groff

Collected Reference Pages

documenting version 1.24.2

Name

addftinfo – add font metrics to *troff* fonts for use with *groff*

Synopsis

addftinfo [**−asc−height** *n*] [**−body−depth** *n*] [**−body−height** *n*] [**−cap−height** *n*] [**−comma−depth** *n*] [**−desc−depth** *n*] [**−fig−height** *n*] [**−x−height** *n*] *resolution unit-width font*

addftinfo −−help

addftinfo −v
addftinfo −−version

Description

addftinfo reads an AT&T *troff* font description file *font*, adds further font metric information required by GNU *troff* (1), and writes the combined result to the standard output stream. The information added is derived from the font's existing parameters and assumptions about traditional *troff* names for characters. Among the font metrics added are the heights and depths of characters (how far each extends vertically above and below the baseline). The *resolution* and *unit-width* arguments should be the same as the corresponding parameters in the *DESC* file. *font* is the name of the file describing the font; if *font* ends with "**I**", the font is assumed to be oblique (or italic).

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

All other options change parameters that are used to derive the heights and depths. Like the existing quantities in the font description file, each value *n* is in *scaled points,* inches/*resolution* for a font whose type size is *unit-width*; see *groff_font*(5).

−asc−height *n*
: height of characters with ascenders, such as "b", "d", or "l"

−body−depth *n*
: depth of characters such as parentheses

−body−height *n*
: height of characters such as parentheses

−cap−height *n*
: height of uppercase letters such as "A"

−comma−depth *n*
: depth of a comma

−desc−depth *n*
: depth of characters with descenders, such as "p", "q", or "y"

−fig−height
: height of figures (numerals)

−x−height *n*
: height of lowercase letters without ascenders such as "x"

addftinfo makes no attempt to use the specified parameters to infer unspecified parameters. If a parameter is not specified, the default will be used. The defaults are chosen to produce reasonable values for a Times font.

Exit status

addftinfo exits with status **0** on successful operation, and status **2** if the program cannot interpret its command-line arguments.

See also

groff_font(5), *groff* (1), *groff_char*(7)

Name

afmtodit – adapt Adobe Font Metrics files for *groff* PostScript and PDF output

Synopsis

afmtodit [**–ckmnqsx**] [**–a** *slant*] [**–d** *device-description-file*] [**–e** *encoding-file*] [**–f** *internal-name*] [**–i** *italic-correction-factor*] [**–o** *output-file*] [**–w** *space-width*] *afm-file map-file font-description-file*

afmtodit ––help

afmtodit –v
afmtodit ––version

Description

afmtodit generates a font description file for use with *groff*(1)'s **ps** and **pdf** output devices from an Adobe Font Metric file, *afm-file*. *map-file* associates a *groff* ordinary or special character name with a PostScript glyph name. Output is written in *groff_font*(5) format to *font-description-file,* a file named for the intended *groff* font name (but see the **–o** option).

map-file should contain a sequence of lines of the form

```
ps-glyph groff-char
```

where *ps-glyph* is the PostScript glyph name and *groff-char* is a *groff* ordinary (if of unit length) or special (if longer) character identifier. The same *ps-glyph* can occur multiple times in the file; each *groff-char* must occur at most once. Lines starting with "#" and blank lines are ignored. If the file isn't found in the current directory, it is sought in the *devps/generate* subdirectory of the default font directory.

If a PostScript glyph is not mentioned in *map-file*, and a *groff* character name can't be deduced using the Adobe Glyph List (AGL, built into *afmtodit*), then *afmtodit* puts the PostScript glyph into the *groff* font description file as an unnamed glyph which can only be accessed by the "\N" escape sequence in a *roff* document. In particular, this is true for glyph variants named in the form "*foo.bar*"; all glyph names containing one or more periods are mapped to unnamed entities. Unless **–e** is specified, the encoding defined in the AFM file (i.e., entries with non-negative codes) is used. Refer to section "Using Symbols" in *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual, or *groff_char*(7), which describe how *groff* character identifiers are constructed.

Glyphs not encoded in the AFM file (i.e., entries indexed as "–1") are still available in *groff*; they get glyph index values greater than 255 (or greater than the biggest code used in the AFM file in the unlikely case that it is greater than 255) in the *groff* font description file. Unencoded glyph indices don't have a specific order; it is best to access them only via special character identifiers.

If the font file proper (not just its metrics) is available, listing it in the files */usr/share/groff/1.24.2/font/devps/download* and */usr/share/groff/1.24.2/font/devpdf/download* enables it to be embedded in the output produced by *grops*(1) and *gropdf*(1), respectively.

If the **–i** option is used, *afmtodit* automatically generates an italic correction, a left italic correction, and a subscript correction for each glyph (the significance of these is explained in *groff_font*(5)); they can be specified for individual glyphs by adding to the *afm-file* lines of the form:

```
italicCorrection ps-glyph n
leftItalicCorrection ps-glyph n
subscriptCorrection ps-glyph n
```

where *ps-glyph* is the PostScript glyph name, and *n* is the desired value of the corresponding parameter in thousandths of an em. Such parameters are normally needed only for italic (or oblique) fonts.

The **–s** option should be given if the font is "special", meaning that *groff* should search it whenever a glyph is not found in the current font. To enable this search, *font-description-file* can be listed as an argument to the **fonts** directive in the output device's *DESC* file; alternatively, a document can designate it with the **special** request. If the font is not special, there is no need to do either, since *troff*(1) will automatically mount it when it is first used.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−a *slant*
: Use *slant* as the slant ("angle") parameter in the font description file; this is used by *groff* in the positioning of accents. By default *afmtodit* uses the negative of the **ItalicAngle** specified in the AFM file; with true italic fonts it is sometimes desirable to use a slant that is less than this. If you find that an italic font places accents over base glyphs too far to the right, use **−a** to give it a smaller slant.

−c
: Include comments in the font description file identifying the PostScript font.

−d *device-description-file*
: The device description file is *desc-file* rather than the default *DESC*. If not found in the current directory, the *devps* subdirectory of the default font directory is searched (this is true for both the default device description file and a file given with option **−d**).

−e *encoding-file*
: The PostScript font should be reencoded to use the encoding described in *enc-file*. The format of *enc-file* is described in *grops*(1). If not found in the current directory, the *devps* subdirectory of the default font directory is searched.

−f *internal-name*
: The internal name of the *groff* font is set to *name*.

−i *italic-correction-factor*
: Generate an italic correction for each glyph so that its width plus its italic correction is equal to *italic-correction-factor* thousandths of an em plus the amount by which the right edge of the glyph's bounding box is to the right of its origin. If this would result in a negative italic correction, use a zero italic correction instead.

  Also generate a subscript correction equal to the product of the tangent of the slant of the font and four fifths of the x-height of the font. If this would result in a subscript correction greater than the italic correction, use a subscript correction equal to the italic correction instead.

  Also generate a left italic correction for each glyph equal to *italic-correction-factor* thousandths of an em plus the amount by which the left edge of the glyph's bounding box is to the left of its origin. The left italic correction may be negative unless option **−m** is given.

  This option is normally needed only with italic (or oblique) fonts. The font description files distributed with *groff* were created using an option of **−i50** for italic fonts.

−o *output-file*
: Write to *output-file* instead of *font-description-file.*

−k
: Omit any kerning data from the *groff* font; use only for monospaced (constant-width) fonts.

−m
: Prevent negative left italic correction values. Font description files for roman styles distributed with *groff* were created with "**−i0 −m**" to improve spacing with *eqn*(1).

−n
: Don't output a "**ligatures**" command for this font; use with monospaced (constant-width) fonts.

−q
: Quieten duplicate mapping warnings; see section "Diagnostics" below.

−s
: Add the "**special**" directive to the font description file.

−w *space-width*
: Use *space-width* as the width of inter-word spaces.

−x
: Don't use the built-in Adobe Glyph List.

Exit status

afmtodit exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Files

/usr/share/groff/1.24.2/font/devps/DESC
: describes the **ps** output device.

*/usr/share/groff/1.24.2/font/devps/*F
: describes the font known as *F* on device **ps**.

/usr/share/groff/1.24.2/font/devps/download
: lists fonts available for embedding within the PostScript document (or download to the device).

/usr/share/groff/1.24.2/font/devps/generate/dingbats.map
/usr/share/groff/1.24.2/font/devps/generate/dingbats−reversed.map
/usr/share/groff/1.24.2/font/devps/generate/slanted−symbol.map
/usr/share/groff/1.24.2/font/devps/generate/symbol.map
/usr/share/groff/1.24.2/font/devps/generate/text.map
: map names in the Adobe Glyph List to *groff* special character identifiers for Zapf Dingbats (**ZD**), reversed Zapf Dingbats (**ZDR**), slanted symbol (**SS**), symbol (**S**), and text fonts, respectively. These *map-file*s produce the font description files provided with *groff* for the *grops* output driver.

Diagnostics

AGL name '*x*' already mapped to groff name ' *y*'; ignoring AGL name 'uni*XXXX*'
: You can disregard these if they're in the form shown, where the ignored AGL name contains four hexadecimal digits *XXXX*. The Adobe Glyph List (AGL) has its own names for glyphs; they are often different from *groff*'s special character names. *afmtodit* is constructing a mapping from *groff* special character names to AGL names; this can be a one-to-one or many-to-one mapping, but one-to-many will not work, so *afmtodit* discards the excess mappings. For example, if *x* is **Delta**, *y* is ***D**, and *XXXX* is **0394**, *afmtodit* is telling you that the *groff* font description that it is writing cannot map the *groff* special character **\[*D]** to AGL glyphs **Delta** and **uni0394** at the same time.

  If you get a message like this but are unhappy with which mapping is ignored, a remedy is to craft an alternative *map-file* and re-run *afmtodit* using it.

  When the **−q** option is used, these messages are suppressed in favor of a count of how many would have been emitted were the option not present.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. Section "Using Symbols" may be of particular note. You can browse it interactively with "info '(groff)Using Symbols'".

groff(1), *gropdf*(1), *grops*(1), *groff_font*(5)

Name

chem – embed chemical structure diagrams in *groff* documents

Synopsis

chem [**−−**] [*file* . . .]

chem −h
chem −−help

chem −v
chem −−version

Description

chem produces chemical structure diagrams. Today's version is best suited for organic chemistry (bonds, rings). The *chem* program is a *groff* preprocessor like *eqn*, *pic*, *tbl*, etc. It generates *pic* output such that all *chem* parts are translated into diagrams of the *pic* language.

If no operands are given, or if *file* is "**−**", *chem* reads the standard input stream. **−h** and **−−help** display a usage message, whereas **−v** and **−−version** display version information; all exit.

The program *chem* originates from the Perl source file *chem.pl*. It tells *pic* to include a copy of the macro file *chem.pic*. Moreover the *groff* source file *pic.tmac* is loaded.

In a style reminiscent of *eqn* and *pic*, the *chem* diagrams are written in a special language.

A set of *chem* lines looks like this

```
.cstart
chem data
.cend
```

Lines containing the keywords **.cstart** and **.cend** start and end the input for *chem*, respectively. In *pic* context, i.e., after the call of **.PS**, *chem* input can optionally be started by the line **begin chem** and ended by the line with the single word **end** instead.

Anything outside these initialization lines is copied through without modification; all data between the initialization lines is converted into *pic* commands to draw the diagram.

As an example,

```
.cstart
CH3
bond
CH3
.cend
```

prints two **CH3** groups with a bond between them.

If you want to create just *groff* output, you must run *chem* followed by *groff* with the option **−p** for the activation of *pic*:

chem [*file* . . .] **| groff −p** . . .

chem requires the *Math::Trig* Perl module, available via CPAN.

Language

The *chem* input language is rather small. It provides rings of several styles and a way to glue them together as desired, bonds of several styles, moieties (e.g., **C**, **NH3**, . . ., and strings.

Setting variables

There are some variables that can be set by commands. Such commands have two possible forms, either

variable value

or

variable **=** *value*

This sets the given *variable* to the argument *value*. If more arguments are given only the last argument is taken, all other arguments are ignored.

There are only a few variables to be set by these commands:

textht *arg*
Set the height of the text to *arg*; default is 0.16.

cwid *arg*
Set the character width to *arg*; default is 0.12.

db *arg*
Set the bond length to *arg*; default is 0.2.

size *arg*
Scale the diagram to make it look plausible at point size *arg*; default is 10 point.

