of file. The paragraph left-margin width is reset to the default.

SM Text is rendered in small size (one point smaller than the default font).

SS Begin a sub-section. The scope of a sub-section is closed by a subsequent sub-section, section, or end of file. The paragraph left-margin width is reset to the default.

SY Begin a synopsis block with the following syntax:

```
.SY command
arguments
.YS
```

This is a non-standard GNU extension and very rarely used even in GNU manual pages. Formatting is similar to **IP**.

TH Set the name of the manual page for use in the page header and footer with the following syntax:

.**TH** *name section date* [*source* [*volume*]]

Conventionally, the document *name* is given in all caps. The *section* is usually a single digit, in a few cases followed by a letter. The recommended *date* format is **YYYY-MM-DD** as specified in the ISO-8601 standard; if the argument does not conform, it is printed verbatim. If the *date* is empty or not specified, the current date is used. The optional *source* string specifies the organisation providing the utility. When unspecified, *mandoc*(1) uses its `-Ios` argument. The *volume* string replaces the default volume title of the *section*.

Examples:

```
.TH CVS 5 1992-02-12 GNU
```

TP Begin a paragraph where the head, if exceeding the indentation width, is followed by a newline; if not, the body follows on the same line after advancing to the indentation width. Subsequent output lines are indented. The syntax is as follows:

.**TP** [*width*]
head \" one line
body

The *width* argument is a *roff*(7) scaling width. If specified, it's saved for later paragraph left-margins; if unspecified, the saved or default width is used.

TQ Like **TP**, except that no vertical spacing is inserted before the paragraph. This is a non-standard GNU extension and very rarely used even in GNU manual pages.

UC Sets the volume for the footer for compatibility with man pages from BSD releases. The optional first argument specifies which release it is from. This macro is an extension that first appeared in 3BSD.

UE End a uniform resource identifier block started with **UR**. This is a non-standard GNU extension.

UR Begin a uniform resource identifier block. This is a non-standard GNU extension. It has the following syntax:

.**UR** *uri*
link description to be shown
.**UE**

YS End a synopsis block started with **SY**. This is a non-standard GNU extension.

in Indent relative to the current indentation:

.**in** [*width*]

If *width* is signed, the new offset is relative. Otherwise, it is absolute. This value is reset upon the next paragraph, section, or sub-section.

MACRO SYNTAX

The **man** macros are classified by scope: line scope or block scope. Line macros are only scoped to the current line (and, in some situations, the subsequent line). Block macros are scoped to the current line and subsequent lines until closed by another block macro.

Line Macros

Line macros are generally scoped to the current line, with the body consisting of zero or more arguments. If a macro is scoped to the next line and the line arguments are empty, the next line, which must be text, is used instead. Thus:

```
.I
foo
```

is equivalent to '.I foo'. If next-line macros are invoked consecutively, only the last is used. If a next-line macro is followed by a non-next-line macro, an error is raised.

The syntax is as follows:

```
.YO [body...]
[body...]
```

Macro	*Arguments*	*Scope*	*Notes*
AT	<=1	current	
B	n	next-line	
BI	n	current	
BR	n	current	
DT	0	current	
EE	0	current	Version 9 AT&T UNIX
EX	0	current	Version 9 AT&T UNIX
I	n	next-line	
IB	n	current	
IR	n	current	
OP	>=1	current	DWB
PD	1	current	
RB	n	current	
RI	n	current	
SB	n	next-line	
SM	n	next-line	
TH	>1, <6	current	
UC	<=1	current	
in	1	current	*roff*(7)

Block Macros

Block macros comprise a head and body. As with in-line macros, the head is scoped to the current line and, in one circumstance, the next line (the next-line stipulations as in "Line Macros" apply here as well).

The syntax is as follows:

```
.YO [head...]
[head...]
[body...]
```

The closure of body scope may be to the section, where a macro is closed by **SH**; sub-section, closed by a section or **SS**; or paragraph, closed by a section, sub-section, **HP**, **IP**, **LP**, **P**, **PP**, **RE**, **SY**, or **TP**. No closure refers to an explicit block closing macro.

As a rule, block macros may not be nested; thus, calling a block macro while another block macro scope is open, and the open scope is not implicitly closed, is syntactically incorrect.

Macro	*Arguments*	*Head Scope*	*Body Scope*	*Notes*
HP	<2	current	paragraph	
IP	<3	current	paragraph	
LP	0	current	paragraph	
ME	0	none	none	GNU
MT	1	current	to ME	GNU
P	0	current	paragraph	
PP	0	current	paragraph	

RE	<=1	current	none	
RS	1	current	to RE	
SH	>0	next-line	section	
SS	>0	next-line	sub-section	
SY	1	current	to YS	GNU
TP	n	next-line	paragraph	
TQ	n	next-line	paragraph	GNU
UE	0	current	none	GNU
UR	1	current	part	GNU
YS	0	none	none	GNU

If a block macro is next-line scoped, it may only be followed by in-line macros for decorating text.

Font handling

In **man** documents, both "Physical markup" macros and *roff*(7) '`\f`' font escape sequences can be used to choose fonts. In text lines, the effect of manual font selection by escape sequences only lasts until the next macro invocation; in macro lines, it only lasts until the end of the macro scope. Note that macros like **BR** open and close a font scope for each argument.

SEE ALSO

man(1), *mandoc*(1), *eqn*(7), *mandoc_char*(7), *mdoc*(7), *roff*(7), *tbl*(7)

HISTORY

The **man** language first appeared as a macro package for the roff typesetting system in Version 7 AT&T UNIX.

The stand-alone implementation that is part of the *mandoc*(1) utility first appeared in OpenBSD 4.6.

AUTHORS

Douglas McIlroy <m.douglas.mcilroy@dartmouth.edu> designed and implemented the original version of these macros, wrote the original version of this manual page, and was the first to use them when he edited volume 1 of the Version 7 AT&T UNIX manual pages.

James Clark later rewrote the macros for groff. Eric S. Raymond <esr@thyrsus.com> and Werner Lemberg <wl@gnu.org> added the extended **man** macros to groff in 2007.

The *mandoc*(1) program and this **man** reference were written by Kristaps Dzonsons <kristaps@bsd.lv>.

NAME

mandoc_char — mandoc special characters

DESCRIPTION

This page documents the *roff*(7) escape sequences accepted by *mandoc*(1) to represent special characters in *mdoc*(7) and *man*(7) documents.

The rendering depends on the *mandoc*(1) output mode; it can be inspected by calling *man*(1) on the **mandoc_char** manual page with different `-T` arguments. In ASCII output, the rendering of some characters may be hard to interpret for the reader. Many are rendered as descriptive strings like "<integral>", "<degree>", or "<Gamma>", which may look ugly, and many are replaced by similar ASCII characters. In particular, accented characters are usually shown without the accent. For that reason, try to avoid using any of the special characters documented here except those discussed in the “DESCRIPTION”, unless they are essential for explaining the subject matter at hand, for example when documenting complicated mathematical functions.

In particular, in English manual pages, do not use special-character escape sequences to represent national language characters in author names; instead, provide ASCII transcriptions of the names.

Dashes and Hyphens

In typography there are different types of dashes of various width: the hyphen (-), the en-dash (–), the em-dash (—), and the mathematical minus sign (−).

Hyphens are used for adjectives; to separate the two parts of a compound word; or to separate a word across two successive lines of text. The hyphen does not need to be escaped:

```
blue-eyed
lorry-driver
```

The en-dash is used to separate the two elements of a range, or can be used the same way as an em-dash. It should be written as ‘\(en’:

```
pp. 95\(en97.
Go away \(en or else!
```

The em-dash can be used to show an interruption or can be used the same way as colons, semi-colons, or parentheses. It should be written as ‘\(em’:

```
Three things \(em apples, oranges, and bananas.
This is not that \(em rather, this is that.
```

In *roff*(7) documents, the minus sign is normally written as ‘\-’. In manual pages, some style guides recommend to also use ‘\-’ if an ASCII 0x2d “hyphen-minus” output glyph that can be copied and pasted is desired in output modes supporting it, for example in `-T utf8` and `-T html`. But currently, no practically relevant manual page formatter requires that subtlety, so in manual pages, it is sufficient to write plain ‘-’ to represent hyphen, minus, and hyphen-minus.

If a word on a text input line contains a hyphen, a formatter may decide to insert an output line break after the hyphen if that helps filling the current output line, but the whole word would overflow the line. If it is important that the word is not broken across lines in this way, a zero-width space (‘\&’) can be inserted before or after the hyphen. While *mandoc*(1) never breaks the output line after hyphens adjacent to a zero-width space, after any of the other dash- or hyphen-like characters represented by escape sequences, or after hyphens inside words in macro arguments, other software may not respect these rules and may break the line even in such cases.

Some *roff*(7) implementations contains dictionaries allowing to break the line at syllable boundaries even inside words that contain no hyphens. Such automatic hyphenation is not supported by *mandoc*(1), which only breaks the line at whitespace, and inside words only after existing hyphens.

Spaces

To separate words in normal text, for indenting and alignment in literal context, and when none of the following special cases apply, just use the normal space character (‘ ’).

When filling text, output lines may be broken between words, i.e. at space characters. To prevent a line break between two particular words, use the unpaddable non-breaking space escape sequence (‘\ ’) instead of the normal space character. For example, the input string “number\ 1” will be kept together as “number 1” on the same output line.

On request and macro lines, the normal space character serves as an argument delimiter. To include whitespace into arguments, quoting is usually the best choice; see the MACRO SYNTAX section in *roff*(7). In some cases, using the non-breaking space escape sequence (‘\ ’) may be preferable.

To escape macro names and to protect whitespace at the end of input lines, the zero-width space (‘\&’) is often useful. For example, in *mdoc*(7), a normal space character can be displayed in single quotes in either of the following ways:

```
.Sq " "
.Sq \ \&
```

Quotes

On request and macro lines, the double-quote character (‘"’) is handled specially to allow quoting. One way to prevent this special handling is by using the ‘\(dq’ escape sequence.

Note that on text lines, literal double-quote characters can be used verbatim. All other quote-like characters can be used verbatim as well, even on request and macro lines.

Accents

In output modes supporting such special output characters, for example `-T pdf`, and sometimes less consistently in `-T utf8`, some *roff*(7) formatters convert the following ASCII input characters to the following Unicode special output characters:

`	U+2018	left single quotation mark
'	U+2019	right single quotation mark
~	U+02DC	small tilde
^	U+02C6	modifier letter circumflex

In prose, this automatic substitution is often desirable; but when these characters have to be displayed as plain ASCII characters, for example in source code samples, they require escaping to render as follows:

\(ga	U+0060	grave accent
\(aq	U+0027	apostrophe
\(ti	U+007E	tilde
\(ha	U+005E	circumflex accent

Periods

The period (‘.’) is handled specially at the beginning of an input line, where it introduces a *roff*(7) request or a macro, and when appearing alone as a macro argument in *mdoc*(7). In such situations, prepend a zero-width space (‘\&.’) to make it behave like normal text.

Do not use the ‘\.’ escape sequence. It does not prevent special handling of the period.

Backslashes

To include a literal backslash (‘\’) into the output, use the (‘\e’) escape sequence.

Note that doubling it (‘\\’) is not the right way to output a backslash. Because *mandoc*(1) does not implement full *roff*(7) functionality, it may work with *mandoc*(1), but it may have weird effects on complete *roff*(7) implementations.

SPECIAL CHARACTERS

Special characters are encoded as ‘\X’ (for a one-character escape), ‘\(XX’ (two-character), and ‘\[N]’ (N-character). For details, see the *Special Characters* subsection of the *roff*(7) manual.

Spaces, non-breaking unless stated otherwise:

Input	*Description*
‘\	unpaddable space’
\~	paddable space
\0	digit-width space
|	one-sixth \(em narrow space, zero width in nroff mode
\^	one-twelfth \(em half-narrow space, zero width in nroff
\&	zero-width space
\)	zero-width space transparent to end-of-sentence detection
\%	zero-width space allowing hyphenation
\:	zero-width space allowing line break

Lines:

Input	*Rendered*	*Description*
\(ba	|	bar
\(br	│	box rule
\(ul	_	underscore
\(ru	_	underscore (width 0.5m)
\(rn	‾	overline
\(bb	¦	broken bar
\(sl	/	forward slash
\(rs	\	backward slash

Text markers:

Input	*Rendered*	*Description*
\(ci	○	circle
\(bu	•	bullet
\(dd	‡	double dagger
\(dg	†	dagger
\(lz	◊	lozenge
\(sq	□	white square
\(ps	¶	paragraph
\(sc	§	section
\(lh	☜	left hand
\(rh	☞	right hand
\(at	@	at
\(sh	#	hash (pound)
\(CR	↵	carriage return
\(OK	✓	check mark
\(CL	♣	club suit
\(SP	♠	spade suit
\(HE	♥	heart suit
\(DI	♦	diamond suit

Legal symbols:

Input	*Rendered*	*Description*
\(co	©	copyright
\(rg	®	registered
\(tm	™	trademarked

Punctuation:

Input	*Rendered*	*Description*
\(em	—	em-dash
\(en	–	en-dash

\(hy	-	hyphen
\e	\	back-slash
\.	.	period
\(r!	¡	upside-down exclamation
\(r?	¿	upside-down question

Quotes:

Input	*Rendered*	*Description*
\(Bq	„	right low double-quote
\(bq	‚	right low single-quote
\(lq	“	left double-quote
\(rq	”	right double-quote
\(oq	‘	left single-quote
\(cq	’	right single-quote
\(aq	'	apostrophe quote (ASCII character)
\(dq	"	double quote (ASCII character)
\(Fo	«	left guillemet
\(Fc	»	right guillemet
\(fo	‹	left single guillemet
\(fc	›	right single guillemet

Brackets:

Input	*Rendered*	*Description*
\(lB	[	left bracket
\(rB	]	right bracket
\(lC	{	left brace
\(rC	}	right brace
\(la	⟨	left angle
\(ra	⟩	right angle
\(bv	⎪	brace extension (special font)
\[braceex]	⎪	brace extension
\[bracketlefttp]	⎡	top-left hooked bracket
\[bracketleftbt]	⎣	bottom-left hooked bracket
\[bracketleftex]	⎢	left hooked bracket extension
\[bracketrighttp]	⎤	top-right hooked bracket
\[bracketrightbt]	⎦	bottom-right hooked bracket
\[bracketrightex]	⎥	right hooked bracket extension
\(lt	⎧	top-left hooked brace
\[bracelefttp]	⎧	top-left hooked brace
\(lk	⎨	mid-left hooked brace
\[braceleftmid]	⎨	mid-left hooked brace
\(lb	⎩	bottom-left hooked brace
\[braceleftbt]	⎩	bottom-left hooked brace
\[braceleftex]	⎪	left hooked brace extension
\(rt	⎫	top-left hooked brace
\[bracerighttp]	⎫	top-right hooked brace
\(rk	⎬	mid-right hooked brace
\[bracerightmid]	⎬	mid-right hooked brace
\(rb	⎭	bottom-right hooked brace
\[bracerightbt]	⎭	bottom-right hooked brace
\[bracerightex]	⎪	right hooked brace extension
\[parenlefttp]	⎛	top-left hooked parenthesis
\[parenleftbt]	⎝	bottom-left hooked parenthesis

\[parenleftex]	⎪	left hooked parenthesis extension
\[parenrighttp]	⎫	top-right hooked parenthesis
\[parenrightbt]	⎭	bottom-right hooked parenthesis
\[parenrightex]	⎪	right hooked parenthesis extension

Arrows:

Input	*Rendered*	*Description*
\(<-	←	left arrow
\(->	→	right arrow
\(<>	↔	left-right arrow
\(da	↓	down arrow
\(ua	↑	up arrow
\(va	↕	up-down arrow
\(lA	⇐	left double-arrow
\(rA	⇒	right double-arrow
\(hA	⇔	left-right double-arrow
\(uA	⇑	up double-arrow
\(dA	⇓	down double-arrow
\(vA		up-down double-arrow
\(an	—	horizontal arrow extension

Logical:

Input	*Rendered*	*Description*
\(AN	∧	logical and
\(OR	∨	logical or
\[tno]	¬	logical not (text font)
\(no	¬	logical not (special font)
\(te	∃	existential quantifier
\(fa	∀	universal quantifier
\(st	∋	such that
\(tf	∴	therefore
\(3d	∴	therefore
\(or	|	bitwise or

Mathematical:

Input	*Rendered*	*Description*
\-	−	minus (text font)
\(mi	−	minus (special font)
+	+	plus (text font)
\(pl	+	plus (special font)
\(-+		minus-plus
\[t+-]	±	plus-minus (text font)
\(+-	±	plus-minus (special font)
\(pc	·	center-dot
\[tmu]	×	multiply (text font)
\(mu	×	multiply (special font)
\(c*	⊗	circle-multiply
\(c+	⊕	circle-plus
\[tdi]	÷	divide (text font)
\(di	÷	divide (special font)
\(f/	⁄	fraction
\(**	*	asterisk
\(<=	≤	less-than-equal

\(>=	≥	greater-than-equal
\(<<	≪	much less
\(>>	≫	much greater
\(eq	=	equal
\(!=	≠	not equal
\(==	≡	equivalent
\(ne	≢	not equivalent
\(ap	~	tilde operator
\(|=	≃	asymptotically equal
\(=~	≅	approximately equal
\(~~	≈	almost equal
\(~=	≈	almost equal
\(pt	∝	proportionate
\(es	∅	empty set
\(mo	∈	element
\(nm	∉	not element
\(sb	⊂	proper subset
\(nb	⊄	not subset
\(sp	⊃	proper superset
\(nc	⊅	not superset
\(ib	⊆	reflexive subset
\(ip	⊇	reflexive superset
\(ca	∩	intersection
\(cu	∪	union
\(/_	∠	angle
\(pp	⊥	perpendicular
\(is	∫	integral
\[integral]	∫	integral
\[sum]	∑	summation
\[product]	∏	product
\[coproduct]		coproduct
\(gr	∇	gradient
\(sr	√	square root
\[sqrt]	√	square root
\(lc	⌈	left-ceiling
\(rc	⌉	right-ceiling
\(lf	⌊	left-floor
\(rf	⌋	right-floor
\(if	∞	infinity
\(Ah	ℵ	aleph
\(Im	ℑ	imaginary
\(Re	ℜ	real
\(wp	℘	Weierstrass p
\(pd	∂	partial differential
\(-h	ℏ	Planck constant over 2π
\[hbar]	ℏ	Planck constant over 2π
\(12	½	one-half
\(14	¼	one-fourth
\(34	¾	three-fourths
\(18	⅛	one-eighth
\(38	⅜	three-eighths

\(58	⅝	five-eighths
\(78	⅞	seven-eighths
\(S1	¹	superscript 1
\(S2	²	superscript 2
\(S3	³	superscript 3

Ligatures:

Input	*Rendered*	*Description*
\(ff	ff	ff ligature
\(fi	fi	fi ligature
\(fl	fl	fl ligature
\(Fi	ffi	ffi ligature
\(Fl	ffl	ffl ligature
\(AE	Æ	AE
\(ae	æ	ae
\(OE	Œ	OE
\(oe	œ	oe
\(ss	ß	German eszett
\(IJ	IJ	IJ ligature
\(ij	ij	ij ligature

Accents:

Input	*Rendered*	*Description*
\(a"	˝	Hungarian umlaut
\(a-	¯	macron
\(a.	˙	dotted
\(aˆ	^	circumflex
\(aa	´	acute
\'	´	acute
\(ga	`	grave
\`	`	grave
\(ab	˘	breve
\(ac	¸	cedilla
\(ad	¨	dieresis
\(ah	ˇ	caron
\(ao	°	ring
\(a~	~	tilde
\(ho	˛	ogonek
\(ha	^	hat (ASCII character)
\(ti	~	tilde (ASCII character)

Accented letters:

Input	*Rendered*	*Description*
\('A	Á	acute A
\('E	É	acute E
\('I	Í	acute I
\('O	Ó	acute O
\('U	Ú	acute U
\('Y	Ý	acute Y
\('a	á	acute a
\('e	é	acute e
\('i	í	acute i
\('o	ó	acute o