Bonds

bond [*direction*] [*length n*] [**from** *Name*|*picstuff*]

The **bond** command draws a single bond in direction from nearest corner of *Name*. **bond** can also be **double bond**, **front bond**, **back bond**, etc. (We will get back to *Name* soon.)

direction is the angle in degrees (0 up, positive clockwise) or a direction word like **up**, **down**, **sw** (= southwest), etc. If no direction is specified, the bond goes in the current direction (usually that of the last bond).

Normally the bond begins at the last object placed; this can be changed by naming a **from** place. For instance, to make a simple alkyl chain:

CH3	
bond	(this one goes right from the CH3)
C	(at the right end of the bond)
double bond up	(from the C)
O	(at the end of the double bond)
bond right from C	
CH3	

A length in inches may be specified to override the default length. Other *pic* commands can be tacked on to the end of a bond command, to created dotted or dashed bonds or to specify a **to** place.

Rings

There are lots of rings, but only five- and six-sided rings get much support. **ring** by itself is a six-sided ring; **benzene** is the benzene ring with a circle inside. **aromatic** puts a circle into any kind of ring.

ring [**pointing** (**up**|**right**|**left**|**down**)] [**aromatic**] [**put Mol at** *n*] [**double** *i*,*j k*,*l* . . . [*picstuff*]

The vertices of a ring are numbered 1, 2, . . . from the vertex that points in the natural compass direction. So for a hexagonal ring with the point at the top, the top vertex is 1, while if the ring has a point at the east side, that is vertex 1. This is expressed as

```
R1: ring pointing up
R2: ring pointing right
```

The ring vertices are named **.V1**, . . ., **.V***n*, with **.V1** in the pointing direction. So the corners of **R1** are **R1.V1** (the *top*), **R1.V2**, **R1.V3**, **R1.V4** (the *bottom*), etc., whereas for **R2**, **R2.V1** is the rightmost vertex and **R2.V4** the leftmost. These vertex names are used for connecting bonds or other rings. For example,

```
R1: benzene pointing right
R2: benzene pointing right with .V6 at R1.V2
```

creates two benzene rings connected along a side.

Interior double bonds are specified as **double** *n1*,*n2 n3*,*n4* . . .**;** each number pair adds an interior bond. So the

alternate form of a benzene ring is

ring double 1,2 3,4 5,6

Heterocycles (rings with something other than carbon at a vertex) are written as **put** *X* **at** *V*, as in

R: ring put N at 1 put O at 2

In this heterocycle, **R.N** and **R.O** become synonyms for **R.V1** and **R.V2**.

There are two five-sided rings. **ring5** is pentagonal with a side that matches the six-sided ring; it has four natural directions. A **flatring** is a five-sided ring created by chopping one corner of a six-sided ring so that it exactly matches the six-sided rings.

The description of a ring has to fit on a single line.

Moieties and strings

A moiety is a string of characters beginning with a capital letter, such as N(C2H5)2. Numbers are converted to subscripts (unless they appear to be fractional values, as in N2.5H). The name of a moiety is determined from the moiety after special characters have been stripped out: e.g., N(C2H5)2) has the name NC2H52.

Moieties can be specified in two kinds. Normally a moiety is placed right after the last thing mentioned, separated by a semicolon surrounded by spaces, e.g.,

B1: bond ; OH

Here the moiety is **OH**; it is set after a bond.

As the second kind a moiety can be positioned as the first word in a *pic*-like command, e.g.,

CH3 at C + (0.5,0.5)

Here the moiety is **CH3**. It is placed at a position relative to **C**, a moiety used earlier in the chemical structure.

So moiety names can be specified as *chem* positions everywhere in the *chem* code. Beneath their printing moieties are names for places.

The moiety **BP** is special. It is not printed but just serves as a mark to be referred to in later *chem* commands. For example,

bond ; BP

sets a mark at the end of the bond. This can be used then for specifying a place. The name **BP** is derived from *branch point* (i.e., line crossing).

A string within double quotes **"** is interpreted as a part of a *chem* command. It represents a string that should be printed (without the quotes). Text within quotes **"**. . . **"** is treated more or less like a moiety except that no changes are made to the quoted part.

Names

In the alkyl chain above, notice that the carbon atom **C** was used both to draw something and as the name for a place. A moiety always defines a name for a place; you can use your own names for places instead, and indeed, for rings you will have to. A name is just

*Name***:** . . .

Name is often the name of a moiety like **CH3**, but it need not to be. Any name that begins with a capital letter and which contains only letters and numbers is valid.

```
First: bond
       "bond 30 from First"
```

Miscellaneous

The specific construction

bond . . . **; moiety**

is equivalent to

```
bond
moiety
```

Otherwise, each item has to be on a separate line (and only one line). Note that there must be whitespace after the semicolon which separates the commands.

A period character **.** or a single quote **'** in the first column of a line signals a *troff* command, which is copied through as-is.

A line whose first non-blank character is a hash character (**#**) is treated as a comment and thus ignored. However, hash characters within a word are kept.

A line whose first word is **pic** is copied through as-is after the word **pic** has been removed.

The command

size *n*

scales the diagram to make it look plausible at point size *n* (default is 10 point).

Anything else is assumed to be *pic* code, which is copied through with a label.

Since *chem* is a *pic* preprocessor, it is possible to include *pic* statements in the middle of a diagram to draw things not provided for by *chem* itself. Such *pic* statements should be included in *chem* code by adding **pic** as the first word of this line for clarity.

The following *pic* commands are accepted as *chem* commands, so no **pic** command word is needed:

define Start the definition of *pic* macro within *chem*.

[Start a block composite.

] End a block composite.

{ Start a macro definition block.

} End a macro definition block.

The macro names from **define** statements are stored and their call is accepted as a *chem* command as well.

Wish list

This TODO list was collected by Brian Kernighan.

Error checking is minimal; errors are usually detected and reported in an oblique fashion by *pic*.

There is no library or file inclusion mechanism, and there is no shorthand for repetitive structures.

The extension mechanism is to create *pic* macros, but these are tricky to get right and don't have all the properties of built-in objects.

There is no in-line chemistry yet (e.g., analogous to the **$**. . .**$** construct of *eqn*).

There is no way to control entry point for bonds on groups. Normally a bond connects to the carbon atom if entering from the top or bottom and otherwise to the nearest corner.

Bonds from substituted atoms on heterocycles do not join at the proper place without adding a bit of *pic*.

There is no decent primitive for brackets.

Text (quoted strings) doesn't work very well.

A squiggle bond is needed.

Files

/usr/share/groff/1.24.2/pic/chem.pic
: A collection of *pic* macros needed by *chem*.

/usr/share/groff/1.24.2/tmac/pic.tmac
: A macro file which redefines **.PS**, **.PE**, and **.PF** to center *pic* diagrams.

/usr/share/doc/groff/examples/chem/.chem*
: Example files for *chem*.

/usr/share/doc/groff/examples/chem/122/.chem*
: Example files from the *chem* article by its authors, "CHEM—A Program for Typesetting Chemical Structure Diagrams: User Manual" (CSTR #122).

Authors

The GNU version of *chem* was written by Bernd Warken. It is based on the documentation of Brian Kernighan's original *awk* version of *chem*.

See also

"CHEM—A Program for Typesetting Chemical Diagrams: User Manual" by Jon L. Bentley, Lynn W. Jelinski, and Brian W. Kernighan, 1992, AT&T Bell Laboratories Computing Science Technical Report No. 122

groff (1), *pic*(1)

Name

eqn − format mathematics (equations) for *groff* or MathML

Synopsis

eqn [**−CNrR**] [**−**d *xy*] [**−f** *global-italic-font*] [**−m** *minimum-type-size*] [**−M** *eqnrc-directory*] [**−p** *super/subscript-size-reduction*] [**−s** *global-type-size*] [**−T** *device*] [*file* . . .]

eqn −−help

eqn −v
eqn −−version

Description

The GNU implementation of *eqn* is part of the *groff*(7) document formatting system. *eqn* is a *troff*(1) preprocessor that translates expressions in its own language, embedded in *roff*(7) input, into mathematical notation typeset by *troff*(1). It copies each *file*'s contents to the standard output stream, translating each *equation* between lines starting with **.EQ** and **.EN**, or within a pair of user-specified delimiters. Normally, *eqn* is not executed directly by the user, but invoked by specifying the **−e** option to *groff*(1). While GNU *eqn*'s input syntax is highly compatible with AT&T *eqn*, the output *eqn* produces cannot be processed by AT&T *troff*; GNU *troff* (or a *troff* implementing relevant GNU extensions) must be used. If no *file* operands are present, or if *file* is "**−**", *eqn* reads the standard input stream.

Unless the **−R** option is used, *eqn* searches for the file *eqnrc* in the directories given with the **−M** option first, then in */etc/groff/site−tmac*, and finally in the standard macro directory */usr/share/groff/1.24.2/tmac*. If it exists and is readable, *eqn* processes it before any input.

This man page primarily discusses the differences between GNU *eqn* and AT&T *eqn*. Most of the new features of the GNU *eqn* input language are based on TEX. There are some references to the differences between TEX and GNU *eqn* below; these may safely be ignored if you do not know TEX. Four points are worth note.

- GNU *eqn* emits Presentation MathML output when invoked with the "**−T MathML**" option.
- GNU *eqn* does not support terminal devices well, though it may suffice for simple inputs.
- GNU *eqn* sets the input token "**...**" as an ellipsis on the text baseline, not the three centered dots of AT&T *eqn*. Set an ellipsis on the math axis with the GNU extension macro **cdots**.
- GNU *eqn*'s **delim** primitive does not treat an "**on**" argument as a pair of equation delimiters.

Anatomy of an equation

eqn input consists of tokens. Consider a form of Newton's second law of motion for constant mass. The input

```
.EQ
F =
m a
.EN
```

becomes $F = ma$. Each of **F**, **=**, **m**, and **a** is a token. Spaces and newlines are interchangeable; they separate tokens but do not break lines or produce space in the output.

Beyond their primary functions, the following input characters separate tokens as well.

{ } Braces perform grouping. Whereas "**e sup a b**" expresses $e^a b$, "**e sup { a b }**" means e^{ab}. When immediately preceded by a "**left**" or "**right**" primitive, a brace loses its special meaning.

^ ~ are the *half space* and *full space,* respectively. Use them to tune the appearance of the output.

Tab and leader characters separate tokens as well as advancing the drawing position to the next tab stop, but are seldom used in *eqn* input. When they occur, they must appear at the outermost lexical scope. This roughly means that they can't appear within braces that are necessary to disambiguate the input; *eqn* will diagnose an error in this event. (See subsection "Macros" below for additional token separation rules.)

Other tokens are primitives, macros, an argument to either of the foregoing, or components of an equation.

Primitives are fundamental keywords of the *eqn* language. They can configure an aspect of the preprocessor's state, as when setting a "global" font selection or type size (**gifont** and **gsize**), or declaring or deleting macros ("**define**" and **undef**); these are termed *commands.* Other primitives perform formatting operations on the tokens around them (as with **fat**, **over**, **sqrt**, or **up**).

Equation *components* include mathematical variables, constants, numeric literals, and operators. *eqn* remaps some input character sequences to *groff* special character escape sequences for economy in equation entry and to ensure that glyphs from an unstyled font are used; see *groff_char*(7).

```
+   \[pl]                   '    \[fm]
-   \[mi]                   <=   \[<=]
=   \[eq]                   >=   \[>=]
```

Macros permit primitives, components, and other macros to be collected and used together as a single token. Predefined macros make convenient the preparation of *eqn* input in a form resembling its spoken expression; for example, consider **cos**, **hat**, **inf**, and **lim**.

Spacing and typeface

GNU *eqn* imputes a type to each equation component, adjusting the space around it accordingly. Recognized types follow; most affect spacing only, whereas the "**letter**" subtype of "**ordinary**" also assigns a style.