\('u	ú	acute u
\('y	ý	acute y
\(`A	À	grave A
\(`E	È	grave E
\(`I	Ì	grave I
\(`O	Ò	grave O
\(`U	Ù	grave U
\(`a	à	grave a
\(`e	è	grave e
\(`i	ì	grave i
\(`o	ì	grave o
\(`u	ù	grave u
\(~A	Ã	tilde A
\(~N	Ñ	tilde N
\(~O	Õ	tilde O
\(~a	ã	tilde a
\(~n	ñ	tilde n
\(~o	õ	tilde o
\(:A	Ä	dieresis A
\(:E	Ë	dieresis E
\(:I	Ï	dieresis I
\(:O	Ö	dieresis O
\(:U	Ü	dieresis U
\(:a	ä	dieresis a
\(:e	ë	dieresis e
\(:i	ï	dieresis i
\(:o	ö	dieresis o
\(:u	ü	dieresis u
\(:y	ÿ	dieresis y
\(^A	Â	circumflex A
\(^E	Ê	circumflex E
\(^I	Î	circumflex I
\(^O	Ô	circumflex O
\(^U	Û	circumflex U
\(^a	â	circumflex a
\(^e	ê	circumflex e
\(^i	î	circumflex i
\(^o	ô	circumflex o
\(^u	û	circumflex u
\(,C	Ç	cedilla C
\(,c	ç	cedilla c
\(/L	Ł	stroke L
\(/l	ł	stroke l
\(/O	Ø	stroke O
\(/o	ø	stroke o
\(oA	Å	ring A
\(oa	å	ring a

Special letters:

Input	*Rendered*	*Description*
\(-D	Đ	Eth
\(Sd	ð	eth

\(TP	Þ	Thorn
\(Tp	þ	thorn
\(.i	ı	dotless i
\(.j		dotless j

Currency:

Input	*Rendered*	*Description*
\(Do	$	dollar
\(ct	¢	cent
\(Eu	€	Euro symbol
\(eu	⃀	Euro symbol
\(Ye	¥	yen
\(Po	£	pound
\(Cs	¤	Scandinavian
\(Fn	*ƒ*	florin

Units:

Input	*Rendered*	*Description*
\(de	°	degree
\(%0	‰	per-thousand
\(fm	′	minute
\(sd	″	second
\(mc	µ	micro
\(Of	ª	Spanish female ordinal
\(Om	º	Spanish masculine ordinal

Greek letters:

Input	*Rendered*	*Description*
\(*A	Α	Alpha
\(*B	Β	Beta
\(*G	Γ	Gamma
\(*D	Δ	Delta
\(*E	Ε	Epsilon
\(*Z	Ζ	Zeta
\(*Y	Η	Eta
\(*H	Θ	Theta
\(*I	Ι	Iota
\(*K	Κ	Kappa
\(*L	Λ	Lambda
\(*M	Μ	Mu
\(*N	Ν	Nu
\(*C	Ξ	Xi
\(*O	Ο	Omicron
\(*P	Π	Pi
\(*R	Ρ	Rho
\(*S	Σ	Sigma
\(*T	Τ	Tau
\(*U	ϒ	Upsilon
\(*F	Φ	Phi
\(*X	Χ	Chi
\(*Q	Ψ	Psi
\(*W	Ω	Omega
\(*a	*α*	alpha

\(*b	β	beta
\(*g	γ	gamma
\(*d	δ	delta
\(*e	ε	epsilon
\(*z	ζ	zeta
\(*y	η	eta
\(*h	θ	theta
\(*i	ι	iota
\(*k	κ	kappa
\(*l	λ	lambda
\(*m	µ	mu
\(*n	ν	nu
\(*c	ξ	xi
\(*o	o	omicron
\(*p	π	pi
\(*r	ρ	rho
\(*s	σ	sigma
\(*t	τ	tau
\(*u	υ	upsilon
\(*f	ϕ	phi
\(*x	χ	chi
\(*q	ψ	psi
\(*w	ω	omega
\(+h	ϑ	theta variant
\(+f	φ	phi variant
\(+p	ϖ	pi variant
\(+e		epsilon variant
\(ts	ς	sigma terminal

PREDEFINED STRINGS

Predefined strings are inherited from the macro packages of historical troff implementations. They are *not recommended* for use, as they differ across implementations. Manuals using these predefined strings are almost certainly not portable.

Their syntax is similar to special characters, using '*X' (for a one-character escape), '*(XX' (two-character), and '*[N]' (N-character).

Input	*Rendered*	*Description*
*(Ba	|	vertical bar
*(Ne	≠	not equal
*(Ge	≥	greater-than-equal
*(Le	≤	less-than-equal
*(Gt	>	greater-than
*(Lt	<	less-than
*(Pm	±	plus-minus
*(If	∞	infinity
*(Pi	π	pi
*(Na	*NaN*	NaN
*(Am	&	ampersand
*R		restricted mark
*(Tm		trade mark
*q	"	double-quote
*(Rq	”	right-double-quote

*(Lq	“	left-double-quote
*(lp	(	right-parenthesis
*(rp	)	left-parenthesis
*(lq		left double-quote
*(rq		right double-quote
*(ua	↑	up arrow
*(va		up-down arrow
*(<=	≤	less-than-equal
*(>=	≥	greater-than-equal
*(aa	´	acute
*(ga	`	grave
*(Px	POSIX	POSIX standard name
*(Ai	ANSI	ANSI standard name

UNICODE CHARACTERS

The escape sequences

`\[uXXXX] and \C'uXXXX'`

are interpreted as Unicode codepoints. The codepoint must be in the range above U+0080 and less than U+10FFFF. For compatibility, the hexadecimal digits ‘A’ to ‘F’ must be given as uppercase characters, and points must be zero-padded to four characters; if greater than four characters, no zero padding is allowed. Unicode surrogates are not allowed.

NUMBERED CHARACTERS

For backward compatibility with existing manuals, *mandoc*(1) also supports the

`\N'`*`number`*`'` and `\[char`*`number`*`]`

escape sequences, inserting the character *`number`* from the current character set into the output. Of course, this is inherently non-portable and is already marked as deprecated in the Heirloom roff manual; on top of that, the second form is a GNU extension. For example, do not use \N'34' or \[char34], use \(dq, or even the plain ‘"’ character where possible.

COMPATIBILITY

This section documents compatibility between mandoc and other troff implementations, at this time limited to GNU troff ("groff").

- The \N" escape sequence is limited to printable characters; in groff, it accepts arbitrary character numbers.
- In `-Tascii`, the \(ss, \(nm, \(nb, \(nc, \(ib, \(ip, \(pp, \[sum], \[product], \[coproduct], \(gr, \(-h, and \(a. special characters render differently between mandoc and groff.
- In `-Thtml`, the \(~=, \(nb, and \(nc special characters render differently between mandoc and groff.
- The `-Tps` and `-Tpdf` modes format like `-Tascii` instead of rendering glyphs as in groff.
- The \[radicalex], \[sqrtex], and \(ru special characters have been omitted from mandoc either because they are poorly documented or they have no known representation.

SEE ALSO

mandoc(1), *man*(7), *mdoc*(7), *roff*(7)

AUTHORS

The **`mandoc_char`** manual page was written by Kristaps Dzonsons <kristaps@bsd.lv>.

CAVEATS

The predefined string ‘*(Ba’ mimics the behaviour of the ‘|’ character in *mdoc*(7); thus, if you wish to render a vertical bar with no side effects, use the ‘\(ba’ escape.

NAME

mdoc — semantic markup language for formatting manual pages

DESCRIPTION

The **mdoc** language supports authoring of manual pages for the *man*(1) utility by allowing semantic annotations of words, phrases, page sections and complete manual pages. Such annotations are used by formatting tools to achieve a uniform presentation across all manuals written in **mdoc**, and to support hyperlinking if supported by the output medium.

This reference document describes the structure of manual pages and the syntax and usage of the **mdoc** language. The reference implementation of a parsing and formatting tool is *mandoc*(1); the "COMPATIBILITY" section describes compatibility with other implementations.

In an **mdoc** document, lines beginning with the control character '.' are called "macro lines". The first word is the macro name. It consists of two or three letters. Most macro names begin with a capital letter. For a list of available macros, see "MACRO OVERVIEW". The words following the macro name are arguments to the macro, optionally including the names of other, callable macros; see "MACRO SYNTAX" for details.

Lines not beginning with the control character are called "text lines". They provide free-form text to be printed; the formatting of the text depends on the respective processing context:

```
.Sh Macro lines change control state.
Text lines are interpreted within the current state.
```

Many aspects of the basic syntax of the **mdoc** language are based on the *roff*(7) language; see the *LANGUAGE SYNTAX* and *MACRO SYNTAX* sections in the *roff*(7) manual for details, in particular regarding comments, escape sequences, whitespace, and quoting. However, using *roff*(7) requests in **mdoc** documents is discouraged; *mandoc*(1) supports some of them merely for backward compatibility.

MANUAL STRUCTURE

A well-formed **mdoc** document consists of a document prologue followed by one or more sections.

The prologue, which consists of the **Dd**, **Dt**, and **Os** macros in that order, is required for every document.

The first section (sections are denoted by **Sh**) must be the NAME section, consisting of at least one **Nm** followed by **Nd**.

Following that, convention dictates specifying at least the *SYNOPSIS* and *DESCRIPTION* sections, although this varies between manual sections.

The following is a well-formed skeleton **mdoc** file for a utility "progname":

```
.Dd $Mdocdate$
.Dt PROGNAME section
.Os
.Sh NAME
.Nm progname
.Nd one line about what it does
.\" .Sh LIBRARY
.\" For sections 2, 3, and 9 only.
.\" Not used in OpenBSD.
.Sh SYNOPSIS
.Nm progname
.Op Fl options
.Ar
.Sh DESCRIPTION
The
.Nm
utility processes files ...
.\" .Sh CONTEXT
.\" For section 9 functions only.
```

```
.\" .Sh IMPLEMENTATION NOTES
.\" Not used in OpenBSD.
.\" .Sh RETURN VALUES
.\" For sections 2, 3, and 9 function return values only.
.\" .Sh ENVIRONMENT
.\" For sections 1, 6, 7, and 8 only.
.\" .Sh FILES
.\" .Sh EXIT STATUS
.\" For sections 1, 6, and 8 only.
.\" .Sh EXAMPLES
.\" .Sh DIAGNOSTICS
.\" For sections 1, 4, 6, 7, 8, and 9 printf/stderr messages only.
.\" .Sh ERRORS
.\" For sections 2, 3, 4, and 9 errno settings only.
.\" .Sh SEE ALSO
.\" .Xr foobar 1
.\" .Sh STANDARDS
.\" .Sh HISTORY
.\" .Sh AUTHORS
.\" .Sh CAVEATS
.\" .Sh BUGS
.\" .Sh SECURITY CONSIDERATIONS
.\" Not used in OpenBSD.
```

The sections in an **mdoc** document are conventionally ordered as they appear above. Sections should be composed as follows:

NAME
The name(s) and a one line description of the documented material. The syntax for this as follows:

```
.Nm name0 ,
.Nm name1 ,
.Nm name2
.Nd a one line description
```

Multiple ‘Nm’ names should be separated by commas.

The **Nm** macro(s) must precede the **Nd** macro.

See **Nm** and **Nd**.

LIBRARY
The name of the library containing the documented material, which is assumed to be a function in a section 2, 3, or 9 manual. The syntax for this is as follows:

```
.Lb libarm
```

See **Lb**.

SYNOPSIS
Documents the utility invocation syntax, function call syntax, or device configuration.

For the first, utilities (sections 1, 6, and 8), this is generally structured as follows:

```
.Nm bar
.Op Fl v
.Op Fl o Ar file
.Op Ar
.Nm foo
.Op Fl v
.Op Fl o Ar file
```

```
.Op Ar
```

Commands should be ordered alphabetically.

For the second, function calls (sections 2, 3, 9):

```
.In header.h
.Vt extern const char *global;
.Ft "char *"
.Fn foo "const char *src"
.Ft "char *"
.Fn bar "const char *src"
```

Ordering of **In**, **Vt**, **Fn**, and **Fo** macros should follow C header-file conventions.

And for the third, configurations (section 4):

```
.Cd "it* at isa? port 0x2e"
.Cd "it* at isa? port 0x4e"
```

Manuals not in these sections generally don't need a *SYNOPSIS*.

Some macros are displayed differently in the *SYNOPSIS* section, particularly **Nm**, **Cd**, **Fd**, **Fn**, **Fo**, **In**, **Vt**, and **Ft**. All of these macros are output on their own line. If two such dissimilar macros are pairwise invoked (except for **Ft** before **Fo** or **Fn**), they are separated by a vertical space, unless in the case of **Fo**, **Fn**, and **Ft**, which are always separated by vertical space.

When text and macros following an **Nm** macro starting an input line span multiple output lines, all output lines but the first will be indented to align with the text immediately following the **Nm** macro, up to the next **Nm**, **Sh**, or **Ss** macro or the end of an enclosing block, whichever comes first.

DESCRIPTION
This begins with an expansion of the brief, one line description in *NAME*:

```
The
.Nm
utility does this, that, and the other.
```

It usually follows with a breakdown of the options (if documenting a command), such as:

```
The options are as follows:
.Bl -tag -width Ds
.It Fl v
Print verbose information.
.El
```

List the options in alphabetical order, uppercase before lowercase for each letter and with no regard to whether an option takes an argument. Put digits in ascending order before all letter options.

Manuals not documenting a command won't include the above fragment.

Since the *DESCRIPTION* section usually contains most of the text of a manual, longer manuals often use the **Ss** macro to form subsections. In very long manuals, the *DESCRIPTION* may be split into multiple sections, each started by an **Sh** macro followed by a non-standard section name, and each having several subsections, like in the present **mdoc** manual.

CONTEXT
This section lists the contexts in which functions can be called in section 9. The contexts are autoconf, process, or interrupt.

IMPLEMENTATION NOTES
Implementation-specific notes should be kept here. This is useful when implementing standard functions that may have side effects or notable algorithmic implications.

RETURN VALUES
This section documents the return values of functions in sections 2, 3, and 9.

See **Rv**.

ENVIRONMENT
Lists the environment variables used by the utility, and explains the syntax and semantics of their values. The *environ*(7) manual provides examples of typical content and formatting.

See **Ev**.

FILES
Documents files used. It's helpful to document both the file name and a short description of how the file is used (created, modified, etc.).

See **Pa**.

EXIT STATUS
This section documents the command exit status for section 1, 6, and 8 utilities. Historically, this information was described in *DIAGNOSTICS*, a practise that is now discouraged.

See **Ex**.

EXAMPLES
Example usages. This often contains snippets of well-formed, well-tested invocations. Make sure that examples work properly!

DIAGNOSTICS
Documents error messages. In section 4 and 9 manuals, these are usually messages printed by the kernel to the console and to the kernel log. In section 1, 6, 7, and 8, these are usually messages printed by userland programs to the standard error output.

Historically, this section was used in place of *EXIT STATUS* for manuals in sections 1, 6, and 8; however, this practise is discouraged.

See **Bl** `-diag`.

ERRORS
Documents *errno*(2) settings in sections 2, 3, 4, and 9.

See **Er**.

SEE ALSO
References other manuals with related topics. This section should exist for most manuals. Cross-references should conventionally be ordered first by section, then alphabetically (ignoring case).

References to other documentation concerning the topic of the manual page, for example authoritative books or journal articles, may also be provided in this section.

See **Rs** and **Xr**.

STANDARDS
References any standards implemented or used. If not adhering to any standards, the *HISTORY* section should be used instead.

See **St**.

HISTORY
A brief history of the subject, including where it was first implemented, and when it was ported to or reimplemented for the operating system at hand.

AUTHORS
Credits to the person or persons who wrote the code and/or documentation. Authors should generally be noted by both name and email address.

See **An**.

CAVEATS
Common misuses and misunderstandings should be explained in this section.

BUGS
Known bugs, limitations, and work-arounds should be described in this section.

SECURITY CONSIDERATIONS
Documents any security precautions that operators should consider.

MACRO OVERVIEW

This overview is sorted such that macros of similar purpose are listed together, to help find the best macro for any given purpose. Deprecated macros are not included in the overview, but can be found below in the alphabetical "MACRO REFERENCE".

Document preamble and NAME section macros

Dd	document date: `$Mdocdate$` | *month day*, *year*
Dt	document title: *TITLE section* [*arch*]
Os	operating system version: [*system* [*version*]]
Nm	document name (one argument)
Nd	document description (one line)

Sections and cross references

Sh	section header (one line)
Ss	subsection header (one line)
Sx	internal cross reference to a section or subsection
Xr	cross reference to another manual page: *name section*
Tg	tag the definition of a *term* (<= 1 arguments)
Pp	start a text paragraph (no arguments)

Displays and lists

Bd, **Ed**	display block: *-type* [`-offset` *width*] [`-compact`]
D1	indented display (one line)
Dl	indented literal display (one line)
Ql	in-line literal display: `text`
Bl, **El**	list block: *-type* [`-width` *val*] [`-offset` *val*] [`-compact`]
It	list item (syntax depends on *-type*)
Ta	table cell separator in **Bl** `-column` lists
Rs, **%***, **Re**	bibliographic block (references)

Spacing control

Pf	prefix, no following horizontal space (one argument)
Ns	roman font, no preceding horizontal space (no arguments)
Ap	apostrophe without surrounding whitespace (no arguments)
Sm	switch horizontal spacing mode: [`on` | `off`]
Bk, **Ek**	keep block: `-words`

Semantic markup for command line utilities

Nm	start a SYNOPSIS block with the name of a utility
Fl	command line options (flags) (>=0 arguments)
Cm	command modifier (>0 arguments)
Ar	command arguments (>=0 arguments)
Op, **Oo**, **Oc**	optional syntax elements (enclosure)
Ic	internal or interactive command (>0 arguments)
Ev	environmental variable (>0 arguments)

Pa file system path (>=0 arguments)

Semantic markup for function libraries

Lb function library (one argument)
In include file (one argument)
Fd other preprocessor directive (>0 arguments)
Ft function type (>0 arguments)
Fo, **Fc** function block: *funcname*
Fn function name: *funcname* [*argument ...*]
Fa function argument (>0 arguments)
Vt variable type (>0 arguments)
Va variable name (>0 arguments)
Dv defined variable or preprocessor constant (>0 arguments)
Er error constant (>0 arguments)
Ev environmental variable (>0 arguments)

Various semantic markup

An author name (>0 arguments)
Lk hyperlink: *uri* [*display_name*]
Mt “mailto” hyperlink: *localpart@domain*
Cd kernel configuration declaration (>0 arguments)
Ad memory address (>0 arguments)
Ms mathematical symbol (>0 arguments)

Physical markup

Em italic font or underline (emphasis) (>0 arguments)
Sy boldface font (symbolic) (>0 arguments)
No return to roman font (normal) (>0 arguments)
Bf, **Ef** font block: *−type* | Em | Li | Sy

Physical enclosures

Dq, **Do**, **Dc** enclose in typographic double quotes: “text”
Qq, **Qo**, **Qc** enclose in typewriter double quotes: "text"
Sq, **So**, **Sc** enclose in single quotes: ‘text’
Pq, **Po**, **Pc** enclose in parentheses: (text)
Bq, **Bo**, **Bc** enclose in square brackets: [text]
Brq, **Bro**, **Brc** enclose in curly braces: {text}
Aq, **Ao**, **Ac** enclose in angle brackets: ⟨text⟩
Eo, **Ec** generic enclosure

Text production

Ex −std standard command exit values: [*utility ...*]
Rv −std standard function return values: [*function ...*]
St reference to a standards document (one argument)
At AT&T UNIX
Bx BSD
Bsx BSD/OS
Nx NetBSD
Fx FreeBSD
Ox OpenBSD
Dx DragonFly

MACRO REFERENCE

This section is a canonical reference of all macros, arranged alphabetically. For the scoping of individual macros, see “MACRO SYNTAX”.

%A *first_name ... last_name*
Author name of an **Rs** block. Multiple authors should each be accorded their own **%A** line. Author names should be ordered with full or abbreviated forename(s) first, then full surname.

%B *title*
Book title of an **Rs** block. This macro may also be used in a non-bibliographic context when referring to book titles.

%C *location*
Publication city or location of an **Rs** block.

%D [*month day*,] *year*
Publication date of an **Rs** block. Provide the full English name of the *month* and all four digits of the *year*.

%I *name*
Publisher or issuer name of an **Rs** block.

%J *name*
Journal name of an **Rs** block.

%N *number*
Issue number (usually for journals) of an **Rs** block.

%O *line*
Optional information of an **Rs** block.

%P *number*
Book or journal page number of an **Rs** block. Conventionally, the argument starts with '`p.`' for a single page or `pp.` for a range of pages, for example:

```
.%P pp. 42\(en47
```

%Q *name*
Institutional author (school, government, etc.) of an **Rs** block. Multiple institutional authors should each be accorded their own **%Q** line.

%R *name*
Technical report name of an **Rs** block.

%T *title*
Article title of an **Rs** block. This macro may also be used in a non-bibliographical context when referring to article titles.

%U *protocol*://*path*
URI of reference document.

%V *number*
Volume number of an **Rs** block.

Ac Close an **Ao** block. Does not have any tail arguments.

Ad *address*
Memory address. Do not use this for postal addresses.