`ordinary`	character such as "1", "a", or "!"
`letter`	character to be italicized by default
`digit`	*n/a*
`operator`	large operator such as "$\sum$"
`binary`	binary operator such as "+"
`relation`	relational operator such as "="
`opening`	opening bracket such as "("
`closing`	closing bracket such as ")"
`punctuation`	punctuation character such as ","
`inner`	sub-formula contained within brackets
`suppress`	component to which automatic spacing is not applied

Two primitives apply types to equation components.

type *t e*
: Apply type *t* to expression *e*.

chartype *t text*
: Assign each character in (unquoted) *text* type *t*, persistently.

eqn sets up spacings and styles as if by the following commands.

```
chartype "letter"      abcdefghiklmnopqrstuvwxyz
chartype "letter"      ABCDEFGHIKLMNOPQRSTUVWXYZ
chartype "letter"      \[*a]\[*b]\[*g]\[*d]\[*e]\[*z]
chartype "letter"      \[*y]\[*h]\[*i]\[*k]\[*l]\[*m]
chartype "letter"      \[*n]\[*c]\[*o]\[*p]\[*r]\[*s]
chartype "letter"      \[*t]\[*u]\[*f]\[*x]\[*q]\[*w]
chartype "binary"      *\[pl]\[mi]
chartype "relation"    <>\[eq]\[<=]\[>=]
chartype "opening"     {([
chartype "closing"     })]
chartype "punctuation" ,;:.
chartype "suppress"    ^~
```

eqn assigns all other ordinary and special *roff* characters, including numerals 0–9, the "**ordinary**" type. (The "**digit**" type is not used, but is available for customization.) In keeping with common practice in mathematical typesetting, lowercase, but not uppercase, Greek letters are assigned the "**letter**" type to style them in italics. The macros for producing ellipses, "**...**", **cdots**, and **ldots**, use the "**inner**" type.

Primitives

eqn supports without alteration the AT&T *eqn* primitives **above**, **back**, **bar**, **bold**, **define**, **down**, **fat**, **font**, **from**, **fwd**, **gfont**, **gsize**, **italic**, **left**, **lineup**, **mark**, **matrix**, **ndefine**, **over**, **right**, **roman**, **size**, **sqrt**, **sub**, **sup**, **tdefine**, **to**, **under**, and **up**.

New primitives

We describe the GNU extension primitives "**type**" and **chartype** in subsection "Spacing and typeface" above; "**set**" and "**reset**" in subsection "Customization" below; and **gbfont**, **gifont**, and **grfont** in subsection "Fonts" below. In the following synopses, *X* can be any character not appearing in the parameter thus bracketed.

e1 **accent** *e2*

Set *e2* as an accent over *e1*. *eqn* assumes that *e2* is at the appropriate height for a lowercase letter without an ascender, and shifts it vertically based on *e1*'s height. For example, *eqn* defines **hat** as follows.

```
accent { roman "^" }
```

dotdot, **dot**, **tilde**, **vec**, and **dyad** are also defined using the **accent** primitive.

big *e* Enlarge the expression *e*; semantics like those of CSS "large" are intended. In *troff* output, the type size is increased by 5 scaled points. MathML output emits the following.

```
<mstyle mathsize='big'>
```

copy *file*
include *file*

Interpolate the contents of *file*, omitting lines beginning with **.EQ** or **.EN**. *file* is sought relative to the current working directory.

ifdef *name X anything X*

If *name* is defined as a primitive or macro, interpret *anything*.

nosplit *text*

As "*text*", but since *text* is not quoted it is subject to macro expansion; it is not split up and the spacing between characters not adjusted per subsection "Spacing and typeface" above.

e **opprime**

As **prime**, but set the prime symbol as an operator on *e*. In the input " **A opprime sub 1**", the "1" is tucked under the prime as a subscript to the "A" (as is conventional in mathematical typesetting), whereas when **prime** is used, the "1" is a subscript to the prime character. The precedence of **opprime** is the same as that of **bar** and "**under**", and higher than that of other primitives except **accent** and **uaccent**. In unquoted text, a neutral apostrophe (') that is not the first character on the input line is treated like **opprime**.

sdefine *name X anything X*

As "**define**", but *name* is not recognized as a macro if called with arguments.

e1 **smallover** *e2*

As **over**, but reduce the type size of *e1* and *e2*, and put less vertical space between *e1* and *e2* and the fraction bar. The **over** primitive corresponds to the TEX **\over** primitive in displayed equation styles; **smallover** corresponds to **\over** in non-display ("inline") styles.

space *n*

Set extra vertical spacing around the equation, replacing the default values, where *n* is an integer in hundredths of an em. If positive, *n* increases vertical spacing before the equation; if negative, it does so after the equation. This primitive provides an interface to *groff*'s **\x** escape sequence, but with the opposite sign convention. It has no effect if the equation is part of a *pic*(1) picture.

special *troff-macro e*

Construct an object by calling *troff-macro* on *e*. The *troff* string **0s** contains the *eqn* output for *e*, and the registers **0w**, **0h**, **0d**, **0skern**, and **0skew** the width, height, depth, subscript kern, and skew of *e*, respectively. (The *subscript kern* of an object indicates how much a subscript on that object should be "tucked in", or placed to the left relative to a non-subscripted glyph of the same size. The *skew* of an object is how far to the right of the center of the object an accent over it should be placed.) The macro must modify **0s** so that it outputs the desired result, returns the drawing position to the text baseline at the beginning of *e*, and updates the foregoing registers to correspond to the new dimensions of the result.

Suppose you want a construct that "cancels" an expression by drawing a diagonal line through it.

```
.de Ca
.  ds 0s \
\Z'\\*(0s'\
\v'\\n(0du'\
\D'l \\n(0wu -\\n(0hu-\\n(0du'\
\v'\\n(0hu'
..
.EQ
special Ca "x \[mi] 3 \[pl] x" ~ 3
.EN
```

We use the **\[mi]** and **\[pl]** special characters instead of + and − because they are part of the argument to a *troff* macro, so *eqn* does not transform them to mathematical glyphs for us. Here's a more complicated construct that draws a box around an expression; the bottom of the box rests on the text baseline. We define the *eqn* macro **box** to wrap the call of the *troff* macro **Bx**.

```
.de Bx
.ds 0s \
\Z'\\h'1n'\\*[0s]'\
\v'\\n(0du+1n'\
\D'l \\n(0wu+2n 0'\
\D'l 0 -\\n(0hu-\\n(0du-2n'\
\D'l -\\n(0wu-2n 0'\
\D'l 0 \\n(0hu+\\n(0du+2n'\
\h'\\n(0wu+2n'
.nr 0w +2n
.nr 0d +1n
.nr 0h +1n
..
.EQ
define box ' special Bx $1 '
box(foo) ~ "bar"
.EN
```

split "*text***"**

As *text*, but since *text* is quoted, it is not subject to macro expansion; it is split up and the spacing between characters adjusted per subsection "Spacing and typeface" above.

e1 **uaccent** *e2*

Set *e2* as an accent under *e1*. *e2* is assumed to be at the appropriate height for a letter without a descender; *eqn* vertically shifts it depending on whether *e1* has a descender. **utilde** is predefined using **uaccent** as a tilde accent below the baseline.

undef *name*
: Remove definition of macro or primitive *name*, making it undefined.

vcenter *e*
: Vertically center *e* about the *math axis*, a horizontal line upon which fraction bars and characters such as "+" and " −" are aligned. MathML already behaves this way, so *eqn* ignores this primitive when producing that output format. The built-in **sum** macro is defined as if by the following.

```
define sum ! { type "operator" vcenter size +5 \(*S } !
```

Extended primitives

GNU *eqn* extends the syntax of some AT&T *eqn* primitives, introducing one deliberate incompatibility.

delim on
: GNU *eqn* recognizes an "**on**" argument to the **delim** primitive specially, restoring any delimiters previously disabled with "**delim off**". If delimiters haven't been specified, neither command has effect. Few *eqn* documents are expected to use "o" and "n" as left and right delimiters, respectively. If yours does, swap them, or select others.

col *n* { ... }
ccol *n* { ... }
lcol *n* { ... }
rcol *n* { ... }
pile *n* { ... }
cpile *n* { ... }
lpile *n* { ... }
rpile *n* { ... }
: The integer value *n*, in hundredths of an em, uses the formatter's **\x** escape sequence to increase the vertical spacing between rows; *eqn* ignores it when producing MathML. Negative values are accepted but have no effect. If more than one *n* occurs in a matrix or pile, the largest is used.

Customization

When *eqn* generates *troff* input, the appearance of equations is controlled by a large number of parameters. They have no effect when generating MathML, which delegates typesetting to a MathML rendering engine. Configure these parameters with the "**set**" and "**reset**" primitives.

set *p n*
: assigns parameter *p* the integer value *n*; *n* is interpreted in units of hundredths of an em unless otherwise stated. For example,

```
set x_height 45
```

says that *eqn* should assume that the font's x-height is 0.45 ems.

reset *p*
: restores the default value of parameter *p*.

Available parameters *p* are as follows; defaults are shown in parentheses. We intend these descriptions to be expository rather than rigorous.

minimum_size
: sets a floor for the type size (in scaled points) at which equations are set (**5**).

fat_offset
: The **fat** primitive emboldens an equation by overprinting two copies of the equation horizontally offset by this amount (**4**). When producing MathML output, components to which **fat_offset** applies instead use the following.

```
<mstyle mathvariant='double-struck'>
```

over_hang
: A fraction bar is longer by twice this amount than the maximum of the widths of the numerator and denominator; in other words, it overhangs the numerator and denominator by at least this amount (**0**).

accent_width
: When **bar** or **under** is applied to a single character, the line is this long (**31**). Normally, **bar** or **under** produces a line whose length is the width of the object to which it applies; in the case of a single character, this tends to produce a line that looks too long.

delimiter_factor Extensible delimiters produced with the **left** and **right** primitives have a combined height and depth of at least this many thousandths of twice the maximum amount by which the sub-equation that the delimiters enclose extends away from the axis (**900**).

delimiter_shortfall Extensible delimiters produced with the **left** and **right** primitives have a combined height and depth not less than the difference of twice the maximum amount by which the sub-equation that the delimiters enclose extends away from the axis and this amount (**50**).

null_delimiter_space This much horizontal space is inserted on each side of a fraction (**12**).

script_space The width of subscripts and superscripts is increased by this amount (**5**).

thin_space This amount of space is automatically inserted after punctuation characters (**17**).

medium_space This amount of space is automatically inserted on either side of binary operators (**22**).

thick_space This amount of space is automatically inserted on either side of relations (**28**).

half_space configures the width of the space produced by the **^** token (**17**).

full_space configures the width of the space produced by the **~** token (**28**).

x_height The height of lowercase letters without ascenders such as "x" (**45**).

axis_height The height above the baseline of the center of characters such as "+" and " –" (**26**). It is important that this value is correct for the font you are using.

default_rule_thickness This should be set to the thickness of the **\[ru]** character, or the thickness of horizontal lines produced with the **\D** escape sequence (**4**).

num1 The **over** primitive shifts up the numerator by at least this amount (**70**).

num2 The **smallover** primitive shifts up the numerator by at least this amount (**36**).

denom1 The **over** primitive shifts down the denominator by at least this amount (**70**).

denom2 The **smallover** primitive shifts down the denominator by at least this amount (**36**).

sup1 Normally superscripts are shifted up by at least this amount (**42**).

sup2 Superscripts within superscripts or upper limits or numerators of **smallover** fractions are shifted up by at least this amount (**37**). Conventionally, this is less than **sup1**.

sup3 Superscripts within denominators or square roots or subscripts or lower limits are shifted up by at least this amount (**28**). Conventionally, this is less than **sup2**.

sub1 Subscripts are normally shifted down by at least this amount (**20**).

sub2 When there is both a subscript and a superscript, the subscript is shifted down by at least this amount (**23**).

sup_drop The baseline of a superscript is no more than this much below the top of the object on which the superscript is set (**38**).

sub_drop The baseline of a subscript is at least this much below the bottom of the object on which the subscript is set (**5**).

big_op_spacing1 The baseline of an upper limit is at least this much above the top of the object on which the limit is set (**11**).

big_op_spacing2 The baseline of a lower limit is at least this much below the bottom of the object on which the limit is set (**17**).

big_op_spacing3 The bottom of an upper limit is at least this much above the top of the object on which the limit is set (**20**).

big_op_spacing4 The top of a lower limit is at least this much below the bottom of the object on which the limit is set (**60**).

big_op_spacing5 This much vertical space is added above and below limits (**10**).

baseline_sep The baselines of the rows in a pile or matrix are normally this far apart (**140**). Usually equal to the sum of **num1** and **denom1**.

shift_down The midpoint between the top baseline and the bottom baseline in a matrix or pile is shifted down by this much from the axis (**26**). Usually equal to **axis_height**.

column_sep This much space is added between columns in a matrix (**100**).

matrix_side_sep This much space is added at each side of a matrix (**17**).

draw_lines If non-zero, *eqn* draws lines using the *troff* **\D** escape sequence, rather than the **\l** escape sequence and the **\[ru]** special character. The *eqnrc* file sets the default: **1** on **ps**, **html**, and the X11 devices, otherwise **0**.

body_height is the presumed height of an equation above the text baseline; *eqn* adds any excess as extra pre-vertical line spacing with *troff*'s **\x** escape sequence (**85**).

body_depth is the presumed depth of an equation below the text baseline; *eqn* adds any excess as extra post-vertical line spacing with *troff*'s **\x** escape sequence (**35**).

nroff If non-zero, then **ndefine** behaves like **define** and **tdefine** is ignored, otherwise **tdefine** behaves like **define** and **ndefine** is ignored. The *eqnrc* file sets the default: **1** on **ascii**, **latin1**, and **utf8** devices, otherwise **0**.