Examples:

```
.Ad [0,$]
.Ad 0x00000000
```

An `-split` | `-nosplit` | *first_name ... last_name*
Author name. Can be used both for the authors of the program, function, or driver documented in the manual, or for the authors of the manual itself. Requires either the name of an author or one of the following arguments:

`-split` Start a new output line before each subsequent invocation of **An**.
`-nosplit` The opposite of `-split`.

The default is `-nosplit`. The effect of selecting either of the `-split` modes ends at the beginning of the *AUTHORS* section. In the *AUTHORS* section, the default is `-nosplit` for the first author listing and `-split` for all other author listings.

Examples:

```
.An -nosplit
.An Kristaps Dzonsons Aq Mt kristaps@bsd.lv
```

Ao *block*

Begin a block enclosed by angle brackets. Does not have any head arguments. This macro is almost never useful. See **Aq** for more details.

Ap Inserts an apostrophe without any surrounding whitespace. This is generally used as a grammatical device when referring to the verb form of a function.

Examples:

```
.Fn execve Ap d
```

Aq *line*

Enclose the rest of the input line in angle brackets. The only important use case is for email addresses. See **Mt** for an example.

Occasionally, it is used for names of characters and keys, for example:

```
Press the
.Aq escape
key to ...
```

For URIs, use **Lk** instead, and **In** for "#include" directives. Never wrap **Ar** in **Aq**.

Since **Aq** usually renders with non-ASCII characters in non-ASCII output modes, do not use it where the ASCII characters '<' and '>' are required as syntax elements. Instead, use these characters directly in such cases, combining them with the macros **Pf**, **Ns**, or **Eo** as needed.

See also **Ao**.

Ar [*placeholder ...*]

Command arguments. If an argument is not provided, the string "file ..." is used as a default.

Examples:

```
.Fl o Ar file
.Ar
.Ar arg1 , arg2 .
```

The arguments to the **Ar** macro are names and placeholders for command arguments; for fixed strings to be passed verbatim as arguments, use **Fl** or **Cm**.

At [*version*]

Formats an AT&T UNIX version. Accepts one optional argument:

`v[1-7]` | `32v` A version of AT&T UNIX.
`III` AT&T System III UNIX.
`V` | `V.[1-4]` A version of AT&T System V UNIX.

Note that these arguments do not begin with a hyphen.

Examples:

```
.At
.At III
.At V.1
```

See also **Bsx**, **Bx**, **Dx**, **Fx**, **Nx**, and **Ox**.

Bc Close a **Bo** block. Does not have any tail arguments.

Bd `-`*type* [`-offset` *width*][`-compact`]
Begin a display block. Display blocks are used to select a different indentation and justification than the one used by the surrounding text. They may contain both macro lines and text lines. By default, a display block is preceded by a vertical space.

The *type* must be one of the following:

`-centered` Produce one output line from each input line, and center-justify each line. Using this display type is not recommended; many **mdoc** implementations render it poorly.

`-filled` Change the positions of line breaks to fill each line, and left- and right-justify the resulting block.

`-literal` Produce one output line from each input line, and do not justify the block at all. Preserve white space as it appears in the input. Always use a constant-width font. Use this for displaying source code.

`-ragged` Change the positions of line breaks to fill each line, and left-justify the resulting block.

`-unfilled` The same as `-literal`, but using the same font as for normal text, which is a variable width font if supported by the output device.

The *type* must be provided first. Additional arguments may follow:

`-offset` *width*
Indent the display by the *width*, which may be one of the following:

One of the pre-defined strings `indent`, the width of a standard indentation (six constant width characters); `indent-two`, twice `indent`; `left`, which has no effect; `right`, which justifies to the right margin; or `center`, which aligns around an imagined center axis.

A macro invocation, which selects a predefined width associated with that macro. The most popular is the imaginary macro *Ds*, which resolves to **6n**.

A scaling width as described in *roff*(7).

An arbitrary string, which indents by the length of this string.

When the argument is missing, `-offset` is ignored.

`-compact` Do not assert vertical space before the display.

Examples:

```
.Bd -literal -offset indent -compact
   Hello       world.
.Ed
```

See also **D1** and **Dl**.

Bf `-emphasis` | `-literal` | `-symbolic` | `Em` | `Li` | `Sy`
Change the font mode for a scoped block of text. The `-emphasis` and `Em` argument are equivalent, as are `-symbolic` and `Sy`, and `-literal` and `Li`. Without an argument, this macro does nothing. The font mode continues until broken by a new font mode in a nested scope or **Ef** is encountered.

See also **Li**, **Ef**, **Em**, and **Sy**.

Bk `-words`
For each macro, keep its output together on the same output line, until the end of the macro or the end of the input line is reached, whichever comes first. Line breaks in text lines are unaffected.

The `-words` argument is required; additional arguments are ignored.

The following example will not break within each **Op** macro line:

```
.Bk -words
.Op Fl f Ar flags
.Op Fl o Ar output
.Ek
```

Be careful in using over-long lines within a keep block! Doing so will clobber the right margin.

Bl `-`*type* [`-width` *val*] [`-offset` *val*] [`-compact`] [*col* ...]
Begin a list. Lists consist of items specified using the **It** macro, containing a head or a body or both.

The list *type* is mandatory and must be specified first. The `-width` and `-offset` arguments accept macro names as described for **Bd** `-offset`, scaling widths as described in *roff*(7), or use the length of the given string. The `-offset` is a global indentation for the whole list, affecting both item heads and bodies. For those list types supporting it, the `-width` argument requests an additional indentation of item bodies, to be added to the `-offset`. Unless the `-compact` argument is specified, list entries are separated by vertical space.

A list must specify one of the following list types:

`-bullet` No item heads can be specified, but a bullet will be printed at the head of each item. Item bodies start on the same output line as the bullet and are indented according to the `-width` argument.

`-column` A columnated list. The `-width` argument has no effect; instead, the string length of each argument specifies the width of one column. If the first line of the body of a `-column` list is not an **It** macro line, **It** contexts spanning one input line each are implied until an **It** macro line is encountered, at which point items start being interpreted as described in the **It** documentation.

`-dash` Like `-bullet`, except that dashes are used in place of bullets.

`-diag` Like `-inset`, except that item heads are not parsed for macro invocations. Most often used in the *DIAGNOSTICS* section with error constants in the item heads.

`-enum` A numbered list. No item heads can be specified. Formatted like `-bullet`, except that cardinal numbers are used in place of bullets, starting at 1.

`-hang` Like `-tag`, except that the first lines of item bodies are not indented, but follow the item heads like in `-inset` lists.

`-hyphen` Synonym for `-dash`.

`-inset` Item bodies follow items heads on the same line, using normal inter-word spacing. Bodies are not indented, and the `-width` argument is ignored.

`-item` No item heads can be specified, and none are printed. Bodies are not indented, and the `-width` argument is ignored.

`-ohang` Item bodies start on the line following item heads and are not indented. The `-width` argument is ignored.

`-tag` Item bodies are indented according to the `-width` argument. When an item head fits inside the indentation, the item body follows this head on the same output line. Otherwise, the body starts on the output line following the head.

Lists may be nested within lists and displays. Nesting of `-column` and `-enum` lists may not be portable.

See also **El** and **It**.

Bo *block*
Begin a block enclosed by square brackets. Does not have any head arguments.

Examples:
```
.Bo 1 ,
.Dv BUFSIZ Bc
```

See also **Bq**.

Bq *line*
Encloses its arguments in square brackets.

Examples:
```
.Bq 1, Dv BUFSIZ
```

Remarks: this macro is sometimes abused to emulate optional arguments for commands; the correct macros to use for this purpose are **Op**, **Oo**, and **Oc**.

See also **Bo**.

Brc
Close a **Bro** block. Does not have any tail arguments.

Bro *block*
Begin a block enclosed by curly braces. Does not have any head arguments.

Examples:
```
.Bro 1 , ... ,
.Va n Brc
```

See also **Brq**.

Brq *line*
Encloses its arguments in curly braces.

Examples:
```
.Brq 1, ..., Va n
```

See also **Bro**.

Bsx [*version*]
Format the BSD/OS version provided as an argument, or a default value if no argument is provided.

Examples:
```
.Bsx 1.0
.Bsx
```

See also **At**, **Bx**, **Dx**, **Fx**, **Nx**, and **Ox**.

Bt Supported only for compatibility, do not use this in new manuals. Prints “is currently in beta test.”

Bx [*version* [*variant*]]
Format the BSD version provided as an argument, or a default value if no argument is provided.

Examples:
```
.Bx 4.3 Tahoe
.Bx 4.4
.Bx
```

See also **At**, **Bsx**, **Dx**, **Fx**, **Nx**, and **Ox**.

Cd *line*
Kernel configuration declaration. This denotes strings accepted by *config*(8). It is most often used in section 4 manual pages.

Examples:
```
.Cd device le0 at scode?
```

Remarks: this macro is commonly abused by using quoted literals to retain whitespace and align consecutive **Cd** declarations. This practise is discouraged.

Cm *keyword ...*
Command modifiers. Typically used for fixed strings passed as arguments to interactive commands, to commands in interpreted scripts, or to configuration file directives, unless **Fl** is more appropriate.

Examples:
```
.Nm mt Fl f Ar device Cm rewind
.Nm ps Fl o Cm pid , Ns Cm command
.Nm dd Cm if= Ns Ar file1 Cm of= Ns Ar file2
.Ic set Fl o Cm vi
.Ic lookup Cm file bind
.Ic permit Ar identity Op Cm as Ar target
```

D1 *line*
One-line indented display. This is formatted by the default rules and is useful for simple indented statements. It is followed by a newline.

Examples:
```
.D1 Fl abcdefgh
```

See also **Bd** and **Dl**.

Db This macro is obsolete. No replacement is needed. It is ignored by *mandoc*(1) and groff including its arguments. It was formerly used to toggle a debugging mode.

Dc Close a **Do** block. Does not have any tail arguments.

Dd `$Mdocdate$` | *month day*, *year*
Document date for display in the page footer, by convention the date of the last change. This is the mandatory first macro of any **mdoc** manual.

The *month* is the full English month name, the *day* is an integer number, and the *year* is the full four-digit year.

Other arguments are not portable; the *mandoc*(1) utility handles them as follows:
- To have the date automatically filled in by the OpenBSD version of *cvs*(1), the special string "$Mdocdate$" can be given as an argument.
- The traditional, purely numeric *man*(7) format *year*–*month*–*day* is accepted, too.
- If a date string cannot be parsed, it is used verbatim.
- If no date string is given, the current date is used.

Examples:
```
.Dd $Mdocdate$
.Dd $Mdocdate: July 2 2018$
.Dd July 2, 2018
```

See also **Dt** and **Os**.

Dl *line*
One-line indented display. This is formatted as literal text and is useful for commands and invocations. It is followed by a newline.

Examples:

```
.Dl % mandoc mdoc.7 \(ba less
```

See also **Ql**, **Bd** **-literal**, and **D1**.

Do *block*

Begin a block enclosed by double quotes. Does not have any head arguments.

Examples:

```
.Do
April is the cruellest month
.Dc
\(em T.S. Eliot
```

See also **Dq**.

Dq *line*

Encloses its arguments in “typographic” double-quotes.

Examples:

```
.Dq April is the cruellest month
\(em T.S. Eliot
```

See also **Qq**, **Sq**, and **Do**.

Dt *TITLE section* [*arch*]

Document title for display in the page header. This is the mandatory second macro of any **mdoc** file.

Its arguments are as follows:

TITLE The document’s title (name), defaulting to “UNTITLED” if unspecified. To achieve a uniform appearance of page header lines, it should by convention be all caps.

section The manual section. This may be one of 1 (General Commands), 2 (System Calls), 3 (Library Functions), 3p (Perl Library), 4 (Device Drivers), 5 (File Formats), 6 (Games), 7 (Miscellaneous Information), 8 (System Manager’s Manual), or 9 (Kernel Developer’s Manual). It should correspond to the manual’s filename suffix and defaults to the empty string if unspecified.

arch This specifies the machine architecture a manual page applies to, where relevant, for example alpha, amd64, i386, or sparc64. The list of valid architectures varies by operating system.

Examples:

```
.Dt FOO 1
.Dt FOO 9 i386
```

See also **Dd** and **Os**.

Dv *identifier ...*

Defined variables such as preprocessor constants, constant symbols, enumeration values, and so on.

Examples:

```
.Dv NULL
.Dv BUFSIZ
.Dv STDOUT_FILENO
```

See also **Er** and **Ev** for special-purpose constants, **Va** for variable symbols, and **Fd** for listing preprocessor variable definitions in the *SYNOPSIS*.

Dx [*version*]

Format the DragonFly version provided as an argument, or a default value if no argument is provided.

Examples:

```
.Dx 2.4.1
.Dx
```

See also **At**, **Bsx**, **Bx**, **Fx**, **Nx**, and **Ox**.

Ec [*closing_delimiter*]
Close a scope started by **Eo**.

The *closing_delimiter* argument is used as the enclosure tail, for example, specifying \(rq will emulate **Dc**.

Ed End a display context started by **Bd**.

Ef End a font mode context started by **Bf**.

Ek End a keep context started by **Bk**.

El End a list context started by **Bl**. See also **It**.

Em *word ...*
Request an italic font. If the output device does not provide that, underline.

This is most often used for stress emphasis (not to be confused with importance, see **Sy**). In the rare cases where none of the semantic markup macros fit, it can also be used for technical terms and placeholders, except that for syntax elements, **Sy** and **Ar** are preferred, respectively.

Examples:

```
Selected lines are those
.Em not
matching any of the specified patterns.
Some of the functions use a
.Em hold space
to save the pattern space for subsequent retrieval.
```

See also **No**, **Ql**, and **Sy**.

En *word ...*
This macro is obsolete. Use **Eo** or any of the other enclosure macros.

It encloses its argument in the delimiters specified by the last **Es** macro.

Eo [*opening_delimiter*]
An arbitrary enclosure. The *opening_delimiter* argument is used as the enclosure head, for example, specifying \(lq will emulate **Do**.

Er *identifier ...*
Error constants for definitions of the *errno* libc global variable. This is most often used in section 2 and 3 manual pages.

Examples:

```
.Er EPERM
.Er ENOENT
```

See also **Dv** for general constants.

Es *opening_delimiter closing_delimiter*
This macro is obsolete. Use **Eo** or any of the other enclosure macros.

It takes two arguments, defining the delimiters to be used by subsequent **En** macros.

Ev *identifier ...*
Environmental variables such as those specified in *environ*(7).

Examples:

```
.Ev DISPLAY
.Ev PATH
```

See also **Dv** for general constants.

Ex −std [*utility ...*]

Insert a standard sentence regarding command exit values of 0 on success and >0 on failure. This is most often used in section 1, 6, and 8 manual pages.

If *utility* is not specified, the document's name set by **Nm** is used. Multiple *utility* arguments are treated as separate utilities.

See also **Rv**.

Fa *argument ...*

Function argument or parameter. Each argument may be a name and a type (recommended for the *SYNOPSIS* section), a name alone (for function invocations), or a type alone (for function prototypes). If both a type and a name are given or if the type consists of multiple words, all words belonging to the same function argument have to be given in a single argument to the **Fa** macro.

This macro is also used to specify the field name of a structure.

Most often, the **Fa** macro is used in the *SYNOPSIS* within **Fo** blocks when documenting multi-line function prototypes. If invoked with multiple arguments, the arguments are separated by a comma. Furthermore, if the following macro is another **Fa**, the last argument will also have a trailing comma.

Examples:

```
.Fa "const char *p"
.Fa "int a" "int b" "int c"
.Fa "char *" size_t
```

See also **Fo**.

Fc End a function context started by **Fo**.

Fd #*directive* [*argument ...*]

Preprocessor directive, in particular for listing it in the *SYNOPSIS*. Historically, it was also used to document include files. The latter usage has been deprecated in favour of **In**.

Examples:

```
.Fd #define sa_handler __sigaction_u.__sa_handler
.Fd #define SIO_MAXNFDS
.Fd #ifdef FS_DEBUG
.Ft void
.Fn dbg_open "const char *"
.Fd #endif
```

See also "MANUAL STRUCTURE", **In**, and **Dv**.

Fl [*word ...*]

Command-line flag or option. Used when listing arguments to command-line utilities. For each argument, prints an ASCII hyphen-minus character '−', immediately followed by the argument. If no arguments are provided, a hyphen-minus is printed followed by a space. If the argument is a macro, a hyphen-minus is prefixed to the subsequent macro output.

Examples:

```
.Nm du Op Fl H | L | P
.Nm ls Op Fl 1AaCcdFfgHhikLlmnopqRrSsTtux
.Nm route Cm add Fl inet Ar destination gateway
.Nm locate.updatedb Op Fl \-fcodes Ns = Ns Ar dbfile
.Nm aucat Fl o Fl
.Nm kill Fl Ar signal_number
```

For GNU-sytle long options, escaping the additional hyphen-minus is not strictly required, but may be safer with future versions of GNU troff; see *mandoc_char*(7) for details.

See also **Cm**.

Fn *funcname* [*argument ...*]
A function name.

Function arguments are surrounded in parenthesis and are delimited by commas. If no arguments are specified, blank parenthesis are output. In the *SYNOPSIS* section, this macro starts a new output line, and a blank line is automatically inserted between function definitions.

Examples:

```
.Fn "int funcname" "int arg0" "int arg1"
.Fn funcname "int arg0"
.Fn funcname arg0

.Ft functype
.Fn funcname
```

When referring to a function documented in another manual page, use **Xr** instead. See also "MANUAL STRUCTURE", **Fo**, and **Ft**.

Fo *funcname*
Begin a function block. This is a multi-line version of **Fn**.

Invocations usually occur in the following context:

```
.Ft functype
.Fo funcname
.Fa "argtype argname"
...
.Fc
```

A **Fo** scope is closed by **Fc**.

See also "MANUAL STRUCTURE", **Fa**, **Fc**, and **Ft**.

Fr *number*
This macro is obsolete. No replacement markup is needed.

It was used to show numerical function return values in an italic font.

Ft *functype*
A function type.

In the *SYNOPSIS* section, a new output line is started after this macro.

Examples:

```
.Ft int
.Ft functype
.Fn funcname
```

See also "MANUAL STRUCTURE", **Fn**, and **Fo**.

Fx [*version*]
Format the FreeBSD version provided as an argument, or a default value if no argument is provided.

Examples:

```
.Fx 7.1
.Fx
```

See also **At**, **Bsx**, **Bx**, **Dx**, **Nx**, and **Ox**.

Hf *filename*
This macro is not implemented in *mandoc*(1). It was used to include the contents of a (header) file literally.

Ic *keyword ...*
Internal or interactive command, or configuration instruction in a configuration file. See also **Cm**.

Examples:
```
.Ic :wq
.Ic hash
.Ic alias
```

Note that using **Ql**, **Dl**, or **Bd** `-literal` is preferred for displaying code samples; the **Ic** macro is used when referring to an individual command name.

In *filename*
The name of an include file. This macro is most often used in section 2, 3, and 9 manual pages.

When invoked as the first macro on an input line in the *SYNOPSIS* section, the argument is displayed in angle brackets and preceded by "#include", and a blank line is inserted in front if there is a preceding function declaration. In other sections, it only encloses its argument in angle brackets and causes no line break.

Examples:
```
.In sys/types.h
```

See also “MANUAL STRUCTURE”.

It [*head*]
A list item. The syntax of this macro depends on the list type.

Lists of type `-hang`, `-ohang`, `-inset`, and `-diag` have the following syntax:

.**It** *args*

Lists of type `-bullet`, `-dash`, `-enum`, `-hyphen` and `-item` have the following syntax:

.**It**

with subsequent lines interpreted within the scope of the **It** until either a closing **El** or another **It**.

The `-tag` list has the following syntax:

.**It** [args]

Subsequent lines are interpreted as with `-bullet` and family. The line arguments correspond to the list’s left-hand side; body arguments correspond to the list’s contents.

The `-column` list is the most complicated. Its syntax is as follows:

.**It** *cell* [**Ta** *cell* ...]
.**It** *cell* [<TAB> *cell* ...]

The arguments consist of one or more lines of text and macros representing a complete table line. Cells within the line are delimited by the special **Ta** block macro or by literal tab characters.

Using literal tabs is strongly discouraged because they are very hard to use correctly and **mdoc** code using them is very hard to read. In particular, a blank character is syntactically significant before and after the literal tab character. If a word precedes or follows the tab without an intervening blank, that word is never interpreted as a macro call, but always output literally.

The tab cell delimiter may only be used within the **It** line itself; on following lines, only the **Ta** macro can be used to delimit cells, and portability requires that **Ta** is called by other macros: some parsers do not recognize it when it appears as the first macro on a line.

Note that quoted strings may span tab-delimited cells on an **It** line. For example,

```
.It "col1 , <TAB> col2 ," ;
```

will preserve the whitespace before both commas, but not the whitespace before the semicolon.

See also **Bl**.

Lb lib*name*
Specify a library.

The *name* parameter may be a system library, such as z or pam, in which case a small library description is printed next to the linker invocation; or a custom library, in which case the library name is printed in quotes. This is most commonly used in the *SYNOPSIS* section as described in "MANUAL STRUCTURE".

Examples:

```
.Lb libz
.Lb libmandoc
```

Li *word ...*
Request a typewriter (literal) font. Deprecated because on terminal output devices, this is usually indistinguishable from normal text. For literal displays, use **Ql** (in-line), **Dl** (single line), or **Bd** **-literal** (multi-line) instead.

Lk *uri* [*display_name*]
Format a hyperlink.

Examples:

```
.Lk https://bsd.lv "The BSD.lv Project"
.Lk https://bsd.lv
```

See also **Mt**.

Lp Deprecated synonym for **Pp**.

Ms *name*
Display a mathematical symbol.

Examples:

```
.Ms sigma
.Ms aleph
```

Mt *localpart@domain*
Format a "mailto:" hyperlink.

Examples:

```
.Mt discuss@manpages.bsd.lv
.An Kristaps Dzonsons Aq Mt kristaps@bsd.lv
```

Nd *line*
A one line description of the manual's content. This is the mandatory last macro of the *NAME* section and not appropriate for other sections.

Examples:

```
.Nd mdoc language reference
.Nd format and display UNIX manuals
```

The **Nd** macro technically accepts child macros and terminates with a subsequent **Sh** invocation. Do not assume this behaviour: some *whatis*(1) database generators are not smart enough to parse more than the line arguments and will display macros verbatim.

See also **Nm**.

Nm [*name*]
The name of the manual page, or — in particular in section 1, 6, and 8 pages — of an additional command or feature documented in the manual page. When first invoked, the **Nm** macro expects a single argument, the name of the manual page. Usually, the first invocation happens in the *NAME* section of the page. The specified name will be remembered and used whenever the macro is called again without arguments later in the page. The **Nm** macro uses "Block full-implicit" semantics when invoked as the first macro on an input line in the *SYNOPSIS* section; otherwise, it uses ordinary "In-line" semantics.

Examples:

```
.Sh SYNOPSIS
.Nm cat
.Op Fl benstuv
.Op Ar
```

In the *SYNOPSIS* of section 2, 3 and 9 manual pages, use the **Fn** macro rather than **Nm** to mark up the name of the manual page.

No *word ...*
Normal text. Closes the scope of any preceding in-line macro. When used after physical formatting macros like **Em** or **Sy**, switches back to the standard font face and weight. Can also be used to embed plain text strings in macro lines using semantic annotation macros.

Examples:

```
.Em italic , Sy bold , No and roman
```

```
.Sm off
.Cm :C No / Ar pattern No / Ar replacement No /
.Sm on
```

See also **Em**, **Ql**, and **Sy**.

Ns Suppress a space between the output of the preceding macro and the following text or macro. Following invocation, input is interpreted as normal text just like after an **No** macro.

This has no effect when invoked at the start of a macro line.

Examples:

```
.Ar name Ns = Ns Ar value
.Cm :M Ns Ar pattern
.Fl o Ns Ar output
```

See also **No** and **Sm**.

Nx [*version*]
Format the NetBSD version provided as an argument, or a default value if no argument is provided.

Examples:

```
.Nx 5.01
.Nx
```

See also **At**, **Bsx**, **Bx**, **Dx**, **Fx**, and **Ox**.

Oc Close multi-line **Oo** context.

Oo *block*
Multi-line version of **Op**.

Examples:

```
.Oo
.Op Fl flag Ns Ar value
.Oc
```

Op *line*
Optional part of a command line. Prints the argument(s) in brackets. This is most often used in the *SYNOPSIS* section of section 1 and 8 manual pages.

Examples:
```
.Op Fl a Ar b
.Op Ar a | b
```

See also **Oo**.

Os [*system* [*version*]]
Operating system version for display in the page footer. This is the mandatory third macro of any **mdoc** file.

The optional *system* parameter specifies the relevant operating system or environment. It is suggested to leave it unspecified, in which case *mandoc*(1) uses its `-Ios` argument or, if that isn't specified either, *sysname* and *release* as returned by *uname*(3).

Examples:
```
.Os
.Os KTH/CSC/TCS
.Os BSD 4.3
```

See also **Dd** and **Dt**.

Ot *functype*
This macro is obsolete. Use **Ft** instead; with *mandoc*(1), both have the same effect.

Historical **mdoc** packages described it as "old function type (FORTRAN)".

Ox [*version*]
Format the OpenBSD version provided as an argument, or a default value if no argument is provided.

Examples:
```
.Ox 4.5
.Ox
```

See also **At**, **Bsx**, **Bx**, **Dx**, **Fx**, and **Nx**.

Pa *name ...*
An absolute or relative file system path, or a file or directory name. If an argument is not provided, the character '~' is used as a default.

Examples:
```
.Pa /usr/bin/mandoc
.Pa /usr/share/man/man7/mdoc.7
```

See also **Lk**.

Pc Close parenthesised context opened by **Po**.

Pf *prefix macro* [*argument ...*]
Removes the space between its argument and the following macro. It is equivalent to:

No \&*prefix* **Ns** *macro* [*argument* ...]

The *prefix* argument is not parsed for macro names or delimiters, but used verbatim as if it were escaped.

Examples:
```
.Pf $ Ar variable_name
.Pf . Ar macro_name
.Pf 0x Ar hex_digits
```

See also **Ns** and **Sm**.

Po *block*
Multi-line version of **Pq**.

Pp Break a paragraph. This will assert vertical space between prior and subsequent macros and/or text.

Paragraph breaks are not needed before or after **Sh** or **Ss** macros or before displays (**Bd** *line*) or lists (**Bl**) unless the `-compact` flag is given.

Pq *line*
Parenthesised enclosure.

See also **Po**.

Qc Close quoted context opened by **Qo**.

Ql *line*
In-line literal display. This can be used for complete command invocations and for multi-word code examples when an indented display is not desired.

See also **Dl** and **Bd** `-literal`.

Qo *block*
Multi-line version of **Qq**.

Qq *line*
Encloses its arguments in "typewriter" double-quotes. Consider using **Dq**.

See also **Dq**, **Sq**, and **Qo**.

Re Close an **Rs** block. Does not have any tail arguments.

Rs Begin a bibliographic (“reference”) block. Does not have any head arguments. The block macro may only contain **%A**, **%B**, **%C**, **%D**, **%I**, **%J**, **%N**, **%O**, **%P**, **%Q**, **%R**, **%T**, **%U**, and **%V** child macros (at least one must be specified).

Examples:

```
.Rs
.%A J. E. Hopcroft
.%A J. D. Ullman
.%B Introduction to Automata Theory, Languages, and Computation
.%I Addison-Wesley
.%C Reading, Massachusetts
.%D 1979
.Re
```

If an **Rs** block is used within a SEE ALSO section, a vertical space is asserted before the rendered output, else the block continues on the current line.

Rv `-std` [*function ...*]
Insert a standard sentence regarding a function call’s return value of 0 on success and −1 on error, with the *errno* libc global variable set on error.

If *function* is not specified, the document’s name set by **Nm** is used. Multiple *function* arguments are treated as separate functions.

See also **Ex**.

Sc Close single-quoted context opened by **So**.

Sh *TITLE LINE*
Begin a new section. For a list of conventional manual sections, see “MANUAL STRUCTURE”. These sections should be used unless it’s absolutely necessary that custom sections be used.

Section names should be unique so that they may be keyed by **Sx**. Although this macro is parsed, it should not consist of child node or it may not be linked with **Sx**.

See also **Pp**, **Ss**, and **Sx**.

Sm [on | off]
Switches the spacing mode for output generated from macros.

By default, spacing is on. When switched off, no white space is inserted between macro arguments and between the output generated from adjacent macros, but text lines still get normal spacing between words and sentences.

When called without an argument, the **Sm** macro toggles the spacing mode. Using this is not recommended because it makes the code harder to read.

So *block*
Multi-line version of **Sq**.

Sq *line*
Encloses its arguments in 'typewriter' single-quotes.

See also **Dq**, **Qq**, and **So**.

Ss *Title line*
Begin a new subsection. Unlike with **Sh**, there is no convention for the naming of subsections. Except *DESCRIPTION*, the conventional sections described in "MANUAL STRUCTURE" rarely have subsections.

Sub-section names should be unique so that they may be keyed by **Sx**. Although this macro is parsed, it should not consist of child node or it may not be linked with **Sx**.

See also **Pp**, **Sh**, and **Sx**.

St *-abbreviation*
Replace an abbreviation for a standard with the full form. The following standards are recognised. Where multiple lines are given without a blank line in between, they all refer to the same standard, and using the first form is recommended.

C language standards

-ansiC	ANSI X3.159-1989 ("ANSI C89")
-ansiC-89	ANSI X3.159-1989 ("ANSI C89")
-isoC	ISO/IEC 9899:1990 ("ISO C90")
-isoC-90	ISO/IEC 9899:1990 ("ISO C90") The original C standard.
-isoC-amd1	ISO/IEC 9899/AMD1:1995 ("ISO C90, Amendment 1")
-isoC-tcor1	ISO/IEC 9899/TCOR1:1994 ("ISO C90, Technical Corrigendum 1")
-isoC-tcor2	ISO/IEC 9899/TCOR2:1995 ("ISO C90, Technical Corrigendum 2")
-isoC-99	ISO/IEC 9899:1999 ("ISO C99") The second major version of the C language standard.
-isoC-2011	ISO/IEC 9899:2011 ("ISO C11") The third major version of the C language standard.

POSIX.1 before the Single UNIX Specification

-p1003.1-88	IEEE Std 1003.1-1988 ("POSIX.1")
-p1003.1	IEEE Std 1003.1 ("POSIX.1") The original POSIX standard, based on ANSI C.

−p1003.1-90 ISO/IEC 9945-1:1990 (“POSIX.1”)
−iso9945-1-90 ISO/IEC 9945-1:1990 (“POSIX.1”)
The first update of POSIX.1.

−p1003.1b-93 IEEE Std 1003.1b-1993 (“POSIX.1”)
−p1003.1b IEEE Std 1003.1b (“POSIX.1”)
Real-time extensions.

−p1003.1c-95 IEEE Std 1003.1c-1995 (“POSIX.1”)
POSIX thread interfaces.

−p1003.1i-95 IEEE Std 1003.1i-1995 (“POSIX.1”)
Technical Corrigendum.

−p1003.1-96 ISO/IEC 9945-1:1996 (“POSIX.1”)
−iso9945-1-96 ISO/IEC 9945-1:1996 (“POSIX.1”)
Includes POSIX.1-1990, 1b, 1c, and 1i.

X/Open Portability Guide version 4 and related standards

−xpg3 X/Open Portability Guide Issue 3 (“XPG3”)
An XPG4 precursor, published in 1989.

−p1003.2 IEEE Std 1003.2 (“POSIX.2”)
−p1003.2-92 IEEE Std 1003.2-1992 (“POSIX.2”)
−iso9945-2-93 ISO/IEC 9945-2:1993 (“POSIX.2”)
An XCU4 precursor.

−p1003.2a-92 IEEE Std 1003.2a-1992 (“POSIX.2”)
Updates to POSIX.2.

−xpg4 X/Open Portability Guide Issue 4 (“XPG4”)
Based on POSIX.1 and POSIX.2, published in 1992.

Single UNIX Specification version 1 and related standards

−susv1 Version 1 of the Single UNIX Specification (“SUSv1”)
−xpg4.2 X/Open Portability Guide Issue 4, Version 2 (“XPG4.2”)
This standard was published in 1994. It was used as the basis for UNIX 95 certification. The following three refer to parts of it.

−xsh4.2

−xcurses4.2 X/Open Curses Issue 4, Version 2 (“XCURSES4.2”)

−p1003.1g-2000 IEEE Std 1003.1g-2000 (“POSIX.1”)
Networking APIs, including sockets.

−svid4 System V Interface Definition, Fourth Edition (“SVID4”),
Published in 1995.

Single UNIX Specification version 2 and related standards

−susv2 Version 2 of the Single UNIX Specification (“SUSv2”) This Standard was published in 1997 and is also called X/Open Portability Guide version 5. It was used as the basis for UNIX 98 certification. The following refer to parts of it.

−xbd5 X/Open Base Definitions Issue 5 (“XBD5”)

−xsh5 X/Open System Interfaces and Headers Issue 5 (“XSH5”)

−xcu5 X/Open Commands and Utilities Issue 5 (“XCU5”)

−xns5 X/Open Networking Services Issue 5 (“XNS5”)

−xns5.2 X/Open Networking Services Issue 5.2 (“XNS5.2”)

Single UNIX Specification version 3

−p1003.1-2001 IEEE Std 1003.1-2001 (“POSIX.1”)

−susv3 Version 3 of the Single UNIX Specification (“SUSv3”)
This standard is based on C99, SUSv2, POSIX.1-1996, 1d, and 1j. It is also called X/Open Portability Guide version 6. It is used as the basis for UNIX 03 certification.

−p1003.1-2004 IEEE Std 1003.1-2004 (“POSIX.1”)
The second and last Technical Corrigendum.

Single UNIX Specification version 4

−p1003.1-2008 IEEE Std 1003.1-2008 (“POSIX.1”)

−susv4 Version 4 of the Single UNIX Specification (“SUSv4”)
This standard is also called X/Open Portability Guide version 7.

Other standards

−ieee754 IEEE Std 754-1985
Floating-point arithmetic.

−iso8601 ISO 8601
Representation of dates and times, published in 1988.

−iso8802-3 ISO/IEC 8802-3:1989
Ethernet local area networks.

−ieee1275-94 IEEE Std 1275-1994 (“Open Firmware”)

Sx *Title line*
Reference a section or subsection in the same manual page. The referenced section or subsection name must be identical to the enclosed argument, including whitespace.

Examples:
```
.Sx MANUAL STRUCTURE
```

See also **Sh** and **Ss**.

Sy *word ...*
Request a boldface font.

This is most often used to indicate importance or seriousness (not to be confused with stress emphasis, see **Em**). When none of the semantic macros fit, it is also adequate for syntax elements that have to be given or that appear verbatim.

Examples:
```
.Sy Warning :
If
.Sy s
appears in the owner permissions, set-user-ID mode is set.
This utility replaces the former
.Sy dumpdir
program.
```

See also **Em**, **No**, and **Ql**.

Ta Table cell separator in **Bl** **−column** lists; can only be used below **It**.

Tg [*term*]
Announce that the next input line starts a definition of the *term*. This macro must appear alone on its own input line. The argument defaults to the first argument of the first macro on the next line. The argument may not contain whitespace characters, not even when it is quoted. This macro is a *mandoc*(1)

extension and is typically ignored by other formatters.

When viewing terminal output with *less*(1), the interactive **:t** command can be used to go to the definition of the *term* as described for the *MANPAGER* variable in *man*(1); when producing HTML output, a fragment identifier (**id** attribute) is generated, to be used for deep linking to this place of the document.

In most cases, adding a **Tg** macro would be redundant because *mandoc*(1) is able to automatically tag most definitions. This macro is intended for cases where automatic tagging of a *term* is unsatisfactory, for example if a definition is not tagged automatically (false negative) or if places are tagged that do not define the *term* (false positives). When there is at least one **Tg** macro for a *term*, no other places are automatically marked as definitions of that *term*.

Tn *word ...*
Supported only for compatibility, do not use this in new manuals. Even though the macro name ("tradename") suggests a semantic function, historic usage is inconsistent, mostly using it as a presentation-level macro to request a small caps font.

Ud Supported only for compatibility, do not use this in new manuals. Prints out "currently under development."

Ux Supported only for compatibility, do not use this in new manuals. Prints out "Unix".

Va [*type*] *identifier ...*
A variable name.

Examples:

```
.Va foo
.Va const char *bar;
```

For function arguments and parameters, use **Fa** instead. For declarations of global variables in the *SYNOPSIS* section, use **Vt**.

Vt *type* [*identifier*]
A variable type.

This is also used for indicating global variables in the *SYNOPSIS* section, in which case a variable name is also specified. Note that it accepts "Block partial-implicit" syntax when invoked as the first macro on an input line in the *SYNOPSIS* section, else it accepts ordinary "In-line" syntax. In the former case, this macro starts a new output line, and a blank line is inserted in front if there is a preceding function definition or include directive.

Examples:

```
.Vt unsigned char
.Vt extern const char * const sys_signame[] ;
```

For parameters in function prototypes, use **Fa** instead, for function return types **Ft**, and for variable names outside the *SYNOPSIS* section **Va**, even when including a type with the name. See also "MANUAL STRUCTURE".

Xc Close a scope opened by **Xo**.

Xo *block*
Extend the header of an **It** macro or the body of a partial-implicit block macro beyond the end of the input line. This macro originally existed to work around the 9-argument limit of historic *roff*(7).

Xr *name section*
Link to another manual ("cross-reference").

Cross reference the *name* and *section* number of another man page.

Examples:

```
.Xr mandoc 1
.Xr mandoc 1 ;
.Xr mandoc 1 Ns s behaviour
```

MACRO SYNTAX

The syntax of a macro depends on its classification. In this section, '−arg' refers to macro arguments, which may be followed by zero or more 'parm' parameters; 'Yo' opens the scope of a macro; and if specified, 'Yc' closes it out.

The *Callable* column indicates that the macro may also be called by passing its name as an argument to another macro. For example, '.Op Fl O Ar file' produces '[**−O** *file*]'. To prevent a macro call and render the macro name literally, escape it by prepending a zero-width space, '\&'. For example, 'Op \&Fl O' produces '[Fl O]'. If a macro is not callable but its name appears as an argument to another macro, it is interpreted as opaque text. For example, '.Fl Sh' produces ' **−Sh**'.

The *Parsed* column indicates whether the macro may call other macros by receiving their names as arguments. If a macro is not parsed but the name of another macro appears as an argument, it is interpreted as opaque text.

The *Scope* column, if applicable, describes closure rules.

Block full-explicit

Multi-line scope closed by an explicit closing macro. All macros contains bodies; only **Bf** and (optionally) **Bl** contain a head.

```
.Yo [−arg [parm...]] [head...]
[body...]
.Yc
```

Macro	*Callable*	*Parsed*	*Scope*
Bd	No	No	closed by **Ed**
Bf	No	No	closed by **Ef**
Bk	No	No	closed by **Ek**
Bl	No	No	closed by **El**
Ed	No	No	opened by **Bd**
Ef	No	No	opened by **Bf**
Ek	No	No	opened by **Bk**
El	No	No	opened by **Bl**

Block full-implicit

Multi-line scope closed by end-of-file or implicitly by another macro. All macros have bodies; some (**It** **−bullet**, **−hyphen**, **−dash**, **−enum**, **−item**) don't have heads; only one (**It** in **Bl** **−column**) has multiple heads.

```
.Yo [−arg [parm...]] [head... [Ta head...]]
[body...]
```

Macro	*Callable*	*Parsed*	*Scope*
It	No	Yes	closed by **It**, **El**
Nd	No	No	closed by **Sh**
Nm	No	Yes	closed by **Nm**, **Sh**, **Ss**
Sh	No	Yes	closed by **Sh**
Ss	No	Yes	closed by **Sh**, **Ss**

Note that the **Nm** macro is a "Block full-implicit" macro only when invoked as the first macro in a *SYNOPSIS* section line, else it is "In-line".

Block partial-explicit

Like block full-explicit, but also with single-line scope. Each has at least a body and, in limited circumstances, a head (**Fo**, **Eo**) and/or tail (**Ec**).

```
.Yo [−arg [parm...]] [head...]
[body...]
.Yc [tail...]

.Yo [−arg [parm...]] [head...] [body...] Yc [tail...]
```

Macro	*Callable*	*Parsed*	*Scope*
Ac	Yes	Yes	opened by **Ao**
Ao	Yes	Yes	closed by **Ac**
Bc	Yes	Yes	closed by **Bo**
Bo	Yes	Yes	opened by **Bc**
Brc	Yes	Yes	opened by **Bro**
Bro	Yes	Yes	closed by **Brc**
Dc	Yes	Yes	opened by **Do**
Do	Yes	Yes	closed by **Dc**
Ec	Yes	Yes	opened by **Eo**
Eo	Yes	Yes	closed by **Ec**
Fc	Yes	Yes	opened by **Fo**
Fo	No	No	closed by **Fc**
Oc	Yes	Yes	closed by **Oo**
Oo	Yes	Yes	opened by **Oc**
Pc	Yes	Yes	closed by **Po**
Po	Yes	Yes	opened by **Pc**
Qc	Yes	Yes	opened by **Oo**
Qo	Yes	Yes	closed by **Oc**
Re	No	No	opened by **Rs**
Rs	No	No	closed by **Re**
Sc	Yes	Yes	opened by **So**
So	Yes	Yes	closed by **Sc**
Xc	Yes	Yes	opened by **Xo**
Xo	Yes	Yes	closed by **Xc**

Block partial-implicit

Like block full-implicit, but with single-line scope closed by the end of the line.

```
.Yo [-arg [val...]] [body...] [res...]
```

Macro	*Callable*	*Parsed*
Aq	Yes	Yes
Bq	Yes	Yes
Brq	Yes	Yes
D1	No	Yes
Dl	No	Yes
Dq	Yes	Yes
En	Yes	Yes
Op	Yes	Yes
Pq	Yes	Yes
Ql	Yes	Yes
Qq	Yes	Yes
Sq	Yes	Yes
Vt	Yes	Yes

Note that the **Vt** macro is a “Block partial-implicit” only when invoked as the first macro in a *SYNOPSIS* section line, else it is “In-line”.