Macros

In GNU *eqn*, macros can take arguments. A word defined by any of the **define**, **ndefine**, or **tdefine** primitives followed immediately by a left parenthesis is treated as a *parameterized macro call:* subsequent tokens up to a matching right parenthesis are treated as comma-separated arguments. In this context only, commas and parentheses also serve as token separators. A macro argument is not terminated by a comma inside parentheses nested within it. In a macro definition, **$***n*, where *n* is between 1 and 9 inclusive, is replaced by the *n*th argument; if there are fewer than *n* arguments, it is replaced by nothing.

Predefined macros

GNU *eqn* supports the predefined macros offered by AT&T *eqn*: **and**, **approx**, **arc**, **cos**, **cosh**, **del**, **det**, **dot**, **dotdot**, **dyad**, **exp**, **for**, **grad**, **half**, **hat**, **if**, **inter**, **Im**, **inf**, **int**, **lim**, **ln**, **log**, **max**, **min**, **nothing**, **partial**, **prime**, **prod**, **Re**, **sin**, **sinh**, **sum**, **tan**, **tanh**, **tilde**, **times**, **union**, **vec**, **==**, **!=**, **+=**, **–>**, **<–**, **<<**, **>>**, and "**...**". The lowercase classical Greek letters are available as **alpha**, **beta**, **chi**, **delta**, **epsilon**, **eta**, **gamma**, **iota**, **kappa**, **lambda**, **mu**, **nu**, **omega**, **omicron**, **phi**, **pi**, **psi**, **rho**, **sigma**, **tau**, **theta**, **upsilon**, **xi**, and **zeta**. Spell them with an initial capital letter (**Alpha**) or in full capitals (**ALPHA**) to obtain uppercase forms.

GNU *eqn* further defines the macros **cdot**, **cdots**, and **utilde** (all discussed above), **dollar**, which sets a dollar sign, and **ldots**, which sets an ellipsis on the text baseline.

Fonts

eqn uses up to three typefaces to set an equation: italic (oblique), roman (upright), and bold (heavy). Assign each a *groff* typeface with the GNU extension primitives **grfont**, **gifont**, and **gbfont.** The defaults are the styles **R**, **I**, and **B** (applied to the current font family). The **chartype** primitive (see above) sets a character's type, which determines the face used to set it. The "**letter**" type is set in italics; others are set in roman. Use the **bold** primitive to select an (upright) bold style.

gbfont *f*
: Select *f* as the bold font.

gifont *f*
: Select *f* as the italic font. GNU *eqn* recognizes **gfont** as a synonym for AT&T compatibility.

grfont *f*
: Select *f* as the roman font.

roff interface

The string **10** stores the most recently formatted equation. The register **MK** stores the horizontal drawing position of the most recently assigned mark. Avoid using these names for any other purpose.

GNU *eqn* internally employs register, string, macro, and diversion names beginning with the numeral **0**. (AT&T *eqn* used identifiers starting with the numeral **1**.) A document to be preprocessed with GNU *eqn* should not use any such identifiers.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−C Recognize **.EQ** and **.EN** even when followed by a character other than space or newline.

−d *xy* Specify delimiters *x* for left and *y* for right ends of equations not bracketed by **.EQ/.EN**. *x* and *y* need not be distinct. Any "**delim** *xy*" statements in the source file override this option.

−f *F* is equivalent to "**gifont** *F*".

−m *n* is equivalent to "**set minimum_size** *n*".

−M *dir* Search *dir* for *eqnrc* before those listed in section "Description" above.

−N Prohibit newlines within delimiters, allowing *eqn* to recover better from missing closing delimiters.

−p *n* Set sub- and superscripts *n* points smaller than the surrounding text. This option is deprecated. *eqn* normally sets sub- and superscripts at 70% of the type size of the surrounding text.

−r Reduce the type size of super- and subscripts at most once relative to the base type size.

−R Don't load *eqnrc*.

−s *n* is equivalent to "**gsize** *n*". This option is deprecated.

−T *dev* Prepare output for the device *dev*. This option defines a macro *dev* with the value **1**; *eqnrc* thereby provides definitions appropriate to the device. However, if *dev* is "**MathML**", *eqn* produces output in that language rather than *roff*, and does not load *eqnrc*. The default device is **ps**.

Exit status

eqn exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Files

/usr/share/groff/1.24.2/tmac/eqnrc
: initializes the preprocessor for *troff* output. Any valid *eqn* input is accepted.

MathML output limitations

MathML's design assumes that it cannot know the exact physical characteristics of the media and devices on which it will be rendered. It does not support control of motions and sizes to the same degree *troff* does.

- GNU *eqn*'s rendering parameters (see section "Customization" above) have no effect on generated MathML.
- The **special**, **up**, **down**, **fwd**, and **back** primitives cannot be implemented, and yield a MathML "<merror>" message instead.

- The **vcenter** primitive is silently ignored, as centering on the math axis is the MathML default.
- Characters that *eqn* sets extra large in *troff* output—notably the integral sign—may appear too small and need to have their "<mstyle>" wrappers adjusted by hand.

As when producing *troff* output, *eqn* generates MathML that leaves the **.EQ** and **.EN** tokens in place, but emits nothing corresponding to **delim** delimiters. They can, however, be recognized as character sequences that begin with "$", end with "$", and do not cross line boundaries.

Caveats

Tokens must be double-quoted in *eqn* input if they are not to be recognized as names of macros or primitives, or if they are to be interpreted by *troff*. In particular, short ones, like " **pi**" and "**PI**", can collide with *troff* identifiers. For instance, the *eqn* command "**gifont PI**" does not select the *gropdf*(1) or *grops*(1) device's Palatino italic font as the global italic face; you must use "**gifont "PI"**" instead.

Delimited equations are set at the type size current at the beginning of the input line, not necessarily that immediately preceding the opening delimiter.

Unlike TEX, *eqn* does not inherently distinguish displayed and inline equation styles; see the **smallover** primitive above. However, macro packages frequently define **EQ** and **EN** macros such that the equation within is displayed. These macros may accept arguments permitting the equation to be labeled or captioned; see the package's documentation.

Bugs

eqn abuses terminology—its "equations" can be inequalities, bare expressions, or unintelligible gibberish. But there's no changing it now.

When producing terminal output, GNU *eqn* renders lowercase Greek letters in roman instead of italic style.

When producing MathML output, the **mark** and **lineup** features don't work. These could, in theory, be implemented with "<maligngroup>" elements.

When producing MathML output, each digit of a numeric literal gets a separate "<mn></mn>" pair, and decimal points are tagged with "<mo></mo>". This is allowed by the specification, but inefficient.

Examples

We first illustrate *eqn* usage with a trigonometric identity.

```
.EQ
sin ( alpha + beta ) = sin alpha cos beta + cos alpha sin beta
.EN
```

$$\sin(\alpha + \beta) = \sin\alpha \cos\beta + \cos\alpha \sin\beta$$

It can be convenient to set up delimiters if mathematical content will appear frequently in running text.

```
.EQ
delim $$
.EN
Having cached a table of logarithms,
the property $ln ( x y ) = ln x + ln y$ sped calculations.
```

Having cached a table of logarithms, the property $\ln(xy) = \ln x + \ln y$ sped calculations.

The quadratic formula affords an opportunity to use fractions, radicals, and the full space token **~**.

```
.EQ
x = { - b ~ \[+-] ~ sqrt { b sup 2 - 4 a c } } over { 2 a }
.EN
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Alternatively, we could define the plus-minus sign as a binary operator. Automatic spacing puts 0.06 em less

space on either side of the plus-minus than ~ does, this being the difference between the widths of the **medium_space** parameter used by binary operators and that of the full space. Independently, we can define a macro "frac" for setting fractions.

```
.EQ
chartype "binary" \[+-]
define frac ! { $1 } over { $2 } !
x = frac(- b \[+-] sqrt { b sup 2 - 4 a c }, 2 a)
.EN
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

See also

"Typesetting Mathematics—User's Guide" (2nd edition), by Brian W. Kernighan and Lorinda L. Cherry, 1978, AT&T Bell Laboratories Computing Science Technical Report No. 17.

The TEXbook, by Donald E. Knuth, 1984, Addison-Wesley Professional. Appendix G discusses many of the parameters from section "Customization" above in greater detail.

groff_char(7) documents a variety of special character escape sequences useful in mathematical typesetting. See subsections "Logical symbols", "Mathematical symbols", and "Greek glyphs" in particular.

groff(1), *troff*(1), *pic*(1), *groff_font*(5)

Name

eqn2graph – convert an *eqn* equation into a cropped image

Synopsis

eqn2graph [**–format** *output-format*] [*convert-argument* ...]

eqn2graph ––help

eqn2graph –v
eqn2graph ––version

Description

eqn2graph reads a one-line *eqn*(1) equation from the standard input and writes an image file, by default in Portable Network Graphics (PNG) format, to the standard output.

The input EQN code should *not* be preceded by the **.EQ** macro that normally precedes it within *groff*(1) macros; nor do you need to have dollar-sign or other delimiters around the equation.

Arguments not recognized by *eqn2graph* are passed to the ImageMagick or GraphicsMagick program *convert*(1). By specifying these, you can give your image a border, set the image's pixel density, or perform other useful transformations.

The output image is clipped using *convert*'s **–trim** option to the smallest possible bounding box that contains all the black pixels.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–format *output-format*
: Write the image in *output-format*, which must be understood by *convert*; the default is PNG.

Environment

GROFF_TMPDIR
TMPDIR
TMP
TEMP
: These environment variables are searched in the order shown to determine the directory where temporary files will be created. If none are set, */tmp* is used.

Authors

eqn2graph was written by Eric S. Raymond, based on a recipe for *pic2graph*(1), by W. Richard Stevens.

See also

pic2graph(1), *grap2graph*(1), *eqn*(1), *groff*(1), *convert*(1)

Name

gdiffmk – mark differences between *groff*/*nroff*/*troff* files

Synopsis

gdiffmk [**−a** *add-mark*] [**−c** *change-mark*] [**−d** *delete-mark*] [**−x** *diff-command*] [**−D** [**−B**] [**−M** *mark1 mark2*]] [**−−**] *file1* *file2* [*output-file*]

gdiffmk −−help

gdiffmk −−version

Description

gdiffmk compares two *roff*(7) documents, *file1* and *file2*, and writes another, derived from both, to the standard output stream (or *output-file*), adding margin character (**mc**) requests at places in the output where the input documents differ. *gdiffmk* does not interpret the source documents; it treats *roff* comments and nilpotent changes to formatting as meaningful. For example, it does not know that **\h'3m'**, **\h'(1 * 3m)'**, and **\h'3m+0'** all mean the same thing. If the *file1* or *file2* argument is "**−**", *gdiffmk* reads the standard input stream for that input. If the *output* operand is present, *gdiffmk* writes output to a file of that name. If it is "**−**" or absent, *gdiffmk* writes output to the standard output stream. "**−**" cannot be both an input and output operand.

Options

−−help displays a usage message and **−−version** shows version information; both exit afterward.

−a *add-mark*
: Annotate material absent from *file1* but present in *file2* with *add-mark* (default: "**+**").

−B
: Suppress **br** requests normally inserted by **−D** option. Use with caution; such requests are the only way to guarantee that deletions and small changes are marked.

−c *change-mark*
: Annotate material differing between *file1* and *file2* with *change-mark* (default: "**|**").

−d *delete-mark*
: Annotate material present in *file1* but absent from *file2* with *delete-mark* (default: "*****").

−D
: Mark changed and deleted material with surrounding delimiters.

−M *mark1 mark2*
: Use *mark1* (default: "**[[**") and *mark2* (default: "**]]**") as delimiters when using the **−D** option.

−x *diff-command*
: Use the *diff-command* program to perform the comparison of *file1* and *file2*. *diff-command* (default: **diff**) must accept GNU *diff*(1)'s **−D** extension option.

−−
: Treat all subsequent arguments as file names, even if they begin with "**−**".

Exit status

gdiffmk exits with status **0** if the input files are the same; **1** if they differ; **2** upon a usage error; **3** if the system's *diff*(1) or *sh*(1) commands do not support features *gdiffmk* requires; and **4** if the *output* argument is a duplicate of *file1* or *file2*.

Caveats

The output is not necessarily compatible with all macro packages or preprocessors. A reliable workaround is to run *gdiffmk* on the output of the final preprocessor instead of the input source.

gdiffmk relies on the **−D** option of GNU *diff* to make a merged "#ifdef" output format. Busybox *diff* is known to not support it. Also see the **−x** *diff-command* option.

Authors

gdiffmk was written by Mike Bianchi, now retired. It is maintained by the *groff* developers.

See also

groff(1), *nroff*(1), *troff*(1), *roff*(7), *diff*(1)

Name

glilypond – embed LilyPond musical notation in *groff* documents

Synopsis

glilypond [**–k**] [{**––ly2eps**|**––pdf2eps**}] [**–e** *directory*] [**–o** *output-file*] [**–p** *filename-prefix*] [**–t** *tdir*] [{**–v**|**–V**}] [**––**] [*file* . . .]
glilypond [{**––ly2eps**|**––pdf2eps**}] [**––eps_dir** *directory*] [**––keep_all**] [**––output** *output-file*] [**––prefix** *filename-prefix*] [**––temp_dir** *tdir*] [**––verbose**] [**––**] [*file* . . .]

glilypond –?
glilypond –h
glilypond ––help
glilypond ––usage

glilypond –l
glilypond ––license

glilypond ––version

Description

glilypond is a *groff*(7) preprocessor that enables the embedding of LilyPond music scores in *groff* documents. If no operands are given, or if *file* is "**–**", *glilypond* reads the standard input stream. A double-dash argument ("**––**") causes all subsequent arguments to be interpreted as *file* operands, even if their names start with a dash.

glilypond requires the *File::HomeDir* Perl module, available via CPAN.

Usage

At present, *glilypond* works with the *groff* **ps**, **dvi**, **html**, and **xhtml** devices. The **lbp** and **lj4** devices are untested. Unfortunately, the **pdf** device does not yet work.

Option overview

–?|**–h**|**––help**|**––usage**
: Display usage information and exit.

––version
: Display version information and exit.

–l|**––license**
: Display copyright license information and exit.

Options for building EPS files

––ly2eps
: Direct *lilypond*(1) to create Encapsulated PostScript (EPS) files. This is the default.

––pdf2eps
: The program *glilypond* generates a PDF file using *lilypond*. Then the EPS file is generated by *pdf2ps* and *ps2eps*.

Directories and files

–e|**––eps_dir** *directory_name*
: Normally all *EPS* files are sent to the temporary directory. With this option, you can generate your own directory, in which all useful *EPS* files are send. So at last, the temporary directory can be removed.

–p|**––prefix** *begin_of_name*
: Normally all temporary files get names that start with the **ly**. . . prefix. With this option, you can freely change this prefix.

–k|**––keep_all**
: Normally all temporary files without the *eps* files are deleted. With this option, all generated files either by the *lilypond* program or other format transposers are kept.

−t|−−temp_dir *dir*
: With this option, you call a directory that is the base for the temporary directory. This directory name is used as is without any extensions. If this directory does not exist it is be created. The temporary directory is created by Perl's security operations directly under this directory. In this temporary directory, the temporary files are stored.

Output

−o|−−output *file_name*
: Normally all *groff* output of this program is sent to **STDOUT**. With this option, that can be changed, such that the output is stored into a file named in the option argument *file_name*.

−v|−V|−−verbose
: A lot more of information is sent to STDERR.

Short option collections

The argument handling of options

Short options are arguments that start with a single dash **−**. Such an argument can consist of arbitrary many options without option argument, composed as a collection of option characters following the single dash.

Such a collection can be terminated by an option character that expects an option argument. If this option character is not the last character of the argument, the following final part of the argument is the option argument. If it is the last character of the argument, the next argument is taken as the option argument.

This is the standard for *POSIX* and *GNU* option management.

For example,

−kVe *some_dir*
: is a collection of the short options **−k** and **−V** without option argument, followed by the short option **−e** with option argument that is the following part of the argument *some_dir*. So this argument could also be written as several arguments **−k −V −e** *some_dir*.

Handling of long options

Arguments that start with a double dash **−−** are so-called *long options R* . Each double dash argument can only have a single long option.

Long options have or have not an option argument. An option argument can be the next argument or can be appended with an equal sign **=** to the same argument as the long option.

−−help
: is a long option without an option argument.

−−eps_dir *some_dir*
−−eps_dir=*some_dir*
: is the long option **−−eps_dir** with the option argument *some_dir*.

Moreover the program allows abbreviations of long options, as much as possible.

The *long option* **−−keep_all** can be abbreviated from **−−keep_al** up to **−−k** because the program does not have another *long option* whose name starts with the character **k**.

On the other hand, the option **−−version** cannot be abbreviated further than **−−vers** because there is also the *long option* **−−verbose** that can be abbreviated up to **−−verb**.

An option argument can also be appended to an abbreviation. So is **−−e=***some_dir* the same as **−−eps_dir** *some_dir*.

Moreover the program allows an arbitrary usage of upper and lower case in the option name. This is *Perl* style.

For example, the *long option* **−−keep_all** can as well be written as **−−Keep_All** or even as an abbreviation like **−−KeE**.

LilyPond regions in *roff* input

Integrated LilyPond code

A *lilypond* part within a structure written in the *groff* language is the whole part between the marks

.lilypond start

and

.lilypond end

A *groff* input can have several of these *lilypond* parts.

When processing such a *lilypond* part between **.lilypond start** and **.lilypond end** we say that the **glilypond** program is in *lilypond mode*.

These *lilypond* parts are sent into temporary *lilypond* files with the file name extension **.ly**. These files are transformed later on into *EPS* files.

Inclusion of *.ly* files

An additional command line for file inclusion of *lilypond* files is given by

.lilypond include *file_name*

in *groff* input. For each such *include* command, one file of *lilypond* code can be included into the *groff* code. Arbitrarily many of these commands can be included in the *groff* input.

These include commands can only be used outside the *lilypond* parts. Within the *lilypond mode*, this inclusion is not possible. So **.lilypond include** may not be used in *lilypond mode*, i.e. between **.lilypond start** and **.lilypond end**. These included *ly*-files are also transformed into *EPS* files.

Generated files

By the transformation process of *lilypond* parts into *EPS* files, there are many files generated. By default, these files are regarded as temporary files and as such stored in a temporary directory. This process can be changed by command-line options.

Command-line options for directories

The temporary directory for this program is either created automatically or can be named by the option **−t|−−temp_dir** *dir*.

Moreover, the *EPS* files that are later on referred by **.PSPIC** command in the final *groff* output can be stored in a different directory that can be set by the command-line option **−e|−−eps_dir** *directory_name*. With this option, the temporary directory can be removed completely at the end of the program.

The beginning of the names of the temporary files can be set by the command-line options **−p** or **−−prefix**.

All of the temporary files except the *EPS* files are deleted finally. This can be changed by setting the command-line options **−k** or **−−keep_files**. With this, all temporary files and directories are kept, not deleted.

These *EPS* files are stored in a temporary or *EPS* directory. But they cannot be deleted by the transformation process because they are needed for the display which can take a long time.

Transformation processes for generating EPS files

Mode pdf2eps

In this mode, the current default, **.ly** files are transformed by the *lilypond*(1) program into *PDF* files, using

lilypond −−pdf −−output=*file-name*

for each **.ly** file. The *file-name* must be provided without the extension **.pdf**. By this process, a file *file-name***.pdf** is generated.

The next step is to transform these *PDF* files into a *PS* file. This is done by the *pdf2ps*(1) program using

$ **pdf2ps** *file-name***.pdf** *file-name***.pds**

The next step creates an *EPS* file from the *PS* file. This is done by the *ps2eps*(1) program using

$ **ps2eps** *file-name***.ps**

By that, a file *file-name***.eps** is created for each *lilypond* part in the *groff* file or standard input.

The last step to be done is replacing all *lilypond* parts by the *groff* command

.PSPIC *file-name***.eps**

Mode ly2eps

In earlier time, this mode was the default. But now it does not work any more, so accept the new default *pdf2eps*. For testing, this mode can also be chosen by the *glilypond* option **−−ly2eps**.

In this mode, the **.ly** files are transformed by the *lilypond* program into many files of different formats, including *eps* files, using

```
$ lilypond --ps -dbackend=eps -dgs-load-fonts --output=file-name
```

for each **.ly** file. The output *file−name* must be provided without an extension, its directory is temporary.

There are many *EPS* files created. One having the complete transformed **ly** file, named *file−name***.eps**.

Moreover there are *EPS* files for each page, named *file−name−digit***.eps**.

The last step to be done is replacing all *lilypond* parts by the collection of the corresponding *EPS* page files. This is done by *groff* commands

```
.PSPIC file-name-digit.eps
```

Generated *groff* output

The new *groff*(7) structure generated by *glilypond* is either

1) sent to standard output and can there be saved into a file or piped into *groff*(1) or

2) stored into a file by given the option **−o** | −−output *file_name*

Authors

glilypond was written by Bernd Warken.

See also

groff(1)
: describes the usage of the *groff* command and contains pointers to further documentation of the *groff* system.

groff_tmac(5)
: describes the **.PSPIC** request.

lilypond(1)
: briefly describes the *lilypond* command and contains pointers to further documentation.

pdf2ps(1)
: transforms a *PDF* file into a *PostScript* format.

ps2eps(1)
: transforms a *PS* file into an *EPS* format.

Name

gperl – execute Perl commands in *groff* documents

Synopsis

gperl [*file* ...]

gperl –h
gperl ––help

gperl –v
gperl ––version

Description

This is a preprocessor for *groff*(1). It allows the use of *perl*(7) code in *groff*(7) files. The result of a *Perl part* can be stored in groff *strings* or *numerical registers* based on the arguments at a final line of a *Perl part*.

If no operands are given, or if *file* is "**–**", *gperl* reads the standard input stream. A double-dash argument ("**––**") causes all subsequent arguments to be interpreted as *file* operands, even if their names start with a dash. **–h** and **––help** display a usage message, whereas **–v** and **––version** display version information; all exit afterward.

Perl regions

Perl parts in *groff files* are enclosed by two **.Perl** requests with different arguments, a *starting* and an *ending* command.

Starting Perl mode

The starting *Perl request* can either be without arguments, or by a request that has the term **start** as its only argument.

- **.Perl**
- **.Perl start**

Ending Perl mode without storage

A **.Perl** command line with an argument different from **start** finishes a running *Perl part*. Of course, it would be reasonable to add the argument **stop**; that's possible, but not necessary.

- **.Perl stop**
- **.Perl** *other_than_start*

The argument *other_than_start* can additionally be used as a *groff* string variable name for storage — see next section.

Ending Perl mode with storage

A useful feature of *gperl* is to store one or more results from the *Perl mode*.

The output of a *Perl part* can be got with backticks **`...`**.

This program collects all printing to STDOUT (normal standard output) by the Perl **print** program. This pseudo-printing output can have several lines, due to printed line breaks with **\n**. By that, the output of a Perl run should be stored into a Perl array, with a single line for each array member.