Special block macro

The **Ta** macro can only be used below **It** in **Bl** `-column` lists. It delimits blocks representing table cells; these blocks have bodies, but no heads.

Macro	*Callable*	*Parsed*	*Scope*
Ta	Yes	Yes	closed by **Ta**, **It**

In-line

Closed by the end of the line, fixed argument lengths, and/or subsequent macros. In-line macros have only text children. If a number (or inequality) of arguments is (n), then the macro accepts an arbitrary number of arguments.

```
.Yo [-arg [val...]] [args...] [res...]

.Yo [-arg [val...]] [args...] Yc...

.Yo [-arg [val...]] arg0 arg1 argN
```

Macro	*Callable*	*Parsed*	*Arguments*
%A	No	No	>0
%B	No	No	>0
%C	No	No	>0
%D	No	No	>0
%I	No	No	>0
%J	No	No	>0
%N	No	No	>0
%O	No	No	>0
%P	No	No	>0
%Q	No	No	>0
%R	No	No	>0
%T	No	No	>0
%U	No	No	>0
%V	No	No	>0
Ad	Yes	Yes	>0
An	Yes	Yes	>0
Ap	Yes	Yes	0
Ar	Yes	Yes	n
At	Yes	Yes	1
Bsx	Yes	Yes	n
Bt	No	No	0
Bx	Yes	Yes	n
Cd	Yes	Yes	>0
Cm	Yes	Yes	>0
Db	No	No	1
Dd	No	No	n
Dt	No	No	n
Dv	Yes	Yes	>0
Dx	Yes	Yes	n
Em	Yes	Yes	>0
Er	Yes	Yes	>0
Es	Yes	Yes	2
Ev	Yes	Yes	>0
Ex	No	No	n
Fa	Yes	Yes	>0
Fd	No	No	>0
Fl	Yes	Yes	n
Fn	Yes	Yes	>0
Fr	Yes	Yes	>0
Ft	Yes	Yes	>0

Fx	Yes	Yes	n
Hf	No	No	n
Ic	Yes	Yes	>0
In	No	No	1
Lb	No	No	1
Li	Yes	Yes	>0
Lk	Yes	Yes	>0
Lp	No	No	0
Ms	Yes	Yes	>0
Mt	Yes	Yes	>0
Nm	Yes	Yes	n
No	Yes	Yes	>0
Ns	Yes	Yes	0
Nx	Yes	Yes	n
Os	No	No	n
Ot	Yes	Yes	>0
Ox	Yes	Yes	n
Pa	Yes	Yes	n
Pf	Yes	Yes	1
Pp	No	No	0
Rv	No	No	n
Sm	No	No	<2
St	No	Yes	1
Sx	Yes	Yes	>0
Sy	Yes	Yes	>0
Tg	No	No	<2
Tn	Yes	Yes	>0
Ud	No	No	0
Ux	Yes	Yes	n
Va	Yes	Yes	n
Vt	Yes	Yes	>0
Xr	Yes	Yes	2

Delimiters

When a macro argument consists of one single input character considered as a delimiter, the argument gets special handling. This does not apply when delimiters appear in arguments containing more than one character. Consequently, to prevent special handling and just handle it like any other argument, a delimiter can be escaped by prepending a zero-width space (‘\&’). In text lines, delimiters never need escaping, but may be used as normal punctuation.

For many macros, when the leading arguments are opening delimiters, these delimiters are put before the macro scope, and when the trailing arguments are closing delimiters, these delimiters are put after the macro scope. Spacing is suppressed after opening delimiters and before closing delimiters. For example,

```
.Aq ( [ word ] ) .
```

renders as:

(⟨word⟩).

Opening delimiters are:

(left parenthesis
[left bracket

Closing delimiters are:

. period
, comma
: colon
; semicolon
) right parenthesis
] right bracket
? question mark
! exclamation mark

Note that even a period preceded by a backslash (‘\.’) gets this special handling; use ‘\&.’ to prevent that.

Many in-line macros interrupt their scope when they encounter delimiters, and resume their scope when more arguments follow that are not delimiters. For example,

```
.Fl a ( b | c \*(Ba d ) e
```

renders as:

```
-a (-b | -c | -d) -e
```

This applies to both opening and closing delimiters, and also to the middle delimiter, which does not suppress spacing:

| vertical bar

As a special case, the predefined string *(Ba is handled and rendered in the same way as a plain ‘|’ character. Using this predefined string is not recommended in new manuals.

Appending a zero-width space (‘\&’) to the end of an input line is also useful to prevent the interpretation of a trailing period, exclamation or question mark as the end of a sentence, for example when an abbreviation happens to occur at the end of a text or macro input line.

Font handling

In **mdoc** documents, usage of semantic markup is recommended in order to have proper fonts automatically selected; only when no fitting semantic markup is available, consider falling back to “Physical markup” macros. Whenever any **mdoc** macro switches the *roff*(7) font mode, it will automatically restore the previous font when exiting its scope. Manually switching the font using the *roff*(7) ‘`\f`’ font escape sequences is never required.

COMPATIBILITY

This section provides an incomplete list of compatibility issues between mandoc and GNU troff ("groff").

The following problematic behaviour is found in groff:

- **Pa** does not format its arguments when used in the FILES section under certain list types.
- **Ta** can only be called by other macros, but not at the beginning of a line.
- ‘\f’ (font face) and ‘\F’ (font family face) “Text Decoration” escapes behave irregularly when specified within line-macro scopes.
- Negative scaling units return to prior lines. Instead, mandoc truncates them to zero.

The following features are unimplemented in mandoc:

- **Bd** `-file` *file* is unsupported for security reasons.
- **Bd** `-filled` does not adjust the right margin, but is an alias for **Bd** `-ragged`.
- **Bd** `-literal` does not use a literal font, but is an alias for **Bd** `-unfilled`.
- **Bd** `-offset center` and `-offset right` don’t work. Groff does not implement centered and flush-right rendering either, but produces large indentations.

SEE ALSO

man(1), *mandoc*(1), *eqn*(7), *man*(7), *mandoc_char*(7), *roff*(7), *tbl*(7)

The web page extended documentation for the mdoc language provides a few tutorial-style pages for beginners, an extensive style guide for advanced authors, and an alphabetic index helping to choose the best macros for various kinds of content.

The manual page groff_mdoc(7) contained in the “groff” package documents exactly the same language in a somewhat different style.

HISTORY

The **mdoc** language first appeared as a troff macro package in 4.4BSD. It was later significantly updated by Werner Lemberg and Ruslan Ermilov in groff-1.17. The standalone implementation that is part of the *mandoc*(1) utility written by Kristaps Dzonsons appeared in OpenBSD 4.6.

AUTHORS

The **mdoc** reference was written by Kristaps Dzonsons <kristaps@bsd.lv>.

NAME

roff — roff language reference for mandoc

DESCRIPTION

The **roff** language is a general purpose text formatting language. Since traditional implementations of the *mdoc*(7) and *man*(7) manual formatting languages are based on it, many real-world manuals use small numbers of **roff** requests and escape sequences intermixed with their *mdoc*(7) or *man*(7) code. To properly format such manuals, the *mandoc*(1) utility supports a subset of **roff** requests and escapes. Even though this manual page lists all **roff** requests and escape sequences, it only contains partial information about requests not supported by *mandoc*(1) and about language features that do not matter for manual pages. For complete **roff** manuals, consult the "SEE ALSO" section.

Input lines beginning with the control character '.' are parsed for requests and macros. Such lines are called "request lines" or "macro lines", respectively. Requests change the processing state and manipulate the formatting; some macros also define the document structure and produce formatted output. The single quote ("'") is accepted as an alternative control character, treated by *mandoc*(1) just like '`.`'

Lines not beginning with control characters are called "text lines". They provide free-form text to be printed; the formatting of the text depends on the respective processing context.

LANGUAGE SYNTAX

roff documents may contain only graphable 7-bit ASCII characters, the space character, and, in certain circumstances, the tab character. The backslash character '\' indicates the start of an escape sequence, used for example for "Comments" and "Special Characters". For a complete listing of escape sequences, consult the "ESCAPE SEQUENCE REFERENCE" below.

Comments

Text following an escaped double-quote '\"', whether in a request, macro, or text line, is ignored to the end of the line. A request line beginning with a control character and comment escape '.\"' is also ignored. Furthermore, request lines with only a control character and optional trailing whitespace are stripped from input.

Examples:

```
.\" This is a comment line.
.\" The next line is ignored:
.
.Sh EXAMPLES \" This is a comment, too.
example text \" And so is this.
```

Special Characters

Special characters are used to encode special glyphs and are rendered differently across output media. They may occur in request, macro, and text lines. Sequences begin with the escape character '\' followed by either an open-parenthesis '(' for two-character sequences; an open-bracket '[' for n-character sequences (terminated at a close-bracket ']'); or a single one character sequence.

Examples:

`\(em` Two-letter em dash escape.
`\e` One-letter backslash escape.

See *mandoc_char*(7) for a complete list.

Font Selection

In *mdoc*(7) and *man*(7) documents, fonts are usually selected with macros. The **\f** escape sequence and the **ft** request can be used to manually change the font, but this is not recommended in *mdoc*(7) documents. Such manual font changes are overridden by many subsequent macros.

The following fonts are supported:

`B` Bold font.
`BI` A font that is both bold and italic.

`CB` Bold constant width font. Same as `B` in terminal output.
`CI` Italic constant width font. Same as `I` in terminal output.
`CR` Regular constant width font. Same as `R` in terminal output.
`CW` An alias for `CR`.
`I` Italic font.
`P` Return to the previous font. If a macro caused a font change since the last **`\f`** eascape sequence or **`ft`** request, this returns to the font before the last font change in the macro rather than to the font before the last manual font change.
`R` Roman font. This is the default font.
`1` An alias for `R`.
`2` An alias for `I`.
`3` An alias for `B`.
`4` An alias for `BI`.

Examples:

`\fBbold\fR`
Write in **bold**, then switch to regular font mode.

`\fIitalic\fP`
Write in *italic*, then return to previous font mode.

`\f(BIbold italic\fP`
Write in ***bold italic***, then return to previous font mode.

Whitespace

Whitespace consists of the space character. In text lines, whitespace is preserved within a line. In request and macro lines, whitespace delimits arguments and is discarded.

Unescaped trailing spaces are stripped from text line input unless in a literal context. In general, trailing whitespace on any input line is discouraged for reasons of portability. In the rare case that a space character is needed at the end of an input line, it may be forced by '\ \&'.

Literal space characters can be produced in the output using escape sequences. In macro lines, they can also be included in arguments using quotation; see "MACRO SYNTAX" for details.

Blank text lines, which may include whitespace, are only permitted within literal contexts. If the first character of a text line is a space, that line is printed with a leading newline.

Scaling Widths

Many requests and macros support scaled widths for their arguments. The syntax for a scaled width is '`[+-]?[0-9]*.[0-9]*[:unit:]`', where a decimal must be preceded or followed by at least one digit.

The following scaling units are accepted:

c	centimetre
i	inch
P	pica (1/6 inch)
p	point (1/72 inch)
f	scale 'u' by 65536
v	default vertical span
m	width of rendered 'm' (em) character
n	width of rendered 'n' (en) character
u	default horizontal span for the terminal
M	mini-em (1/100 em)

Using anything other than 'm', 'n', or 'v' is necessarily non-portable across output media. See "COMPATIBILITY".

If a scaling unit is not provided, the numerical value is interpreted under the default rules of 'v' for vertical spaces and 'u' for horizontal ones.

Examples:

`.Bl -tag -width 2i`
two-inch tagged list indentation in *mdoc*(7)

`.HP 2i`
two-inch tagged list indentation in *man*(7)

`.sp 2v`
two vertical spaces

Sentence Spacing

Each sentence should terminate at the end of an input line. By doing this, a formatter will be able to apply the proper amount of spacing after the end of sentence (unescaped) period, exclamation mark, or question mark followed by zero or more non-sentence closing delimiters (‘)’, ‘]’, ‘’’, ‘"’). The proper spacing is also intelligently preserved if a sentence ends at the boundary of a macro line.

If an input line happens to end with a period, exclamation or question mark that isn’t the end of a sentence, append a zero-width space (‘\&’).

Examples:

```
Do not end sentences mid-line like this.  Instead,
end a sentence like this.
A macro would end like this:
.Xr mandoc 1 .
An abbreviation at the end of an input line needs escaping, e.g.\&
like this.
```

REQUEST SYNTAX

A request or macro line consists of:

1. the control character ‘.’ or ‘’’ at the beginning of the line,
2. optionally an arbitrary amount of whitespace,
3. the name of the request or the macro, which is one word of arbitrary length, terminated by whitespace,
4. and zero or more arguments delimited by whitespace.

Thus, the following request lines are all equivalent:

```
.ig end
.ig    end
.   ig end
```

MACRO SYNTAX

Macros are provided by the *mdoc*(7) and *man*(7) languages and can be defined by the **de** request. When called, they follow the same syntax as requests, except that macro arguments may optionally be quoted by enclosing them in double quote characters (‘"’). Quoted text, even if it contains whitespace or would cause a macro invocation when unquoted, is always considered literal text. Inside quoted text, pairs of double quote characters (‘""’) resolve to single double quote characters.

To be recognised as the beginning of a quoted argument, the opening quote character must be preceded by a space character. A quoted argument extends to the next double quote character that is not part of a pair, or to the end of the input line, whichever comes earlier. Leaving out the terminating double quote character at the end of the line is discouraged. For clarity, if more arguments follow on the same input line, it is recommended to follow the terminating double quote character by a space character; in case the next character after the terminating double quote character is anything else, it is regarded as the beginning of the next, unquoted argument.

Both in quoted and unquoted arguments, pairs of backslashes (‘\\’) resolve to single backslashes. In unquoted arguments, space characters can alternatively be included by preceding them with a backslash (‘\ ’), but quoting is usually better for clarity.

Examples:

`.Fn strlen "const char *s"`
: Group arguments "const char *s" into one function argument. If unspecified, "const", "char", and "*s" would be considered separate arguments.

`.Op "Fl a"`
: Consider "Fl a" as literal text instead of a flag macro.

REQUEST REFERENCE

The *mandoc*(1) **roff** parser recognises the following requests. For requests marked as "ignored" or "unsupported", any arguments are ignored, and the number of arguments is not checked.

ab [*message*]
: Abort processing. Currently unsupported.

ad [b | c | l | n | r]
: Set line adjustment mode for subsequent text. Currently ignored.

af *registername format*
: Assign an output format to a number register. Currently ignored.

aln *newname oldname*
: Create an alias for a number register. Currently unsupported.

als *newname oldname*
: Create an alias for a request, string, macro, or diversion.

am *macroname* [*endmacro*]
: Append to a macro definition. The syntax of this request is the same as that of **de**.

am1 *macroname* [*endmacro*]
: Append to a macro definition, switching roff compatibility mode off during macro execution (groff extension). The syntax of this request is the same as that of **de1**. Since *mandoc*(1) does not implement **roff** compatibility mode at all, it handles this request as an alias for **am**.

ami *macrostring* [*endstring*]
: Append to a macro definition, specifying the macro name indirectly (groff extension). The syntax of this request is the same as that of **dei**.

ami1 *macrostring* [*endstring*]
: Append to a macro definition, specifying the macro name indirectly and switching roff compatibility mode off during macro execution (groff extension). The syntax of this request is the same as that of **dei1**. Since *mandoc*(1) does not implement **roff** compatibility mode at all, it handles this request as an alias for **ami**.

as *stringname* [*string*]
: Append to a user-defined string. The syntax of this request is the same as that of **ds**. If a user-defined string with the specified name does not yet exist, it is set to the empty string before appending.

as1 *stringname* [*string*]
: Append to a user-defined string, switching roff compatibility mode off during macro execution (groff extension). The syntax of this request is the same as that of **ds1**. Since *mandoc*(1) does not implement **roff** compatibility mode at all, it handles this request as an alias for **as**.

asciify *divname*
: Fully unformat a diversion. Currently unsupported.

backtrace
: Print a backtrace of the input stack. This is a groff extension and currently ignored.

bd *font* [*curfont*] [*offset*]
Artificially embolden by repeated printing with small shifts. Currently ignored.

bleedat *left top width height*
Set the BleedBox page parameter for PDF generation. This is a Heirloom extension and currently ignored.

blm *macroname*
Set a blank line trap. Currently unsupported.

box *divname*
Begin a diversion without including a partially filled line. Currently unsupported.

boxa *divname*
Add to a diversion without including a partially filled line. Currently unsupported.

bp [+|-]*pagenumber*
Begin a new page. Currently ignored.

BP *source height width position offset flags label*
Define a frame and place a picture in it. This is a Heirloom extension and currently unsupported.

br Break the output line.

break Break out of the innermost **while** loop.

breakchar *char ...*
Optional line break characters. This is a Heirloom extension and currently ignored.

brnl *N*
Break output line after the next *N* input lines. This is a Heirloom extension and currently ignored.

brp Break and spread output line. Currently, this is implemented as an alias for **br**.

brpnl *N*
Break and spread output line after the next *N* input lines. This is a Heirloom extension and currently ignored.

c2 [*char*]
Change the no-break control character. Currently unsupported.

cc [*char*]
Change the control character. If *char* is not specified, the control character is reset to '.'. Trailing characters are ignored.

ce [*N*]
Center the next *N* input lines without filling. *N* defaults to 1. An argument of 0 or less ends centering. Currently, high level macros abort centering.

cf *filename*
Output the contents of a file. Ignored because insecure.

cflags *flags char ...*
Set character flags. This is a groff extension and currently ignored.

ch *macroname* [*dist*]
Change a trap location. Currently ignored.

char *glyph* [*string*]
Define or redefine the ASCII character or character escape sequence *glyph* to be rendered as *string*, which can be empty. Only partially supported in *mandoc*(1); may interact incorrectly with **tr**.

chop *stringname*
Remove the last character from a macro, string, or diversion. Currently unsupported.

class *classname char ...*
Define a character class. This is a groff extension and currently ignored.

close *streamname*
Close an open file. Ignored because insecure.

CL *color text*
Print text in color. This is a Heirloom extension and currently unsupported.

color [1 | 0]
Activate or deactivate colors. This is a groff extension and currently ignored.

composite *from to*
Define a name component for composite glyph names. This is a groff extension and currently unsupported.

continue
Immediately start the next iteration of a **while** loop. Currently unsupported.

cp [1 | 0]
Switch **roff** compatibility mode on or off. Currently ignored.

cropat *left top width height*
Set the CropBox page parameter for PDF generation. This is a Heirloom extension and currently ignored.

cs *font* [*width* [*emsize*]]
Constant character spacing mode. Currently ignored.

cu [*N*]
Underline next *N* input lines including whitespace. Currently ignored.

da *divname*
Append to a diversion. Currently unsupported.

dch *macroname* [*dist*]
Change a trap location in the current diversion. This is a Heirloom extension and currently unsupported.

de *macroname* [*endmacro*]
Define a **roff** macro. Its syntax can be either

```
.de macroname
definition
..
```

or

```
.de macroname endmacro
definition
.endmacro
```

Both forms define or redefine the macro *macroname* to represent the *definition*, which may consist of one or more input lines, including the newline characters terminating each line, optionally containing calls to **roff** requests, **roff** macros or high-level macros like *man*(7) or *mdoc*(7) macros, whichever applies to the document in question.

Specifying a custom *endmacro* works in the same way as for **ig**; namely, the call to '.*endmacro*' first ends the *definition*, and after that, it is also evaluated as a **roff** request or **roff** macro, but not as a high-level macro.

The macro can be invoked later using the syntax

.*macroname* [*argument* [*argument* ...]]

Regarding argument parsing, see "MACRO SYNTAX" above.