This Perl array output can be stored by *gperl* in either

groff strings
: by creating a groff command **.ds**

groff register
: by creating a groff command **.rn**

The storage modes can be determined by arguments of a final stopping **.Perl** command. Each argument **.ds** changes the mode into *groff string* and **.nr** changes the mode into *groff register* for all following output parts.

By default, all output is saved as strings, so **.ds** is not really needed before the first **.nr** command. That suits to *groff* (7), because every output can be saved as *groff* string, but the registers can be very restrictive.

In *string mode*, *gperl* generates a *groff string* storage line

```
.ds var_name content
```

In *register mode* the following groff command is generated

```
.nr var_name content
```

We present argument collections in the following. You can add as first argument for all **stop**. We omit this additional element.

.Perl .ds *var_name*

This will store 1 output line into the groff string named *var_name* by the automatically created command

```
.ds var_name output
```

.Perl *var_name*

If *var_name* is different from **start** this is equivalent to the former command, because the string mode is string with **.ds** command. default.

.Perl *var_name1 var_name2*

This will store 2 output lines into groff string names *var_name1* and *var_name2*, because the default mode **.ds** is active, such that no **.ds** argument is needed. Of course, this is equivalent to

```
.Perl .ds var_name1 var_name2
```

and

```
.Perl .ds var_name1 .ds var_name2
```

.Perl .nr *var_name1 varname2*

stores both variables as register variables. *gperl* generates

```
.nr var_name1 output_line1
.nr var_name2 output_line2
```

.Perl .nr *var_name1* **.ds** *var_name2*

stores the 1st argument as *register* and the second as *string* by

```
.nr var_name1 output_line1
.ds var_name2 output_line2
```

Example

A possible *Perl part* in a *roff file* could look like that:

```
before
.Perl start
my $result = 'some data';
print $result;
.Perl stop .ds string_var
after
```

This stores the result **"some data"** into the *roff string* called **string_var**, such that the following line is printed:

```
.ds string_var some data
```

by *gperl* as food for the coming *groff* run.

A *Perl part* with several outputs is:

```
.Perl start
print "first\n";
print "second line\n";
print "3\n";
.Perl var1 var2 .nr var3
```

This stores 3 printed lines into 3 *groff* strings. **var1**,**var2**,**var3**. So the following *groff* command lines are created:

```
.ds var1 first
.ds var2 second line
.nr var3 3
```

Authors

gperl was written by Bernd Warken.

See also

Man pages related to *groff* are *groff*(1), *groff*(7), and *grog*(1).

Documents related to *Perl* are *perl*(1), *perl*(7).

Name

gpinyin – use Hanyu Pinyin Chinese in *groff* documents

Synopsis

gpinyin [*file* ...]

gpinyin −h
gpinyin −−help

gpinyin −v
gpinyin −−version

Description

gpinyin is a preprocessor for *groff*(1) that facilitates use of Hanyu Pinyin in *groff*(7) files. Pinyin is a method for writing the Mandarin Chinese language with the Latin alphabet. Mandarin consists of more than four hundred base syllables, each spoken with one of five different tones. Changing the tone applied to the syllable generally alters the meaning of the word it forms. In Pinyin, a syllable is written in the Latin alphabet and a numeric tone indicator can be appended to each syllable.

Each *input-file* is a file name or the character "**−**" to indicate that the standard input stream should be read. As usual, the argument "**−−**" can be used in order to force interpretation of all remaining arguments as file names, even if an *input-file* argument begins with a "**−**". **−h** and **−−help** display a usage message, while **−v** and **−−version** show version information; all exit afterward.

Pinyin sections

Pinyin sections in *groff* files are enclosed by two **.pinyin** requests with different arguments. The starting request is

```
.pinyin start
```

or

```
.pinyin begin
```

and the ending request is

```
.pinyin stop
```

or

```
.pinyin end
```

.

Syllables

In Pinyin, each syllable is represented by one to six letters drawn from the fifty-two upper- and lowercase letters of the Unicode basic Latin character set, plus the letter "U" with dieresis (umlaut) in both cases—in other words, the members of the set "[a–zA–ZüÜ]".

In *groff* input, all basic Latin letters are written as themselves. The "u with dieresis" can be written as "\[:u]" in lowercase or "\[:U]" in uppercase. Within **.pinyin** sections, *gpinyin* supports the form "ue" for lowercase and the forms "Ue" and "UE" for uppercase.

Tones

Each syllable has exactly one of five *tones*. The fifth tone is not explicitly written at all, but each of the first through fourth tones is indicated with a diacritic above a specific vowel within the syllable.

In a *gpinyin* source file, these tones are written by adding a numeral in the range 0 to 5 after the syllable. The tone numbers 1 to 4 are transformed into accents above vowels in the output. The tone numbers 0 and 5 are synonymous.

The tones are written as follows.

Tone	Description	Diacritic	Example Input	Example Output
first	flat	¯	ma1	mā
second	rising	´	ma2	má
third	falling-rising	ˇ	ma3	mǎ
fourth	falling	`	ma4	mà
fifth	neutral	(none)	ma0 ma5	ma

The neutral tone number can be omitted from a word-final syllable, but not otherwise.

Authors

gpinyin was written by Bernd Warken.

See also

Useful documents on the World Wide Web related to Pinyin include

Pinyin to Unicode,
On-line Chinese Tools,
Pinyin.info: a guide to the writing of Mandarin Chinese in romanization,
"Where do the tone marks go?",
pinyin.txt from the CJK macro package for TEX,

and

pinyin.sty from the CJK macro package for TEX.

groff(1) and *grog*(1) explain how to view *roff* documents.

groff(7) and *groff_char*(7) are comprehensive references covering the language elements of GNU *troff* and the available glyph repertoire, respectively.

Name

grap2graph – convert a *grap* diagram into a cropped image

Synopsis

grap2graph [**–unsafe**] [**–format** *output-format*] [*convert-argument* ...]

grap2graph ––help

grap2graph –v
grap2graph ––version

Description

grap2graph reads a *grap*(1) program from the standard input and writes an image file, by default in Portable Network Graphics (PNG) format, to the standard output.

The input GRAP code should *not* be wrapped with the **.G1** and **.G2** macros that normally guard it within *groff*(1) documents.

Arguments not recognized by *grap2graph* are passed to the ImageMagick or GraphicsMagick program *convert*(1). By specifying these, you can give your image a border, set the image's pixel density, or perform other useful transformations.

The output image is clipped using *convert*'s **–trim** option to the smallest possible bounding box that contains all the black pixels.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–format *output-format*
: Write the image in *output-format*, which must be understood by *convert*; the default is PNG.

–unsafe
: Run *groff* in *unsafe* mode, enabling the PIC command **sh** to execute arbitrary Unix shell commands. The *groff* default is to forbid this.

Environment

GROFF_TMPDIR
TMPDIR
TMP
TEMP
: These environment variables are searched in the order shown to determine the directory where temporary files will be created. If none are set, */tmp* is used.

Authors

grap2graph was written by Eric S. Raymond, based on a recipe for *pic2graph*(1), by W. Richard Stevens.

See also

pic2graph(1), *eqn2graph*(1), *grap*(1), *pic*(1), *groff*(1), *convert*(1)

Name

grn – embed Gremlin images in *groff* documents

Synopsis

grn [**−C**] [**−T** *dev*] [**−M** *dir*] [**−F** *dir*] [*file* . . .]

grn −?
grn −−help

grn −v
grn −−version

Description

grn is a preprocessor for including *gremlin* pictures in *troff*(1) input. *grn* writes to standard output, processing only input lines between two that start with **.GS** and **.GE**. Those lines must contain *grn* commands (see below). These macros request a *gremlin* file; the picture in that file is converted and placed in the *troff* input stream. **.GS** may be called with a **C**, **L**, or **R** argument to center, left-, or right-align the whole *gremlin* picture (the default is to center). If no *file* is mentioned, the standard input is read. At the end of the picture, the position on the page is the bottom of the *gremlin* picture. If the *grn* entry is ended with **.GF** instead of **.GE**, the position is left at the top of the picture.

Currently only the *me* macro package has support for **.GS**, **.GE**, and **.GF**.

grn produces drawing escape sequences that use *groff*'s color scheme extension (**\D'F** . . .'), and thus may not work with other *troff*s.

grn commands

Each input line between **.GS** and **.GE** may have one *grn* command. Commands consist of one or two strings separated by whitespace, the first string being the command and the second its operand. Commands may be upper- or lowercase and abbreviated down to one character.

Commands that affect a picture's environment (those listed before "**default**", see below) are only in effect for the current picture: the environment is reinitialized to the defaults at the start of the next picture. The commands are as follows.

1 *N*
2 *N*
3 *N*
4 *N*
: Set *gremlin*'s text size number 1 (2, 3, or 4) to *N* points. The default is 12 (16, 24, and 36, respectively).

roman *f*
italics *f*
bold *f*
special *f*
: Set the roman (italics, bold, or special) font to *troff*'s font *f* (either a name or number). The default is R (I, B, and S, respectively).

l *f*
stipple *f*
: Set the stipple font to *troff*'s stipple font *f* (name or number). The command **stipple** may be abbreviated down as far as "**st**" (to avoid confusion with "**special**"). There is *no* default for stipples (unless one is set by the "**default**" command), and it is invalid to include a *gremlin* picture with polygons without specifying a stipple font.

x *N*
scale *N*
: Magnify the picture (in addition to any default magnification) by *N*, a floating-point number larger than zero. The command **scale** may be abbreviated down to "**sc**".

narrow *N*
medium *N*
thick *N*

Set the thickness of *gremlin*'s narrow (medium and thick, respectively) lines to *N* times 0.15pt (this value can be changed at compile time). The default is 1.0 (3.0 and 5.0, respectively), which corresponds to 0.15pt (0.45pt and 0.75pt, respectively). A thickness value of zero selects the smallest available line thickness. Negative values cause the line thickness to be proportional to the current point size.

pointscale [**off**|**on**]

Scale text to match the picture. Gremlin text is usually printed in the point size specified with the commands **1**, **2**, **3**, or **4**, regardless of any scaling factors in the picture. Setting **pointscale** will cause the point sizes to scale with the picture (within *troff*'s limitations, of course). An operand of anything but **off** will turn text scaling on.

default

Reset the picture environment defaults to the settings in the current picture. This is meant to be used as a global parameter setting mechanism at the beginning of the *troff* input, but can be used at any time to reset the default settings.

width *N*

Force the picture to be *N* inches wide. This overrides any scaling factors present in the same picture. "**width 0**" is ignored.

height *N*

Force the picture to be *N* inches high, overriding other scaling factors. If both **width** and **height** are specified, the tighter constraint will determine the scale of the picture. **height** and **width** commands are not saved with a "**default**" command. They will, however, affect point size scaling if that option is set.

file *name*

Get picture from *gremlin* file *name* located the current directory (or in the library directory; see the **–M** option above). If multiple **file** commands are given, the last one controls. If *name* doesn't exist, an error message is reported and processing continues from the **.GE** line.

Usage with *groff*

Since *grn* is a preprocessor, it has no access to elements of formatter state, such as indentation, line length, type size, or register values. Consequently, no *troff* input can be placed between the **.GS** and **.GE** macros. However, *gremlin* text elements are subsequently processed by *troff*, so anything valid in a single line of *troff* input is valid in a line of *gremlin* text (barring the dot control character "." at the beginning of a line). Thus, it is possible to have equations within a *gremlin* figure by including in the *gremlin* file *eqn* expressions enclosed by previously defined delimiters (e.g., "$$").

When using *grn* along with other preprocessors, run *tbl*(1) before *grn*, *pic*(1), and/or *ideal*. If *eqn*(1) is needed, run it as the last preprocessor. *groff*(1) automatically runs preprocessors in the correct order.

A picture is considered an entity, but that doesn't stop *troff* from trying to break it up if it falls off the end of a page. Placing the picture between "keeps" in the *me* macros will ensure proper placement.

grn uses *troff*'s registers **g1** through **g9** and sets registers **g1** and **g2** to the width and height of the *gremlin* figure (in device units) before entering the **.GS** macro (this is for those who want to rewrite these macros).