The line invoking the macro will be replaced in the input stream by the *definition*, replacing all occurrences of \\$*N*, where *N* is a digit, by the *N*th *argument*. For example,

```
.de ZN
\fI\^\\$1\^\fP\\$2
..
.ZN XtFree .
```

produces

\fI\^XtFree\^\fP.

in the input stream, and thus in the output: *XtFree*. Each occurrence of \\$∗ is replaced with all the arguments, joined together with single space characters. The variant \\$@ is similar, except that each argument is individually quoted.

Since macros and user-defined strings share a common string table, defining a macro *macroname* clobbers the user-defined string *macroname*, and the *definition* can also be printed using the '\∗' string interpolation syntax described below **ds**, but this is rarely useful because every macro definition contains at least one explicit newline character.

In order to prevent endless recursion, both groff and *mandoc*(1) limit the stack depth for expanding macros and strings to a large, but finite number, and *mandoc*(1) also limits the length of the expanded input line. Do not rely on the exact values of these limits.

de1 *macroname* [*endmacro*]
Define a **roff** macro that will be executed with **roff** compatibility mode switched off during macro execution. This is a groff extension. Since *mandoc*(1) does not implement **roff** compatibility mode at all, it handles this request as an alias for **de**.

defcolor *newname scheme component ...*
Define a color name. This is a groff extension and currently ignored.

dei *macrostring* [*endstring*]
Define a **roff** macro, specifying the macro name indirectly (groff extension). The syntax of this request is the same as that of **de**. The effect is the same as:

.de \∗[*macrostring*] [\∗[*endstring*]]

dei1 *macrostring* [*endstring*]
Define a **roff** macro that will be executed with **roff** compatibility mode switched off during macro execution, specifying the macro name indirectly (groff extension). Since *mandoc*(1) does not implement **roff** compatibility mode at all, it handles this request as an alias for **dei**.

device *string ...*

devicem *stringname*
These two requests only make sense with the groff-specific intermediate output format and are unsupported.

di *divname*
Begin a diversion. Currently unsupported.

do *command* [*argument ...*]
Execute **roff** request or macro line with compatibility mode disabled. Currently unsupported.

ds *stringname* [["]*string*]
: Define a user-defined string. The *stringname* and *string* arguments are space-separated. If the *string* begins with a double-quote character, that character will not be part of the string. All remaining characters on the input line form the *string*, including whitespace and double-quote characters, even trailing ones.

  The *string* can be interpolated into subsequent text by using *[*stringname*] for a *stringname* of arbitrary length, or *(NN or *N if the length of *stringname* is two or one characters, respectively. Interpolation can be prevented by escaping the leading backslash; that is, an asterisk preceded by an even number of backslashes does not trigger string interpolation.

  Since user-defined strings and macros share a common string table, defining a string *stringname* clobbers the macro *stringname*, and the *stringname* used for defining a string can also be invoked as a macro, in which case the following input line will be appended to the *string*, forming a new input line passed to the **roff** parser. For example,

```
.ds badidea .S
.badidea
H SYNOPSIS
```

  invokes the **SH** macro when used in a *man*(7) document. Such abuse is of course strongly discouraged.

ds1 *stringname* [["]*string*]
: Define a user-defined string that will be expanded with **roff** compatibility mode switched off during string expansion. This is a groff extension. Since *mandoc*(1) does not implement **roff** compatibility mode at all, it handles this request as an alias for **ds**.

dwh *dist macroname*
: Set a location trap in the current diversion. This is a Heirloom extension and currently unsupported.

dt [*dist macroname*]
: Set a trap within a diversion. Currently unsupported.

ec [*char*]
: Enable the escape mechanism and change the escape character. The *char* argument defaults to the backslash (‘\’).

ecr
: Restore the escape character. Currently unsupported.

ecs
: Save the escape character. Currently unsupported.

el *body*
: The “else” half of an if/else conditional. Pops a result off the stack of conditional evaluations pushed by **ie** and uses it as its conditional. If no stack entries are present (e.g., due to no prior **ie** calls) then false is assumed. The syntax of this request is similar to **if** except that the conditional is missing.

em *macroname*
: Set a trap at the end of input. Currently unsupported.

EN
: End an equation block. See **EQ**.

eo
: Disable the escape mechanism completely.

EP
: End a picture started by **BP**. This is a Heirloom extension and currently unsupported.

EQ
: Begin an equation block. See *eqn*(7) for a description of the equation language.

errprint *message*
: Print a string like an error message. This is a Heirloom extension and currently ignored.

ev [*envname*]
Switch to another environment. Currently unsupported.

evc [*envname*]
Copy an environment into the current environment. Currently unsupported.

ex Abort processing and exit. Currently unsupported.

fallback *curfont font ...*
Select the fallback sequence for a font. This is a Heirloom extension and currently ignored.

fam [*familyname*]
Change the font family. This is a groff extension and currently ignored.

fc [*delimchar* [*padchar*]]
Define a delimiting and a padding character for fields. Currently unsupported.

fchar *glyphname* [*string*]
Define a fallback glyph. Currently unsupported.

fcolor *colorname*
Set the fill color for \D objects. This is a groff extension and currently ignored.

fdeferlig *font string ...*
Defer ligature building. This is a Heirloom extension and currently ignored.

feature +|-*name*
Enable or disable an OpenType feature. This is a Heirloom extension and currently ignored.

fi Break the output line and switch to fill mode, which is active by default but can be ended with the **nf** request. In fill mode, input from subsequent input lines is added to the same output line until the next word no longer fits, at which point the output line is broken. This request is implied by the *mdoc*(7) **Sh** macro and by the *man*(7) **SH**, **SS**, and **EE** macros.

fkern *font minkern*
Control the use of kerning tables for a font. This is a Heirloom extension and currently ignored.

fl Flush output. Currently ignored.

flig *font string char ...*
Define ligatures. This is a Heirloom extension and currently ignored.

fp *position font* [*filename*]
Assign font position. Currently ignored.

fps *mapname ...*
Mount a font with a special character map. This is a Heirloom extension and currently ignored.

fschar *font glyphname* [*string*]
Define a font-specific fallback glyph. This is a groff extension and currently unsupported.

fspacewidth *font* [*afmunits*]
Set a font-specific width for the space character. This is a Heirloom extension and currently ignored.

fspecial *curfont* [*font ...*]
Conditionally define a special font. This is a groff extension and currently ignored.

ft [*font*]
Change the font; see “Font Selection”. The *font* argument defaults to P.

ftr *newname* [*oldname*]
Translate font name. This is a groff extension and currently ignored.

fzoom *font* [*permille*]
Zoom font size. Currently ignored.

gcolor [*colorname*]
Set glyph color. This is a groff extension and currently ignored.

hc [*char*]
Set the hyphenation character. Currently ignored.

hcode *char code ...*
Set hyphenation codes of characters. Currently ignored.

hidechar *font char ...*
Hide characters in a font. This is a Heirloom extension and currently ignored.

hla *language*
Set hyphenation language. This is a groff extension and currently ignored.

hlm [*number*]
Set maximum number of consecutive hyphenated lines. Currently ignored.

hpf *filename*
Load hyphenation pattern file. This is a groff extension and currently ignored.

hpfa *filename*
Load hyphenation pattern file, appending to the current patterns. This is a groff extension and currently ignored.

hpfcode *code code ...*
Define mapping values for character codes in hyphenation patterns. This is a groff extension and currently ignored.

hw *word ...*
Specify hyphenation points in words. Currently ignored.

hy [*mode*]
Set automatic hyphenation mode. Currently ignored.

hylang *language*
Set hyphenation language. This is a Heirloom extension and currently ignored.

hylen *nchar*
Minimum word length for hyphenation. This is a Heirloom extension and currently ignored.

hym [*length*]
Set hyphenation margin. This is a groff extension and currently ignored.

hypp *penalty ...*
Define hyphenation penalties. This is a Heirloom extension and currently ignored.

hys [*length*]
Set hyphenation space. This is a groff extension and currently ignored.

ie *condition body*
The “if” half of an if/else conditional. The result of the conditional is pushed into a stack used by subsequent invocations of **el**, which may be separated by any intervening input (or not exist at all). Its syntax is equivalent to **if**.

if *condition body*
Begin a conditional. This request can also be written as follows:

```
.if condition \{body
body ...\}
```

```
.if condition \{\
body ...
.\}
```

The *condition* is a boolean expression. Currently, *mandoc*(1) supports the following subset of roff conditionals:

- If '!' is prefixed to *condition*, it is logically inverted.
- If the first character of *condition* is 'n' (nroff mode) or 'o' (odd page), it evaluates to true, and the *body* starts with the next character.
- If the first character of *condition* is 'e' (even page), 't' (troff mode), or 'v' (vroff mode), it evaluates to false, and the *body* starts with the next character.
- If the first character of *condition* is 'c' (character available), it evaluates to true if the following character is an ASCII character or a valid character escape sequence, or to false otherwise. The *body* starts with the character following that next character.
- If the first character of *condition* is 'd', it evaluates to true if the rest of *condition* is the name of an existing user defined macro or string; otherwise, it evaluates to false.
- If the first character of *condition* is 'r', it evaluates to true if the rest of *condition* is the name of an existing number register; otherwise, it evaluates to false.
- If the *condition* starts with a parenthesis or with an optionally signed integer number, it is evaluated according to the rules of "Numerical expressions" explained below. It evaluates to true if the result is positive, or to false if the result is zero or negative.
- Otherwise, the first character of *condition* is regarded as a delimiter and it evaluates to true if the string extending from its first to its second occurrence is equal to the string extending from its second to its third occurrence.
- If *condition* cannot be parsed, it evaluates to false.

If a conditional is false, its children are not processed, but are syntactically interpreted to preserve the integrity of the input document. Thus,

```
.if t .ig
```

will discard the '.ig', which may lead to interesting results, but

```
.if t .if t \{\
```

will continue to syntactically interpret to the block close of the final conditional. Sub-conditionals, in this case, obviously inherit the truth value of the parent.

If the *body* section is begun by an escaped brace '\{', scope continues until the end of the input line containing the matching closing-brace escape sequence '\}'. If the *body* is not enclosed in braces, scope continues until the end of the line. If the *condition* is followed by a *body* on the same line, whether after a brace or not, then requests and macros *must* begin with a control character. It is generally more intuitive, in this case, to write

```
.if condition \{\
.request
.\}
```

than having the request or macro follow as

```
.if condition \{.request
```

The scope of a conditional is always parsed, but only executed if the conditional evaluates to true.

Note that the '\}' is converted into a zero-width escape sequence if not passed as a standalone macro '.\}'. For example,

.Fl a \} b

will result in '\}' being considered an argument of the 'Fl' macro.

ig [*endmacro*]
Ignore input. Its syntax can be either

```
.ig
ignored text
..
```

or

```
.ig endmacro
ignored text
.endmacro
```

In the first case, input is ignored until a '..' request is encountered on its own line. In the second case, input is ignored until the specified '.*endmacro*' is encountered. Do not use the escape character '\' anywhere in the definition of *endmacro*; it would cause very strange behaviour.

When the *endmacro* is a roff request or a roff macro, like in

.ig if

the subsequent invocation of **if** will first terminate the *ignored text*, then be invoked as usual. Otherwise, it only terminates the *ignored text*, and arguments following it or the '..' request are discarded.

in [[+|-]*width*]
Change indentation. See *man*(7). Ignored in *mdoc*(7).

index *register stringname substring*
Find a substring in a string. This is a Heirloom extension and currently unsupported.

it *expression macro*
Set an input line trap. The named *macro* will be invoked after processing the number of input text lines specified by the numerical *expression*. While evaluating the *expression*, the unit suffixes described below "Scaling Widths" are ignored.

itc *expression macro*
Set an input line trap, not counting lines ending with \c. Currently unsupported.

IX *class keystring*
To support the generation of a table of contents, *pod2man*(1) emits this user-defined macro, usually without defining it. To avoid reporting large numbers of spurious errors, *mandoc*(1) ignores it.

kern [1 | 0]
Switch kerning on or off. Currently ignored.

kernafter *font char ... afmunits ...*
Increase kerning after some characters. This is a Heirloom extension and currently ignored.

kernbefore *font char ... afmunits ...*
Increase kerning before some characters. This is a Heirloom extension and currently ignored.

kernpair *font char ... font char ... afmunits*
Add a kerning pair to the kerning table. This is a Heirloom extension and currently ignored.

lc [*glyph*]
Define a leader repetition character. Currently unsupported.

lc_ctype *localename*
Set the LC_CTYPE locale. This is a Heirloom extension and currently unsupported.

lds *macroname string*
Define a local string. This is a Heirloom extension and currently unsupported.

length *register string*
Count the number of input characters in a string. Currently unsupported.

letadj *lspmin lshmin letss lspmax lshmax*
Dynamic letter spacing and reshaping. This is a Heirloom extension and currently ignored.

lf *lineno* [*filename*]
Change the line number for error messages. Ignored because insecure.

lg [1 | 0]
Switch the ligature mechanism on or off. Currently ignored.

lhang *font char ... afmunits*
Hang characters at left margin. This is a Heirloom extension and currently ignored.

linetabs [1 | 0]
Enable or disable line-tabs mode. This is a groff extension and currently unsupported.

ll [[+|-]*width*]
Change the output line length. If the *width* argument is omitted, the line length is reset to its previous value. The default setting for terminal output is 78n. If a sign is given, the line length is added to or subtracted from; otherwise, it is set to the provided value. Using this request in new manuals is discouraged for several reasons, among others because it overrides the *mandoc*(1) `-O width` command line option.

lnr *register* [+|-]*value* [*increment*]
Set local number register. This is a Heirloom extension and currently unsupported.

lnrf *register* [+|-]*value* [*increment*]
Set local floating-point register. This is a Heirloom extension and currently unsupported.

lpfx *string*
Set a line prefix. This is a Heirloom extension and currently unsupported.

ls [*factor*]
Set line spacing. It takes one integer argument specifying the vertical distance of subsequent output text lines measured in v units. Currently ignored.

lsm *macroname*
Set a leading spaces trap. This is a groff extension and currently unsupported.

lt [[+|-]*width*]
Set title line length. Currently ignored.

mc *glyph* [*dist*]
Print margin character in the right margin. The *dist* is currently ignored; instead, 1n is used.

mediasize *media*
Set the device media size. This is a Heirloom extension and currently ignored.

minss *width*
Set minimum word space. This is a Heirloom extension and currently ignored.

mk [*register*]
Mark vertical position. Currently ignored.

mso *filename*
Load a macro file using the search path. Ignored because insecure.

na Disable adjusting without changing the adjustment mode. Currently ignored.

ne [*height*]
Declare the need for the specified minimum vertical space before the next trap or the bottom of the page. Currently ignored.

nf Break the output line and switch to no-fill mode. Subsequent input lines are kept together on the same output line even when exceeding the right margin, and line breaks in subsequent input cause output line breaks. This request is implied by the *mdoc*(7) **Bd** `-unfilled` and **Bd** `-literal` macros and by the *man*(7) **EX** macro. The **fi** request switches back to the default fill mode.

nh Turn off automatic hyphenation mode. Currently ignored.

nhychar *char ...*
Define hyphenation-inhibiting characters. This is a Heirloom extension and currently ignored.

nm [*start* [*inc* [*space* [*indent*]]]]
Print line numbers. Currently unsupported.

nn [*number*]
Temporarily turn off line numbering. Currently unsupported.

nop *body*
Execute the rest of the input line as a request, macro, or text line, skipping the **nop** request and any space characters immediately following it. This is mostly used to indent text lines inside macro definitions.

nr *register* [+|-]*expression* [*stepsize*]
Define or change a register. A register is an arbitrary string value that defines some sort of state, which influences parsing and/or formatting. For the syntax of *expression*, see "Numerical expressions" below. If it is prefixed by a sign, the register will be incremented or decremented instead of assigned to.

The *stepsize* is used by the **\n+** auto-increment feature. It remains unchanged when omitted while changing an existing register, and it defaults to 0 when defining a new register.

The following *register* is handled specially:

nS If set to a positive integer value, certain *mdoc*(7) macros will behave in the same way as in the *SYNOPSIS* section. If set to 0, these macros will behave in the same way as outside the *SYNOPSIS* section, even when called within the *SYNOPSIS* section itself. Note that starting a new *mdoc*(7) section with the **Sh** macro will reset this register.

nrf *register* [+|-]*expression* [*increment*]
Define or change a floating-point register. This is a Heirloom extension and currently unsupported.

nroff Force nroff mode. This is a groff extension and currently ignored.

ns Turn on no-space mode. Currently ignored.

nx [*filename*]
Abort processing of the current input file and process another one. Ignored because insecure.

open *stream file*
Open a file for writing. Ignored because insecure.

opena *stream file*
Open a file for appending. Ignored because insecure.

os Output saved vertical space. Currently ignored.

output *string*
Output directly to intermediate output. Not supported.

padj [1 | 0]
Globally control paragraph-at-once adjustment. This is a Heirloom extension and currently ignored.

papersize *media*
Set the paper size. This is a Heirloom extension and currently ignored.

pc [*char*]
Change the page number character. Currently ignored.

pev Print environments. This is a groff extension and currently ignored.

pi *command*
Pipe output to a shell command. Ignored because insecure.

PI Low-level request used by **BP**. This is a Heirloom extension and currently unsupported.

pl [[+|-]*height*]
Change page length. Currently ignored.

pm Print names and sizes of macros, strings, and diversions to standard error output. Currently ignored.

pn [+|-]*number*
Change the page number of the next page. Currently ignored.

pnr Print all number registers on standard error output. Currently ignored.

po [[+|-]*offset*]
Set a horizontal page offset. If no argument is specified, the page offset is reverted to its previous value. If a sign is specified, the new page offset is calculated relative to the current one; otherwise, it is absolute. The argument follows the syntax of "Scaling Widths" and the default scaling unit is m.

ps [[+|-]size]
Change point size. Currently ignored.

psbb *filename*
Retrieve the bounding box of a PostScript file. Currently unsupported.

pshape *indent length ...*
Set a special shape for the current paragraph. This is a Heirloom extension and currently unsupported.

pso *command*
Include output of a shell command. Ignored because insecure.

ptr Print the names and positions of all traps on standard error output. This is a groff extension and currently ignored.

pvs [[+|-]*height*]
Change post-vertical spacing. This is a groff extension and currently ignored.

rchar *glyph ...*
Remove glyph definitions. Currently unsupported.

rd [*prompt* [*argument ...*]]
Read from standard input. Currently ignored.

recursionlimit *maxrec maxtail*
Set the maximum stack depth for recursive macros. This is a Heirloom extension and currently ignored.

return [*twice*]
Exit the presently executed macro and return to the caller. The argument is currently ignored.

rfschar *font glyph ...*
Remove font-specific fallback glyph definitions. Currently unsupported.

rhang *font char ... afmunits*
Hang characters at right margin. This is a Heirloom extension and currently ignored.

rj [*N*]
Justify the next *N* input lines to the right margin without filling. *N* defaults to 1. An argument of 0 or less ends right adjustment.

rm *macroname*
Remove a request, macro or string.

rn *oldname newname*
Rename a request, macro, diversion, or string. In *mandoc*(1), user-defined macros, *mdoc*(7) and *man*(7) macros, and user-defined strings can be renamed, but renaming of predefined strings and of **roff** requests is not supported, and diversions are not implemented at all.

rnn *oldname newname*
Rename a number register. Currently unsupported.

rr *register*
Remove a register.

rs End no-space mode. Currently ignored.

rt [*dist*]
Return to marked vertical position. Currently ignored.

schar *glyph* [*string*]
Define global fallback glyph. This is a groff extension and currently unsupported.

sentchar *char ...*
Define sentence-ending characters. This is a Heirloom extension and currently ignored.

shc [*glyph*]
Change the soft hyphen character. Currently ignored.

shift [*number*]
Shift macro arguments *number* times, by default once: \\$i becomes what \\$i+number was. Also decrement \n(.$ by *number*.

sizes *size ...*
Define permissible point sizes. This is a groff extension and currently ignored.

so *filename*
Include a source file. The file is read and its contents processed as input in place of the **so** request line. To avoid inadvertent inclusion of unrelated files, *mandoc*(1) only accepts relative paths not containing the strings "../" and "/..".

This request requires *man*(1) to change to the right directory before calling *mandoc*(1), per convention to the root of the manual tree. Typical usage looks like:

```
.so man3/Xcursor.3
```

As the whole concept is rather fragile, the use of **so** is discouraged. Use *ln*(1) instead.

sp [*height*]
Break the output line and emit vertical space. The argument follows the syntax of “Scaling Widths” and defaults to one blank line (1v).

spacewidth [1 | 0]
Set the space width from the font metrics file. This is a Heirloom extension and currently ignored.

special [*font ...*]
Define a special font. This is a groff extension and currently ignored.

spreadwarn [*width*]
Warn about wide spacing between words. Currently ignored.

ss *wordspace* [*sentencespace*]
Set space character size. Currently ignored.

sty *position style*
Associate style with a font position. This is a groff extension and currently ignored.

substring *stringname startpos* [*endpos*]
Replace a user-defined string with a substring. Currently unsupported.

sv [*height*]
Save vertical space. Currently ignored.

sy *command*
Execute shell command. Ignored because insecure.

T& Re-start a table layout, retaining the options of the prior table invocation. See **TS**.

ta [*width ...* [T *width ...*]]
Set tab stops. Each *width* argument follows the syntax of "Scaling Widths". If prefixed by a plus sign, it is relative to the previous tab stop. The arguments after the T marker are used repeatedly as often as needed; for each reuse, they are taken relative to the last previously established tab stop. When **ta** is called without arguments, all tab stops are cleared.

tc [*glyph*]
Change tab repetition character. Currently unsupported.

TE End a table context. See **TS**.

ti [+|-]*width*
Break the output line and indent the next output line by *width*. If a sign is specified, the temporary indentation is calculated relative to the current indentation; otherwise, it is absolute. The argument follows the syntax of "Scaling Widths" and the default scaling unit is m.

tkf *font minps width1 maxps width2*
Enable track kerning for a font. Currently ignored.

tl *'left'center'right'*
Print a title line. Currently unsupported.

tm *string*
Print to standard error output. Currently ignored.

tm1 *string*
Print to standard error output, allowing leading blanks. This is a groff extension and currently ignored.

tmc *string*
Print to standard error output without a trailing newline. This is a groff extension and currently ignored.

tr *glyph glyph ...*
Output character translation. The first glyph in each pair is replaced by the second one. Character escapes can be used; for example,

```
tr \(xx\(yy
```

replaces all invocations of \(xx with \(yy.

track *font minps width1 maxps width2*
Static letter space tracking. This is a Heirloom extension and currently ignored.

transchar *char ...*
Define transparent characters for sentence-ending. This is a Heirloom extension and currently ignored.

trf *filename*
Output the contents of a file, disallowing invalid characters. This is a groff extension and ignored because insecure.

trimat *left top width height*
Set the TrimBox page parameter for PDF generation. This is a Heirloom extension and currently ignored.

trin *glyph glyph ...*
Output character translation, ignored by **asciify**. Currently unsupported.

trnt *glyph glyph ...*
Output character translation, ignored by \!. Currently unsupported.

troff Force troff mode. This is a groff extension and currently ignored.

TS Begin a table, which formats input in aligned rows and columns. See *tbl*(7) for a description of the tbl language.

uf *font*
Globally set the underline font. Currently ignored.

ul [*N*]
Underline next *N* input lines. Currently ignored.

unformat *divname*
Unformat spaces and tabs in a diversion. Currently unsupported.

unwatch *macroname*
Disable notification for string or macro. This is a Heirloom extension and currently ignored.

unwatchn *register*
Disable notification for register. This is a Heirloom extension and currently ignored.

vpt [1 | 0]
Enable or disable vertical position traps. This is a groff extension and currently ignored.

vs [[+|-]*height*]
Change vertical spacing. Currently ignored.

warn *flags*
Set warning level. Currently ignored.

warnscale *si*
Set the scaling indicator used in warnings. This is a groff extension and currently ignored.

watch *macroname*
Notify on change of string or macro. This is a Heirloom extension and currently ignored.

watchlength *maxlength*
On change, report the contents of macros and strings up to the specified length. This is a Heirloom extension and currently ignored.

watchn *register*
: Notify on change of register. This is a Heirloom extension and currently ignored.

wh *dist* [*macroname*]
: Set a page location trap. Currently unsupported.

while *condition body*
: Repeated execution while a *condition* is true, with syntax similar to **if**. Currently implemented with two restrictions: cannot nest, and each loop must start and end in the same scope.

write ["]*string*
: Write to an open file. Ignored because insecure.

writec ["]*string*
: Write to an open file without appending a newline. Ignored because insecure.

writem *macroname*
: Write macro or string to an open file. Ignored because insecure.

xflag *level*
: Set the extension level. This is a Heirloom extension and currently ignored.

Numerical expressions

The **nr**, **if**, and **ie** requests accept integer numerical expressions as arguments. These are always evaluated using the C *int* type; integer overflow works the same way as in the C language. Numbers consist of an arbitrary number of digits '0' to '9' prefixed by an optional sign '+' or '-'. Each number may be followed by one optional scaling unit described below "Scaling Widths". The following equations hold:

```
1i = 6v = 6P = 10m = 10n = 72p = 1000M = 240u = 240
254c = 100i = 24000u = 24000
1f = 65536u = 65536
```

The following binary operators are implemented. Unless otherwise stated, they behave as in the C language:

+ addition
- subtraction
∗ multiplication
/ division
% remainder of division
< less than
> greater than
==
: equal to
= equal to, same effect as **==** (this differs from C)
<=
: less than or equal to
>=
: greater than or equal to
<>
: not equal to (corresponds to C **!=**; this one is of limited portability, it is supported by Heirloom roff, but not by groff)
& logical and (corresponds to C **&&**)
: logical or (corresponds to C **||**)
<?
: minimum (not available in C)
>?
: maximum (not available in C)

There is no concept of precedence; evaluation proceeds from left to right, except when subexpressions are enclosed in parentheses. Inside parentheses, whitespace is ignored.

ESCAPE SEQUENCE REFERENCE

The *mandoc*(1) **roff** parser recognises the following escape sequences. In *mdoc*(7) and *man*(7) documents, using escape sequences is discouraged except for those described in the "LANGUAGE SYNTAX" section above.

A backslash followed by any character not listed here simply prints that character itself.

\<newline>
: A backslash at the end of an input line can be used to continue the logical input line on the next physical input line, joining the text on both lines together as if it were on a single input line.

\<space>
: The escape sequence backslash-space ('\ ') is an unpaddable space-sized non-breaking space character; see "Whitespace" and *mandoc_char*(7).

\! Embed text up to and including the end of the input line into the current diversion or into intermediate output without interpreting requests, macros, and escapes. Currently unsupported.

\" The rest of the input line is treated as "Comments".

\# Line continuation with comment. Discard the rest of the physical input line and continue the logical input line on the next physical input line, joining the text on both lines together as if it were on a single input line. This is a groff extension.

\$*arg* Macro argument expansion, see **de**.

\% Hyphenation allowed at this point of the word; ignored by *mandoc*(1).

\& Non-printing zero-width character, often used for various kinds of escaping; see "Whitespace", *mandoc_char*(7), and the "MACRO SYNTAX" and "Delimiters" sections in *mdoc*(7).

\' Acute accent special character; use **\(aa** instead.

\(*cc* "Special Characters" with two-letter names, see *mandoc_char*(7).

\) Zero-width space transparent to end-of-sentence detection; ignored by *mandoc*(1).

[name***]**
: Interpolate the string with the *name*. For short names, there are variants *****_c_ and ***(**_cc_.

  One string is predefined on the **roff** language level: ***(.T** expands to the name of the output device, for example ascii, utf8, ps, pdf, html, or markdown.

  Macro sets traditionally predefine additional strings which are not portable and differ across implementations. Those supported by *mandoc*(1) are listed in *mandoc_char*(7).

  Strings can be defined, changed, and deleted with the **ds**, **as**, and **rm** requests.

\, Left italic correction (groff extension); ignored by *mandoc*(1).

\- Special character "mathematical minus sign"; see *mandoc_char*(7) for details.

\/ Right italic correction (groff extension); ignored by *mandoc*(1).

\: Breaking the line is allowed at this point of the word without inserting a hyphen.

\? Embed the text up to the next **\?** into the current diversion without interpreting requests, macros, and escapes. This is a groff extension and currently unsupported.

\[*name***]**
: "Special Characters" with names of arbitrary length, see *mandoc_char*(7).

\^ One-twelfth em half-narrow space character, effectively zero-width in *mandoc*(1).

_ Underline special character; use **\(ul** instead.

\‘ Grave accent special character; use **\(ga** instead.

\{ Begin conditional input; see **if**.

\| One-sixth em narrow space character, effectively zero-width in *mandoc*(1).

\} End conditional input; see **if**.

\~ Paddable non-breaking space character.

\0 Digit width space character.

\A'*string*'
Anchor definition; ignored by *mandoc*(1).

\a Leader character; ignored by *mandoc*(1).

\B'*string*'
Interpolate ‘1’ if *string* conforms to the syntax of “Numerical expressions” explained above or ‘0’ otherwise.

\b'*string*'
Bracket building function; ignored by *mandoc*(1).

\C'*name*'
“Special Characters” with names of arbitrary length.

\c When encountered at the end of an input text line, the next input text line is considered to continue that line, even if there are request or macro lines in between. No whitespace is inserted.

\D'*string*'
Draw graphics function; ignored by *mandoc*(1).

\d Move down by half a line; ignored by *mandoc*(1).

\E Escape character intended to not be interpreted in copy mode. In *mandoc*(1), it currently does the same as **** itself.

\e Backslash special character.

\F[*name***]**
Switch font family (groff extension); ignored by *mandoc*(1). For short names, there are variants **\F***c* and **\F(***cc*.

\f[*name***]**
Switch to the font *name*, see “Font Selection”. For short names, there are variants **\f***c* and **\f(***cc*. An empty name **\f[]** defaults to **\fP**.

\g[*name***]**
Interpolate the format of a number register; ignored by *mandoc*(1). For short names, there are variants **\g***c* and **\g(***cc*.

\H'[+|-]*number*'
Set the height of the current font; ignored by *mandoc*(1).

\h'[|]*width*'
Horizontal motion. If the vertical bar is given, the motion is relative to the current indentation. Otherwise, it is relative to the current position. The default scaling unit is `m`.

\k[*name***]**
Mark horizontal input place in register; ignored by *mandoc*(1). For short names, there are variants **\k***c* and **\k(***cc*.

\L'*number*[*c*]**'**
Vertical line drawing function; ignored by *mandoc*(1).

\l'*width*[*c*]**'**
Draw a horizontal line of *width* using the glyph *c*.

\M[*name***]**
Set fill (background) color (groff extension); ignored by *mandoc*(1). For short names, there are variants **\M***c* and **\M(***cc*.

\m[*name***]**
Set glyph drawing color (groff extension); ignored by *mandoc*(1). For short names, there are variants **\m***c* and **\m(***cc*.

\N'*number***'**
Character *number* on the current font.

\n[+|-]**[***name***]**
Interpolate the number register *name*. For short names, there are variants **\n***c* and **\n(***cc*. If the optional sign is specified, the register is first incremented or decremented by the *stepsize* that was specified in the relevant **nr** request, and the changed value is interpolated.

\O*digit*, **\O[5**arguments**]**
Suppress output. This is a groff extension and currently unsupported. With an argument of **1**, **2**, **3**, or **4**, it is ignored.

\o'*string***'**
Overstrike, writing all the characters contained in the *string* to the same output position. In terminal and HTML output modes, only the last one of the characters is visible.

\p Break the output line at the end of the current word.

\R'*name* [+|-]*number***'**
Set number register; ignored by *mandoc*(1).

\r Move up by one line; ignored by *mandoc*(1).

\S'*number***'**
Slant output; ignored by *mandoc*(1).

\s'[+|-]*number***'**
Change point size; ignored by *mandoc*(1). Alternative forms **\s**[+|-]*n*, **\s**[+|-]**'***number***'**, **\s[**[+|-]*number***]**, and **\s**[+|-]**[***number***]** are also parsed and ignored.

\t Horizontal tab; ignored by *mandoc*(1).

\u Move up by half a line; ignored by *mandoc*(1).

\V[*name***]**
Interpolate an environment variable; ignored by *mandoc*(1). For short names, there are variants **\V***c* and **\V(***cc*.

\v'*number***'**
Vertical motion; ignored by *mandoc*(1).

\w'*string***'**
Interpolate the width of the *string*. The *mandoc*(1) implementation assumes that after expansion of user-defined strings, the *string* only contains normal characters, no escape sequences, and that each character has a width of 24 basic units.

\X'*string***'**
Output *string* as device control function; ignored in nroff mode and by *mandoc*(1).

\x'*number***'**
Extra line space function; ignored by *mandoc*(1).

\Y[*name***]**
Output a string as a device control function; ignored in nroff mode and by *mandoc*(1). For short names, there are variants **\Y***c* and **\Y(***cc*.

\Z'*string***'**
Print *string* with zero width and height; ignored by *mandoc*(1).

\z Output the next character without advancing the cursor position.

COMPATIBILITY

The *mandoc*(1) implementation of the **roff** language is incomplete. Major unimplemented features include:

- For security reasons, *mandoc*(1) never reads or writes external files except via **so** requests with safe relative paths.
- There is no automatic hyphenation, no adjustment to the right margin, and very limited support for centering; the output is always set flush-left.
- Support for setting tabulator and leader characters is missing, and support for manually changing indentation is limited.
- The ‘u’ scaling unit is the default terminal unit. In traditional troff systems, this unit changes depending on the output media.
- Width measurements are implemented in a crude way and often yield wrong results. Support for explicit movement requests and escapes is limited.
- There is no concept of output pages, no support for floats, graphics drawing, and picture inclusion; terminal output is always continuous.
- Requests regarding color, font families, font sizes, and glyph manipulation are ignored. Font support is very limited. Kerning is not implemented, and no ligatures are produced.
- The "'" macro control character does not suppress output line breaks.
- Diversions and environments are not implemented, and support for traps is very incomplete.
- Use of macros is not supported inside *tbl*(7) code.

The special semantics of the nS number register is an idiosyncrasy of OpenBSD manuals and not supported by other *mdoc*(7) implementations.

SEE ALSO

mandoc(1), *eqn*(7), *man*(7), *mandoc_char*(7), *mdoc*(7), *tbl*(7)

Joseph F. Ossanna and Brian W. Kernighan, *Troff User’s Manual*, *AT&T Bell Laboratories*, Computing Science Technical Report, 54, http://www.kohala.com/start/troff/cstr54.ps, Murray Hill, New Jersey, 1976 and 1992.

Joseph F. Ossanna, Brian W. Kernighan, and Gunnar Ritter, *Heirloom Documentation Tools Nroff/Troff User’s Manual*, http://heirloom.sourceforge.net/doctools/troff.pdf, September 17, 2007.

HISTORY

The RUNOFF typesetting system, whose input forms the basis for **roff**, was written in MAD and FAP for the CTSS operating system by Jerome E. Saltzer in 1964. Doug McIlroy rewrote it in BCPL in 1969, renaming it **roff**. Dennis M. Ritchie rewrote McIlroy’s **roff** in PDP-11 assembly for Version 1 AT&T UNIX, Joseph F. Ossanna improved roff and renamed it nroff for Version 2 AT&T UNIX, then ported nroff to C as troff, which Brian W. Kernighan released with Version 7 AT&T UNIX. In 1989, James Clark re-implemented troff in C++, naming it groff.

AUTHORS

This **roff** reference was written by Kristaps Dzonsons <kristaps@bsd.lv> and Ingo Schwarze <schwarze@openbsd.org>.

NAME

tbl — tbl language reference for mandoc

DESCRIPTION

The **tbl** language formats tables. It is used within *mdoc*(7) and *man*(7) pages. This manual describes the subset of the **tbl** language accepted by the *mandoc*(1) utility.

Each table is started with a *roff*(7) **TS** macro, consist of at most one line of "Options", one or more "Layout" lines, one or more "Data" lines, and ends with a **TE** macro. All input must be 7-bit ASCII.

Options

If the first input line of a table ends with a semicolon, it contains case-insensitive options separated by spaces, tabs, or commas. Otherwise, it is interpreted as the first "Layout" line.

The following options are available. Some of them require arguments enclosed in parentheses:

`allbox`
: Draw a single-line box around each table cell.

`box`
: Draw a single-line box around the table. For GNU compatibility, this may also be invoked with `frame`.

`center`
: Center the table instead of left-adjusting it. For GNU compatibility, this may also be invoked with `centre`.

`decimalpoint`
: Use the single-character argument as the decimal point with the `n` layout key. This is a GNU extension.

`delim`
: Use the two characters of the argument as *eqn*(7) delimiters. Currently unsupported.

`doublebox`
: Draw a double-line box around the table. For GNU compatibility, this may also be invoked with `doubleframe`.

`expand`
: Increase the width of the table to the current line length. Currently ignored.

`linesize`
: Draw lines with the point size given by the unsigned integer argument. Currently ignored.

`nokeep`
: Allow page breaks within the table. This is a GNU extension and currently ignored.

`nospaces`
: Ignore leading and trailing spaces in data cells. This is a GNU extension.

`nowarn`
: Suppress warnings about tables exceeding the current line length. This is a GNU extension and currently ignored.

`tab`
: Use the single-character argument as a delimiter between data cells. By default, the horizontal tabulator character is used.

Layout

The table layout follows an "Options" line or a *roff*(7) **TS** or **T&** macro. Each layout line specifies how one line of "Data" is formatted. The last layout line ends with a full stop. It also applies to all remaining data lines. Multiple layout lines can be joined by commas on a single physical input line.

Each layout line consists of one or more layout cell specifications, optionally separated by whitespace. The following case-insensitive key characters start a new cell specification:

`c` Center the string in this cell.

`r` Right-justify the string in this cell.

`l` Left-justify the string in this cell.

`n` Justify a number around its last decimal point. If no decimal point is found in the number, it is assumed to trail the number.

`s` Horizontally span columns from the last non-`s` layout cell. It is an error if a column span follows a _ or `=` cell, or comes first on a layout line. The combined cell as a whole consumes only one cell of the corresponding data line.

`a` Left-justify a string and pad with one space.

`^` Vertically span rows from the last non-`^` layout cell. It is an error to invoke a vertical span on the first layout line. Unlike a horizontal span, a vertical span consumes a data cell and discards the content.

_ Draw a single horizontal line in this cell. This consumes a data cell and discards the content. It may also be invoked with `-`.

`=` Draw a double horizontal line in this cell. This consumes a data cell and discards the content.

Each cell key may be followed by zero or more of the following case-insensitive modifiers:

`b` Use a bold font for the contents of this cell.

`d` Move content down to the last row of this vertical span. Currently ignored.

`e` Make this column wider to match the maximum width of any other column also having the `e` modifier.

`f` The next one or two characters select the font to use for this cell. One-character font names must be followed by a blank or period. See the *roff*(7) manual for supported font names.

`i` Use an italic font for the contents of this cell.

`m` Specify a cell start macro. This is a GNU extension and currently unsupported.

`p` Set the point size to the following unsigned argument, or change it by the following signed argument. Currently ignored.

`v` Set the vertical line spacing to the following unsigned argument, or change it by the following signed argument. Currently ignored.

`t` Do not vertically center content in this vertical span, leave it in the top row. Currently ignored.

`u` Move cell content up by half a table row. Currently ignored.

`w` Specify a minimum column width.

`x` After determining the width of all other columns, distribute the rest of the line length among all columns having the `x` modifier.

`z` Do not use this cell for determining the width of this column.

`|` Draw a single vertical line to the right of this cell.

`||`
Draw a double vertical line to the right of this cell.

If a modifier consists of decimal digits, it specifies a minimum spacing in units of `n` between this column and the next column to the right. The default is 3. If there is a vertical line, it is drawn inside the spacing.

Data

The data section follows the last “Layout” line. Each data line consists of one or more data cells, delimited by `tab` characters.

If a data cell contains only the two bytes '`\^`', the cell above spans to this row, as if the layout specification of this cell were `^`.

If a data cell contains only the single character '`_`' or '`=`', a single or double horizontal line is drawn across the cell, joining its neighbours. If a data cell contains only the two character sequence '`\_`' or '`\=`', a single or double horizontal line is drawn inside the cell, not joining its neighbours. If a data line contains nothing but the single character '`_`' or '`=`', a horizontal line across the whole table is inserted without consuming a layout row.

In place of any data cell, a text block can be used. It starts with **`T{`** at the end of a physical input line. Input line breaks inside the text block neither end the text block nor its data cell. It only ends if **`T}`** occurs at the beginning of a physical input line and is followed by an end-of-cell indicator. If the **`T}`** is followed by the end of the physical input line, the text block, the data cell, and the data line ends at this point. If the **`T}`** is followed by the `tab` character, only the text block and the data cell end, but the data line continues with the data cell following the `tab` character. If **`T}`** is followed by any other character, it does not end the text block, which instead continues to the following physical input line.