Gremlin file format

There exist two distinct *gremlin* file formats: the original format for AED graphic terminals, and the Sun or X11 version. An extension used by the Sun/X11 version allowing reference points with negative coordinates is *not* compatible with the AED version. As long as a *gremlin* file does not contain negative coordinates, either format will be read correctly by either version of *gremlin* or *grn*. The other difference in Sun/X11 format is the use of names for picture objects (e.g., **POLYGON**, **CURVE**) instead of numbers. Files representing the same picture are shown below.

```
sungremlinfile          gremlinfile
0 240.00 128.00         0 240.00 128.00
CENTCENT                2
240.00 128.00           240.00 128.00
185.00 120.00           185.00 120.00
240.00 120.00           240.00 120.00
296.00 120.00           296.00 120.00
*                       -1.00 -1.00
2 3                     2 3
10 A Triangle           10 A Triangle
POLYGON                 6
224.00 416.00           224.00 416.00
96.00 160.00            96.00 160.00
384.00 160.00           384.00 160.00
*                       -1.00 -1.00
5 1                     5 1
0                       0
-1                      -1
```

- The first line of each *gremlin* file contains either the string "**gremlinfile**" (AED) or "**sungremlinfile**" (Sun/X11).
- The second line of the file contains an orientation and *x* and *y* values for a positioning point, separated by spaces. The orientation, either **0** or **1**, is ignored by the Sun/X11 version. **0** means that *gremlin* will display things in horizontal format (a drawing area wider than it is tall, with a menu across the top). **1** means that *gremlin* will display things in vertical format (a drawing area taller than it is wide, with a menu on the left side). *x* and *y* are floating-point values giving a positioning point to be used when this file is read into another file. The stuff on this line really isn't all that important; a value of "**1 0.00 0.00**" is suggested.
- The rest of the file consists of zero or more element specifications. After the last element specification is a line containing the string "**–1**".
- Lines longer than 127 characters are truncated to that length.

Element specifications

- The first line of each element contains a single decimal number giving the type of the element (AED) or its name (Sun/X11).

gremlin File Format: Object Type Specification

AED Number	Sun/X11 Name	Description
0	**BOTLEFT**	text aligned to bottom left
1	**BOTRIGHT**	text aligned to bottom right
2	**CENTCENT**	center object
3	**VECTOR**	vector
4	**ARC**	arc
5	**CURVE**	curve
6	**POLYGON**	polygon
7	**BSPLINE**	b-spline
8	**BEZIER**	Bézier
10	**TOPLEFT**	text aligned to top left
11	**TOPCENT**	text aligned to top center
12	**TOPRIGHT**	text aligned to top right
13	**CENTLEFT**	text aligned to center left
14	**CENTRIGHT**	text aligned to center right
15	**BOTCENT**	text aligned to bottom center

- Each line after the object type specifies a point used to display the element. It contains an *x* and a *y* coordinate in floating-point format, separated by spaces. The list of points is terminated by a line containing the string "**−1.0 −1.0**" (AED) or a single asterisk, "*" (Sun/X11).
- After the points comes a line containing two decimal values, giving the brush and size for the element. The brush determines the style in which things are drawn. For vectors, arcs, and curves there are six valid brush values.

  1 thin dotted lines
  2 thin dot-dashed lines
  3 thick solid lines
  4 thin dashed lines
  5 thin solid lines
  6 medium solid lines

  For polygons, 0 is also valid: it specifies an invisible border. For text, the brush selects a font as follows.

  1 roman (R font in *troff*)
  2 italics (I font in *troff*)
  3 bold (B font in *troff*)
  4 special (S font in *troff*)

  If you're using *grn* to run your pictures through *groff*, the font is really just a starting font. The text string can contain formatting sequences like "\fI" or "\d" which may change the font (as well as do many other things). For text, the size field is a decimal value between 1 and 4. It selects the size of the font in which the text will be drawn. For polygons, this size field is interpreted as a stipple number to fill the polygon with. The number is used to index into a stipple font at print time.
- The last line of each element contains a decimal number and a string of characters, separated by a single space. The number is a count of the number of characters in the string. This information is used only for text elements, and contains the text string. There can be spaces inside the text. For arcs, curves, and vectors, the character count is zero (**0**), followed by exactly one space before the newline.

Coordinates

gremlin was designed for AED terminals, and its coordinates reflect the AED coordinate space. For vertical pictures, *x* values range 116 to 511, and *y* values from 0 to 483. For horizontal pictures, *x* values range from 0 to 511, and *y* values from 0 to 367. Although you needn't absolutely stick to this range, you'll get better results if you at least stay in this vicinity. Also, point lists are terminated by a point of (−1, −1), so you shouldn't ever use negative coordinates. *gremlin* writes out coordinates using the *printf*(3) format "%f1.2"; it's probably a good idea to use the same format if you want to modify the *grn* code.

Sun/X11 coordinates

There is no restriction on the range of coordinates used to create objects in the Sun/X11 version of *gremlin*. However, files with negative coordinates *will* cause problems if displayed on the AED.

Options

−? and **−−help** display a usage message, while **−v** and **−−version** show version information; all exit afterward.

−C Recognize **.GS** and **.GE** (and **.GF**) even when followed by a character other than space or newline.

−F *dir* Search *dir* for subdirectories *dev*name (*name* is the name of the output driver) for the *DESC* file before the default font directories */etc/groff/site−font*, */usr/share/groff/1.24.2/font*, and */usr/lib/font*.

−M *dir* Prepend *dir* to the search path for *gremlin* files. The default search path is the current directory, the home directory, */etc/groff/site−tmac*, and */usr/share/groff/1.24.2/tmac*, in that order.

−T *dev* Prepare device output using output driver *dev*. The default is **ps**. See *groff*(1) for a list of valid devices.

Exit status

grn exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Files

*/usr/share/groff/1.24.2/font/dev*name*/DESC*
describes the output device *name*.

Authors

David Slattengren and Barry Roitblat wrote the original Berkeley *grn*. Daniel Senderowicz and Werner Lemberg modified it for *groff*.

See also

gremlin(1), *groff*(1), *pic*(1), *ideal*(1)

Name

grodvi – *groff* output driver for TeX DVI format

Synopsis

grodvi [**–dl**] [**–F** *dir*] [**–p** *paper-format*] [**–w** *n*] [*file* ...]

grodvi ––help

grodvi –v
grodvi ––version

Description

The GNU *roff* DVI output driver translates the output of *troff*(1) into TEX DVI format. Normally, *grodvi* is invoked by *groff*(1) when the latter is given the "**–T dvi**" option. (In this installation, **ps** is the default output device.) Use *groff*'s **–P** option to pass any options shown above to *grodvi*. If no *file* arguments are given, or if *file* is "–", *grodvi* reads the standard input stream. It writes to the standard output stream.

The DVI file generated by *grodvi* can be interpreted by any correctly written DVI driver. *troff* drawing primitives are implemented using *tpic* version 2 specials. If the driver does not support these, **\D** escape sequences will not produce any output.

Encapsulated PostScript (EPS) files can be easily included; use the **PSPIC** macro. *pspic.tmac* is loaded automatically by *dvi.tmac*. See *groff_tmac*(5).

The default color used by the **\m** and **\M** escape sequences is black. Currently, the stroke color for **\D** drawing escape sequences is black; fill color values are translated to gray.

In *groff*, as in AT&T *troff*, the **\N** escape sequence can be used to access any glyph in the current font by its position in the corresponding TFM file.

By design, the DVI format doesn't care about the physical dimensions of the output medium. See subsection "Device extension commands" below.

Typefaces

grodvi supports the standard four styles: **R** (roman), **I** (*italic*), **B** (**bold**), and **BI** (***bold-italic***). Fonts are grouped into families **T** and **H** having members in each style. "CM" abbreviates "Computer Modern".

TR	CM Roman (cmr10)
TI	CM Text Italic (cmti10)
TB	CM Bold Extended Roman (cmbx10)
TBI	CM Bold Extended Text Italic (cmbxti10)
HR	CM Sans Serif (cmss10)
HI	CM Slanted Sans Serif (cmssi10)
HB	CM Sans Serif Bold Extended (cmssbx10)
HBI	CM Slanted Sans Serif Bold Extended (cmssbxo10)

The following fonts are not members of a family.

CW	CM Typewriter Text (cmtt10)
CWI	CM Italic Typewriter Text (cmitt10)

Special fonts include **MI** (cmmi10), **S** (cmsy10), **EX** (cmex10), **SC** (cmtex10, only for **CW**), and, perhaps surprisingly, **TR**, **TI**, and **CW**, because TEX places some glyphs in text fonts that *troff* generally does not. For italic fonts, **CWI** is used instead of **CW**.

Finally, the symbol fonts of the American Mathematical Society are available as special fonts **SA** (msam10) and **SB** (msbm10). They are not mounted by default.

You can load the *ec.tmac* macro file to employ the EC and TC fonts instead of CM, which they resemble. They also provide Euro **\[Eu]** and per mille **\[%0]** glyphs. Do so before loading localization macro files, because *ec.tmac* does not set up automatic hyphenation codes.

Device extension commands

grodvi emits the equivalent to TEX's **\special{papersize=***width***,***length***}** on the first page; *dvips* (or another DVI driver) then sets the page size accordingly. If either the page width or length is not positive, no **papersize** special is output.

grodvi supports one device extension, accessed with the *groff* request **device** or *roff* **\X** escape sequence.

\X'papersize=*width***,***length***'**
: Set the page dimensions in centimeters to *width* by *length*. If the **–l** option was specified, these dimensions are swapped. Changes to the paper dimensions should occur prior to the first page, or during page ejection before starting a subsequent one.

  Caution: the ordering of dimensions differs from that used by *papersize.tmac* and *troff*(1)'s "**–d paper**" option.

The parameter(s) to **device** and **\X** are translated to the same DVI file instructions as would be produced by **\special{***anything***}** in TEX; *anything* cannot contain a newline.

Font description files

Use *tfmtodit*(1) to create *groff* font description files from TFM (TEX font metrics) files. The font description file should contain the following additional directives, which *tfmtodit* generates automatically.

internalname *name*
: The name of the TFM file (without the *.tfm* extension) is *name*.

checksum *n*
: The checksum in the TFM file is *n*.

designsize *n*
: The design size in the TFM file is *n*.

Drawing commands

grodvi supports an additional drawing command.

\D'R *dh dv***'**
: Draw a rule (solid black rectangle) with one corner at the drawing position, and the diagonally opposite corner at the drawing position +(*dh*,*dv*), which becomes the new drawing position afterward. This command produces a rule in the DVI file and so can be printed even with a driver that does not support *tpic* specials, unlike the other **\D** commands.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–d
: Do not use *tpic* specials to implement drawing commands. Horizontal and vertical lines are implemented by rules. Other drawing commands are ignored.

–F *dir*
: Prepend directory dir*/dev*name to the search path for font and device description files; *name* is the name of the device, usually **dvi**.

–l
: Use landscape orientation rather than portrait.

–p *paper-format*
: Set physical dimensions of output medium, overriding the **papersize**, **paperlength**, and **paperwidth** directives in the *DESC* file. *paper-format* can be any argument accepted by the **papersize** directive; see *groff_font*(5).

–w *n*
: Draw rules (lines) with a thickness of *n* thousandths of an em. The default thickness is **40** (0.04 em).

Exit status

grodvi exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Environment

GROFF_FONT_PATH
: lists directories in which to search for *devdvi*, *grodvi*'s directory of device and font description files. See *troff* (1) and *groff_font*(5).

Files

/usr/share/groff/1.24.2/font/devdvi/DESC
: describes the **dvi** output device.

*/usr/share/groff/1.24.2/font/devdvi/*F
: describes the font known as *F* on device **dvi**.

/usr/share/groff/1.24.2/tmac/dvi.tmac
: defines font mappings, special characters, and colors for use with the **dvi** output device. It is automatically loaded by *troffrc* when the **dvi** output device is selected.

/usr/share/groff/1.24.2/tmac/ec.tmac
: configures the **dvi** output device to use the EC and TC font families instead of CM (Computer Modern).

Bugs

DVI files produced by *grodvi* use a different resolution (57,816 units per inch) from those produced by TEX. Incorrectly written drivers which assume the resolution used by TEX, rather than using the resolution specified in the DVI file, will not work with *grodvi*.

When using the **−d** option with boxed tables, vertical and horizontal lines can sometimes protrude by one pixel. This is a consequence of the way TEX requires that the heights and widths of rules be rounded.