EXAMPLES

String justification and font selection:

```
.TS
rb c  lb
r  ci l.
r       center  l
ri      ce      le
right   c       left
.TE
```

r	center	**l**
ri	*ce*	le
right	*c*	left

Some ports in OpenBSD 6.1 to show number alignment and line drawing:

```
.TS
box tab(:);
r| l
r  n.
software:version
_
AFL:2.39b
Mutt:1.8.0
Ruby:1.8.7.374
TeX Live:2015
.TE
```

software	version
AFL	2.39b
Mutt	1.8.0
Ruby	1.8.7.374
TeX Live	2015

Spans and skipping width calculations:

```
.TS
box tab(:);
lz  s | rt
lt| cb| ^
```

```
^ | rz  s.
left:r
l:center:
:right
.TE
```

left		r
l	**center**	
	right	

Text blocks, specifying spacings and specifying and equalizing column widths, putting lines into individual cells, and overriding `allbox`:

```
.TS
allbox tab(:);
le le||7 lw10.
The fourth line:_:line 1
of this column:=:line 2
determines:_:line 3
the column width.:T{
This text is too wide to fit into a column of width 17.
T}:line 4
T{
No break here.
T}::line 5
.TE
```

The fourth line		line 1
of this column		line 2
determines		line 3
the column width.	This text is too wide to fit into a column of width 17.	line 4
No break here.		line 5

These examples were constructed to demonstrate many **`tbl`** features in a compact way. In real manual pages, keep tables as simple as possible. They usually look better, are less fragile, and are more portable.

COMPATIBILITY

The *mandoc*(1) implementation of **`tbl`** doesn't support *mdoc*(7) and *man*(7) macros and *eqn*(7) equations inside tables.

SEE ALSO

mandoc(1), *man*(7), *mandoc_char*(7), *mdoc*(7), *roff*(7)

M. E. Lesk, *Tbl — A Program to Format Tables*, June 11, 1976.

HISTORY

The tbl utility, a preprocessor for troff, was originally written by M. E. Lesk at Bell Labs in 1975. The GNU reimplementation of tbl, part of the groff package, was released in 1990 by James Clark. A standalone tbl implementation was written by Kristaps Dzonsons in 2010. This formed the basis of the implementation that first appeared in OpenBSD 4.9 as a part of the *mandoc*(1) utility.

AUTHORS

This **`tbl`** reference was written by Kristaps Dzonsons <kristaps@bsd.lv> and
Ingo Schwarze <schwarze@openbsd.org>.

BUGS

In `-T utf8` output mode, heavy lines are drawn instead of double lines. This cannot be improved because the Unicode standard only provides an incomplete set of box drawing characters with double lines, whereas it provides a full set of box drawing characters with heavy lines. It is unlikely this can be improved in the future because the box drawing characters are already marked in Unicode as characters intended only for backward compatibility with legacy systems, and their use is not encouraged. So it seems unlikely that the missing ones might get added in the future.

NAME

catman — format all manual pages below a directory

SYNOPSIS

catman [−I os=*name*] [−T *output*] *srcdir dstdir*

DESCRIPTION

The **catman** utility assumes that all files below *srcdir* are manual pages in *mdoc*(7) and *man*(7) format and formats all of them, storing the formatted versions in the same relative paths below *dstdir*. Subdirectories of *dstdir* are created as needed. Existing files are not explicitly deleted, but possibly overwritten.

The options are as follows:

−I os=*name*
: Override the default operating system *name* for the *mdoc*(7) **Os** and for the *man*(7) **TH** macro.

−T *output*
: Output format. The *output* argument can be `ascii`, `utf8`, or `html`; see *mandoc*(1). In `html` output mode, the `fragment` output option is implied. Other output options are not supported.

IMPLEMENTATION NOTES

Since this version avoids *fork*(2) and *exec*(3) overhead and uses the much faster **mandoc** parsers and formatters rather than **groff**, it may be about one order of magnitude faster than other **catman** implementations.

EXIT STATUS

The **catman** utility exits 0 on success, and >0 if an error occurs.

Possible errors include:

- missing, invalid, or excessive command line arguments
- failure to change the current working directory to *srcdir*
- failure to open *dstdir*
- communication failure with *mandocd*(8)
- resource exhaustion, for example file descriptor, process table, or memory exhaustion

Except for memory exhaustion and similar system-level failures, failures while trying to open, read, parse, or format individual manual pages, to save individual formatted files to the file system, or even to create directories do not cause **catman** to return an error exit status. In such cases, **catman** will simply continue with the next file or subdirectory.

SEE ALSO

mandoc(1), *mandocd*(8)

HISTORY

A **catman** utility first appeared in FreeBSD 1.0. Other, incompatible implementations appeared in NetBSD 1.0 and in **man-db** 2.2.

This version appeared in version 1.14.1 of the **mandoc** toolkit.

AUTHORS

The first **catman** implementation was a short shell script by Christoph Robitschko in July 1993.

The NetBSD implementations were written by J. T. Conklin <jtc@netbsd.org> in 1993, Christian E. Hopps <chopps@netbsd.org> in 1994, and Dante Profeta <dante@netbsd.org> in 1999; the **man-db** implementation by Graeme W. Wilford in 1994; and the FreeBSD implementations by Wolfram Schneider <wosch@freebsd.org> in 1995 and John Rochester <john@jrochester.org> in 2002.

The concept of the present version was designed and implemented by Michael Stapelberg <stapelberg@debian.org> in 2017. Option and argument handling and directory iteration was added by Ingo Schwarze <schwarze@openbsd.org>.

CAVEATS

All versions of **catman** are incompatible with each other because each caters to the needs of a specific operating system, for example regarding directory structures and file naming conventions.

This version is more flexible than the others in so far as it does not assume any particular directory structure or naming convention. That flexibility comes at the price of not being able to change the names and relative paths of the source files when reusing them to store the formatted files, of not supporting any configuration file formats or environment variables, and of being unable to scan for and remove junk files in *dstdir*.

Currently, **catman** always reformats each page, even if the formatted version is newer than the source version.

NAME

makewhatis — index UNIX manuals

SYNOPSIS

makewhatis [−aDnpQ] [−T utf8] [−C *file*]
makewhatis [−aDnpQ] [−T utf8] *dir* ...
makewhatis [−DnpQ] [−T utf8] −d *dir* [*file* ...]
makewhatis [−Dnp] [−T utf8] −u *dir* [*file* ...]
makewhatis [−DQ] −t *file* ...

DESCRIPTION

The **makewhatis** utility extracts keywords from Unix manuals and indexes them in a database for fast retrieval by *apropos*(1), *whatis*(1), and *man*(1)'s −k option.

By default, **makewhatis** creates a database in each *dir* using the files **man***section*/[*arch*/]*title*.*section* and **cat***section*/[*arch*/]*title*.**0** in that directory. Existing databases are replaced. If a directory contains no manual pages, no database is created in that directory. If *dir* is not provided, **makewhatis** uses the default paths stipulated by *man.conf*(5).

The arguments are as follows:

−a
: Use all directories and files found below *dir* ....

−C *file*
: Specify an alternative configuration *file* in *man.conf*(5) format.

−D
: Display all files added or removed to the index. With a second −D, also show all keywords added for each file.

−d *dir*
: Merge (remove and re-add) *file* ... to the database in *dir*.

−n
: Do not create or modify any database; scan and parse only, and print manual page names and descriptions to standard output.

−p
: Print warnings about potential problems with manual pages to the standard error output.

−Q
: Quickly build reduced-size databases by reading only the NAME sections of manuals. The resulting databases will usually contain names and descriptions only.

−T utf8
: Use UTF-8 encoding instead of ASCII for strings stored in the databases.

−t *file* ...
: Check the given *files* for potential problems. Implies −a, −n, and −p. All diagnostic messages are printed to the standard output; the standard error output is not used.

−u *dir*
: Remove *file* ... from the database in *dir*. If that causes the database to become empty, also delete the database file.

If fatal parse errors are encountered while parsing, the offending file is printed to stderr, omitted from the index, and the parse continues with the next input file.

ENVIRONMENT

MANPATH
: A colon-separated list of directories to create databases in. Ignored if a *dir* argument or the −t option is specified.

FILES

mandoc.db
: A database of manpages relative to the directory of the file. This file is portable across architectures and systems, so long as the manpage hierarchy it indexes does not change.

/etc/man.conf
The default *man*(1) configuration file.

EXIT STATUS

The **makewhatis** utility exits with one of the following values:

0 No errors occurred.
5 Invalid command line arguments were specified. No input files have been read.
6 An operating system error occurred, for example memory exhaustion or an error accessing input files. Such errors cause **makewhatis** to exit at once, possibly in the middle of parsing or formatting a file. The output databases are corrupt and should be removed.

SEE ALSO

apropos(1), *man*(1), *whatis*(1), *man.conf*(5)

HISTORY

A **makewhatis** utility first appeared in 2BSD. It was rewritten in *perl*(1) for OpenBSD 2.7 and in C for OpenBSD 5.6.

The *dir* argument first appeared in NetBSD 1.0; the options `-dpt` in OpenBSD 2.7; the option `-u` in OpenBSD 3.4; and the options `-aCDnQT` in OpenBSD 5.6.

AUTHORS

Bill Joy wrote the original BSD **makewhatis** in February 1979, Marc Espie started the Perl version in 2000, and the current version of **makewhatis** was written by Kristaps Dzonsons <kristaps@bsd.lv> and Ingo Schwarze <schwarze@openbsd.org>.

NAME

man.cgi — CGI program to search and display manual pages

DESCRIPTION

The **man.cgi** CGI program searches for manual pages on a WWW server and displays them to HTTP clients, providing functionality equivalent to the *man*(1) and *apropos*(1) utilities. It can use multiple manual trees in parallel.

HTML search interface

At the top of each generated HTML page, **man.cgi** displays a search form containing these elements:

1. An input box for search queries, expecting either a name of a manual page or an *expression* using the syntax described in the *apropos*(1) manual; filling this in is required for each search.

   The expression is broken into words at whitespace. Whitespace characters and backslashes can be escaped by prepending a backslash. The effect of prepending a backslash to another character is undefined; in the current implementation, it has no effect.

2. A *man*(1) submit button. The string in the input box is interpreted as the name of a manual page.

3. An *apropos*(1) submit button. The string in the input box is interpreted as a search *expression*.

4. A dropdown menu to optionally select a manual section. If one is provided, it has the same effect as the *man*(1) and *apropos*(1) `-s` option. Otherwise, pages from all sections are shown.

5. A dropdown menu to optionally select an architecture. If one is provided, it has the same effect as the *man*(1) and *apropos*(1) `-S` option. By default, pages for all architectures are shown.

6. A dropdown menu to select a manual tree. If the configuration file */var/www/man/manpath.conf* contains only one manpath, the dropdown menu is not shown. By default, the first manpath given in the file is used.

Program output

The **man.cgi** program generates five kinds of output pages:

The index page.
: This is returned when calling **man.cgi** without *PATH_INFO* and without a *QUERY_STRING*. It serves as a starting point for using the program and shows the search form only.

A list page.
: Lists are returned when searches match more than one manual page. The first column shows the names and section numbers of manuals as clickable links. The second column shows the one-line descriptions of the manuals. For *man*(1) style searches, the content of the first manual page follows the list.

A manual page.
: This output format is used when a search matches exactly one manual page, or when a link on a list page or an **Xr** link on another manual page is followed.

A no-result page.
: This is shown when a search request returns no results - either because it violates the query syntax, or because the search does not match any manual pages.

An error page.
: This cannot happen by merely clicking the "Search" button, but only by manually entering an invalid URI. It does not show the search form, but only an error message and a link back to the index page.

Setup

For each manual tree, create one first-level subdirectory below */var/www/man*. The name of one of these directories is called a "manpath" in the context of **man.cgi**. Create a single ASCII text file */var/www/man/manpath.conf* containing the names of these directories, one per line. The directory given first is used as the default manpath.

Inside each of these directories, use the same directory and file structure as found below */usr/share/man*, that is, second-level subdirectories */var/www/man/*/man1*, */var/www/man/*/man2* etc. containing source *mdoc*(7) and *man*(7) manuals with file name extensions matching the section numbers, second-level subdirectories */var/www/man/*/cat1*, */var/www/man/*/cat2* etc. containing preformatted manuals with the file name extension '0', and optional third-level subdirectories for architectures. Use *makewhatis*(8) to create a *mandoc.db*(5) database inside each manpath.

Configure your web server to execute CGI programs located in */cgi-bin*. When using OpenBSD *httpd*(8), the *slowcgi*(8) proxy daemon is needed to translate FastCGI requests to plain old CGI.

To compile **man.cgi**, first copy *cgi.h.example* to *cgi.h* and edit it according to your needs. It contains the following compile-time definitions:

COMPAT_OLDURI
: Only useful for running on www.openbsd.org to deal with old URIs containing "manpath=OpenBSD " where the blank character has to be translated to a hyphen. When compiling for other sites, this definition can be deleted.

`CSS_DIR`
: An optional file system path to the directory containing the file *mandoc.css*, to be specified relative to the server's document root, and to be specified without a trailing slash. When empty, the CSS file is assumed to be in the document root. Otherwise, a leading slash is needed. This is used in generated HTML code.

`CUSTOMIZE_TITLE`
: An ASCII string to be used for the HTML <TITLE> element.

`MAN_DIR`
: A file system path to the **man.cgi** data directory relative to the web server *chroot*(2) directory, to be specified with a leading slash and without a trailing slash. It needs to have at least one component; the root directory cannot be used for this purpose. The files *manpath.conf*, *header.html*, and *footer.html* are looked up in this directory. It is also prepended to the manpath when opening *mandoc.db*(5) and manual page files.

`SCRIPT_NAME`
: The initial component of URIs, to be specified without leading and trailing slashes. It can be empty.

After editing *cgi.h*, run

```
make man.cgi
```

and copy the resulting binary to the proper location, for example using the command:

```
make installcgi
```

In addition to that, make sure the default manpath contains the files *man1/apropos.1* and *man8/man.cgi.8*, or the documentation links at the bottom of the index page will not work.

URI interface

man.cgi uniform resource identifiers are not needed for interactive use, but can be useful for deep linking. They consist of:

1. The `http://` or `https://` protocol specifier.
2. The host name.
3. The `SCRIPT_NAME`, preceded by a slash unless empty.
4. To show a single page, a slash, the manpath, another slash, and the name of the requested file, for example */OpenBSD-current/man1/mandoc.1*. This can be abbreviated according to the following syntax: [/*manpath*][/man*sec*][/*arch*]/*name*[.*sec*]

5. For searches, a query string starting with a question mark and consisting of *key*=*value* pairs, separated by ampersands, for example *?manpath=OpenBSD-current&query=mandoc*. Supported keys are `manpath`, `query`, `sec`, `arch`, corresponding to *apropos*(1) `-M`, *expression*, `-s`, `-S`, respectively, and `apropos`, which is a boolean parameter to select or deselect the *apropos*(1) query mode. For backward compatibility with the traditional **man.cgi**, `sektion` is supported as an alias for `sec`.

Restricted character set

For security reasons, in particular to prevent cross site scripting attacks, some strings used by **man.cgi** can only contain the following characters:

- lower case and upper case ASCII letters
- the ten decimal digits
- the dash ('-')
- the dot ('.')
- the slash ('/')
- the underscore ('_')

In particular, this applies to all manpaths and architecture names.

ENVIRONMENT

The web server may pass the following CGI variables to **man.cgi**:

SCRIPT_NAME
: The initial part of the URI passed from the client to the server, starting after the server's host name and ending before *PATH_INFO*. This is ignored by **man.cgi**. When constructing URIs for links and redirections, the `SCRIPT_NAME` preprocessor constant is used instead.

PATH_INFO
: The final part of the URI path passed from the client to the server, starting after the *SCRIPT_NAME* and ending before the *QUERY_STRING*. It is used by the `show` page to acquire the manpath and filename it needs.

QUERY_STRING
: The HTTP query string passed from the client to the server. It is the final part of the URI, after the question mark. It is used by the `search` page to acquire the named parameters it needs.

FILES

/var/www
: Default web server *chroot*(2) directory. All the following paths are specified relative to this directory.

/cgi-bin/man.cgi
: The usual file system path to the **man.cgi** program inside the web server *chroot*(2) directory. A different name can be chosen, but in any case, it needs to be configured in *httpd.conf*(5).

/htdocs The file system path to the server document root directory relative to the server *chroot*(2) directory. This is part of the web server configuration and not specific to **man.cgi**.

/htdocs/mandoc.css
: A style sheet for *mandoc*(1) HTML styling, referenced from each generated HTML page.

/man Default **man.cgi** data directory containing all the manual trees. Can be overridden by `MAN_DIR`.

/man/manpath.conf
: The list of available manpaths, one per line. If any of the lines in this file contains a slash ('/') or any character not contained in the "Restricted character set", **man.cgi** reports an internal server error and exits without doing anything.

/man/header.html
: An optional file containing static HTML code to be inserted right after opening the <BODY> element.

/man/footer.html
: An optional file containing static HTML code to be inserted right before closing the <BODY> element.

/man/OpenBSD-current/man1/mandoc.1
: An example *mdoc*(7) source file located below the "OpenBSD-current" manpath.

COMPATIBILITY

The **man.cgi** CGI program is call-compatible with queries from the traditional *man.cgi* script by Wolfram Schneider. However, the output looks quite different.

SEE ALSO

apropos(1), *mandoc.db*(5), *makewhatis*(8), *slowcgi*(8)

HISTORY

A version of **man.cgi** based on *mandoc*(1) first appeared in mdocml-1.12.1 (March 2012). The current *mandoc.db*(5) database format first appeared in OpenBSD 6.1.

AUTHORS

The **man.cgi** program was written by Kristaps Dzonsons <kristaps@bsd.lv> and is maintained by Ingo Schwarze <schwarze@openbsd.org>, who also designed and implemented the database format.

NAME

mandocd — server process to format manual pages in batch mode

SYNOPSIS

mandocd [`-I os=`*name*] [`-T` *output*] *socket_fd*

DESCRIPTION

The **mandocd** utility formats many manual pages without requiring *fork*(2) and *exec*(3) overhead in between. It does not require listing all the manuals to be formatted on the command line, and it supports writing each formatted manual to its own file descriptor.

This server requires that a connected UNIX domain *socket*(2) is already present at *exec*(3) time. Consequently, it cannot be started from the *sh*(1) command line because the shell cannot supply such a socket. Typically, the socket is created by the parent process using *socketpair*(2) before calling *fork*(2) and *exec*(3) on **mandocd**. The parent process will pass the file descriptor number as an argument to *exec*(3), formatted as a decimal ASCII-encoded integer. See *catman*(8) for a typical implementation of a parent process.

mandocd loops reading one-byte messages with *recvmsg*(2) from the file descriptor number *socket_fd*. It ignores the byte read and only uses the out-of-band auxiliary *struct cmsghdr* control data, typically supplied by the calling process using *CMSG_FIRSTHDR*(3). The parent process is expected to pass three file descriptors with each dummy byte. The first one is used for *mdoc*(7) or *man*(7) input, the second one for formatted output, and the third one for error output.

The options are as follows:

`-I os=`*name*
: Override the default operating system *name* for the *mdoc*(7) **Os** and for the *man*(7) **TH** macro.

`-T` *output*
: Output format. The *output* argument can be `ascii`, `utf8`, or `html`; see *mandoc*(1). In `html` output mode, the `fragment` output option is implied. Other output options are not supported.

After exhausting one input file descriptor, all three file descriptors are closed before reading the next dummy byte and control message.

When a zero-byte message is read, when the *socket_fd* is closed by the parent process, or when an error occurs, **mandocd** exits.

EXIT STATUS

The **mandocd** utility exits 0 on success, and >0 if an error occurs.

A zero-byte message or a closed *socket_fd* is considered success. Possible errors include:

- missing, invalid, or excessive *exec*(3) arguments
- *recvmsg*(2) failure, for example due to `EMSGSIZE`
- missing or unexpected control data, in particular a *cmsg_level* in the *struct cmsghdr* that differs from `SOL_SOCKET`, a *cmsg_type* that differs from `SCM_RIGHTS`, or a *cmsg_len* that is not three times the size of an *int*
- invalid file descriptors passed in the *CMSG_DATA*(3)
- resource exhaustion, in particular *dup*(2) or *malloc*(3) failure

Except for memory exhaustion and similar system-level failures, parsing and formatting errors do not cause **mandocd** to return an error exit status. Even after severe parsing errors, **mandocd** will simply accept and process the next input file descriptor.

SEE ALSO

mandoc(1), *mandoc*(3), *catman*(8)

HISTORY

The **mandocd** utility appeared in version 1.14.1 or the **mandoc** toolkit.

AUTHORS

The concept was designed and implemented by Michael Stapelberg <stapelberg@debian.org>. The *mandoc*(3) glue needed to make it a stand-alone process was added by Ingo Schwarze <schwarze@openbsd.org>.

CAVEATS

If the parsed manual pages contain *roff*(7) **.so** requests, **mandocd** needs to be started with the current working directory set to the root of the manual page tree. Avoid starting it in directories that contain secret files in any subdirectories, in particular in the user starting it has read access to these secret files.