See also

"What are the EC fonts?"; TEX FAQ: Frequently Asked Question List for TEX

tfmtodit(1), *groff* (1), *troff* (1), *groff_out*(5), *groff_font*(5), *groff_char*(7), *groff_tmac*(5)

Name

groff – front end to the GNU *roff* document formatting system

Synopsis

groff [**–abcCeEgGijklNpRsStUVXzZ**] [**–d** *ctext*] [**–d** *string*=*text*] [**–D** *fallback-encoding*] [**–f** *font-family*] [**–F** *font-directory*] [**–I** *inclusion-directory*] [**–K** *input-encoding*] [**–L** *spooler-argument*] [**–m** *macro-package*] [**–M** *macro-directory*] [**–n** *page-number*] [**–o** *page-list*] [**–P** *postprocessor-argument*] [**–r** *cnumeric-expression*] [**–r** *register*=*numeric-expression*] [**–T** *output-device*] [**–w** *warning-category*] [**–W** *warning-category*] [*file* . . .]

groff –h
groff ––help

groff –v [*option* . . .] [*file* . . .]
groff ––version [*option* . . .] [*file* . . .]

Description

groff is the primary front end to the GNU *roff* document formatting system. GNU *roff* is a typesetting system that reads plain text input that includes formatting commands to produce output in PostScript, PDF, HTML, or other formats, or for display to a terminal. Formatting commands can be low-level typesetting primitives, macros from a supplied package, or user-defined macros. All three approaches can be combined. If no *file* operands are specified, or if *file* is "**–**", *groff* reads the standard input stream.

A reimplementation and extension of *troff* and other programs from AT&T Unix, *groff* is widely available on POSIX and other systems owing to its long association with Unix manuals, including man pages. It and its predecessor have produced several best-selling software engineering texts. *groff* can create typographically sophisticated documents while consuming minimal system resources.

Like its predecessor "troff", the term "groff" affords two popular pronunciations: as one syllable (like the surname), rhyming with "trough", or as "jee-roff", in analogy to the Bell Labs pronunciation "tee-roff". Little risk of confusion exists; use whichever suits you.

The *groff* command orchestrates the execution of preprocessors, the transformation of input documents into a device-independent page description language, and the production of output from that language.

Options

–h and **––help** display a usage message and exit.

Because *groff* is intended to subsume most users' direct invocations of the *troff*(1) formatter, the two programs share a set of options. However, *groff* has some options that *troff* does not share, and others which *groff* interprets differently. At the same time, not all valid *troff* options can be given to *groff*.

groff-specific options

The following options either do not exist in GNU *troff* or are interpreted differently by *groff*.

–D *enc* Use *enc* as *preconv*(1)'s fallback input encoding; implies **–k**.

–e Run *eqn*(1) preprocessor.

–g Run *grn*(1) preprocessor.

–G Run *grap*(1) preprocessor; implies **–p**.

–I *dir* Works as *troff*'s option (see below), but also implies **–g** and **–s**. *groff* passes **–I** options and their arguments to *soelim*(1), *troff*(1), and output drivers; with the option letter changed to **–M**, it passes the same arguments to *grn*(1).

–j Run *chem*(1) preprocessor; implies **–p**.

–k Run *preconv*(1) preprocessor. Refer to its man page for its behavior if neither of *groff*'s **–K** or **–D** options is also specified.

−K *enc* Set input encoding used by *preconv*(1) to *enc*; implies **−k**.

−l Send the output to a spooler program for printing. The "**print**" directive in the device description file specifies the default command to be used; see *groff_font*(5). If no such directive is present for the output device, this option is ignored. See options **−L** and **−X**.

−L *arg* Pass *arg* to the print spooler. If multiple *arg*s are required, pass each with a separate **−L** option. *groff* does not prefix an option dash to *arg* before passing it to the spooler.

−M Works as *troff*'s option (see below), but is also passed to *eqn*(1), *grap*(1), and *grn*(1).

−N Prohibit newlines between *eqn* delimiters: pass **−N** to *eqn*(1).

−p Run *pic*(1) preprocessor.

−P *arg* Pass *arg* to the postprocessor. If multiple *arg*s are required, pass each with a separate **−P** option. *groff* does not prefix an option dash to *arg* before passing it to the postprocessor.

−R Run *refer*(1) preprocessor. No mechanism is provided for passing arguments to it; most *refer* options have equivalent language elements that can be specified within the document.

−s Run *soelim*(1) preprocessor.

−S Enable safer mode and ignore any subsequent **−U** option. *groff* passes the **−S** option to *pic*(1) and *troff*(1).

−t Run *tbl*(1) preprocessor.

−T *dev* Prepare output for device *dev*. *groff* passes the **−T** option and its argument to *troff*, then (unless the **−Z** option is used) runs an output driver to convert *troff*'s output to a form appropriate for *dev*; see subsection "Output devices" below.

−U Operate in unsafe mode. *groff* passes the **−U** option to *pic*(1) and *troff*(1).

−−version

−v Write version information for *groff* and all programs run by it to the standard output stream; that is, the given command line is processed in the usual way, passing **−v** to the formatter and any pre- or postprocessors invoked.

−V Output the pipeline that *groff* would run to the standard output stream and exit. If given more than once, *groff* both writes the pipeline to the standard error stream and runs it.

−X Use *gxditview*(1) instead of the usual postprocessor to (pre)view a document on an X11 display. Combining this option with "**−T ps**" uses the font metrics of the PostScript device, whereas the "**−T X75**", "**−T X75−12**" "**−T X100**", and "**−T X100−12**" options use the metrics of X11 fonts.

−Z Disable postprocessing. *troff* output appears on the standard output stream (unless suppressed with **−z**); see *groff_out*(5) for a description of this format.

Transparent options

The following options are passed as-is to the formatter program *troff*(1) and described in more detail in its man page.

−a Generate a plain text approximation of the typeset output.

−b Write a backtrace to the standard error stream on each error or warning.

−c Start with color output disabled.

−C Enable AT&T *troff* compatibility mode; implies **−c**.

−d *ctext*
−d *string*=*text*
Define string.

–E Inhibit *troff* error messages; implies **–Ww**.

–f *fam* Set default font family.

–F *dir* Search in directory *dir* for the selected output device's directory of device and font description files.

–i Process standard input after the specified input files.

–I *dir* Search *dir* for input files.

–m *mac* Read macro package *mac* before input. *groff* passes **–m** options and their arguments to *eqn*(1), *grap*(1), and *grn*(1).

–M *dir* Search directory *dir* for macro files. *groff* passes **–M** options and their arguments to *eqn*(1), *grap*(1), and *grn*(1).

–n *num* Begin numbering pages at *num.*

–o *list* Output only pages in *list*.

–r *cnumeric-expression*
–r *register*=*numeric-expression*
Define register.

–S Enable safer mode and ignore any subsequent **–U** option.

–U Operate in unsafe mode.

–w *cat*
–W *cat* Enable and inhibit, respectively, warnings in category *cat.*

–z Suppress formatted device-independent output of *troff*.

Usage

The architecture of the GNU *roff* system follows that of other device-independent *roff* implementations, comprising preprocessors, macro packages, output drivers (or "postprocessors"), and a suite of utilities, with the formatter program *troff* (1) at its heart. See *roff* (7) for a survey of how a *roff* system works.

The front end programs available in the GNU *roff* system make it easier to use than traditional *roff*'s that required the construction of pipelines or use of temporary files to carry a source document from maintainable form to device-ready output. The discussion below summarizes the constituent parts of the GNU *roff* system. It complements *roff* (7) with *groff*-specific information.

Getting started

Those who prefer to learn by experimenting or are desirous of rapid feedback from the system may wish to start with a "Hello, world!" document.

```
$ echo "Hello, world!" | groff -Tascii | sed '/^$/d'
Hello, world!
```

We used a *sed*(1) command only to eliminate the 65 blank lines that would otherwise flood the terminal screen. (*roff* systems were developed in the days of paper-based terminals with 66 lines to a page.)

Today's users may prefer output to a UTF-8-capable terminal.

```
$ echo "Hello, world!" | groff -Tutf8 | sed '/^$/d'
```

Producing PDF, HTML, or TEX's DVI is also straightforward. The hard part may be selecting a viewer program for the output.

```
$ echo "Hello, world!" | groff -Tpdf > hello.pdf
$ evince hello.pdf
$ echo "Hello, world!" | groff -Thtml > hello.html
$ firefox hello.html
$ echo "Hello, world!" | groff -Tdvi > hello.dvi
```

```
$ xdvi hello.dvi
```

Using *groff* as a REPL

Those with a programmer's bent may be pleased to know that they can use *groff* in a read-evaluate-print loop (REPL). Doing so can be handy to verify one's understanding of the formatter's behavior and/or the syntax it accepts. Turning on all warnings with **-ww** can aid this goal.

```
$ groff -ww -Tutf8
\# This is a comment. Let's define a register.
.nr a 1
\# Do integer arithmetic with operators evaluated left-to-right.
.nr b \n[a]+5/2
\# Let's get the result on the standard error stream.
.tm \n[b]
3
\# Now we'll define a string.
.ds name Leslie\" This is another form of comment.
.nr b (\n[a] + (7/2))
\# Center the next two text input lines.
.ce 2
Hi, \*[name].
Your secret number is \n[b].
\# We will see that the division rounded toward zero.
It is
\# Here's an if-else control structure.
.ie (\n[b] % 2) odd.
.el even.
\# This trick sets the page length to the current vertical
\# position, so that blank lines don't spew when we're done.
.pl \n[nl]u
<Control-D>
                        Hi, Leslie.
                Your secret number is 4.

It is even.
```

Paper format

The formatter reads the device description file *DESC* for the selected output device when it starts; page dimensions declared there are used if present. *groff*'s build process configures a default page format and writes it to typesetters' *DESC* files. This installation defaults to "**letter**". If the *DESC* file lacks this information, the formatter and output driver use a page length of **"**11i**"** (eleven inches) for compatibility with AT&T *troff*. See *groff_font*(5).

In the formatter, the **pl** request changes the page length, but macro packages often do not support alteration of the paper format within a document. One might, for instance, want to switch between portrait and landscape orientations. Macro packages lack a consistent approach to configuration of parameters dependent on the paper format; some, like *ms*, benefit from a preamble in the document prior to the first macro call, while others, like *mm*, instead require the specification of registers on the command line, or otherwise before its macro file is interpreted, to configure page dimensions.

Output drivers for typesetters also recognize command-line options **-p** to override the default page dimensions and **-l** to use landscape orientation. The output driver's man page, such as *grops*(1), may be helpful.

groff's " **-d paper**" command-line option is a convenient means of setting the paper format; see *groff_tmac*(5). Combine it with appropriate **-P** options for the output driver, overriding its defaults. The following command formats for PostScript on A4 paper in landscape orientation.

```
$ groff -T ps -d paper=a4l -P -pa4 -P -l -ms my.ms >my.ps
```

Front end

The *groff* program wraps *troff* (1), allowing one to specify preprocessors via command-line options and running the appropriate output driver for the selected output device. This convenience avoids the manual construction of pipelines or management of temporary files required of users of traditional *roff* (7) systems. Use *grog*(1) to infer an appropriate *groff* command line to format a document.

Language

Input to a *roff* system is in plain text interleaved with control lines and escape sequences. The combination constitutes a document in one of a family of languages we also call *roff*; see *roff* (7) for background. An overview of GNU *roff* language syntax and features, including lists of all supported escape sequences, requests, and predefined registers, can be found in *groff* (7). GNU *roff* extensions to the AT&T *troff* language, a common subset of *roff* dialects extant today, are detailed in *groff_diff* (7).

Preprocessors

A preprocessor interprets a domain-specific language that produces *roff* language output. Frequently, such input is confined to sections or regions of *roff* input (bracketed with macro calls specific to each preprocessor), which it replaces. Preprocessors therefore often interpret a subset of *roff* syntax along with their own language. GNU *roff* provides reimplementations of most preprocessors familiar to users of AT&T *troff*; these routinely have extended features and/or require GNU *troff* to format their output.

tbl	lays out tables;
eqn	typesets mathematics;
pic	draws diagrams;
refer	processes bibliographic references;
soelim	preprocesses "sourced" input files;
grn	renders *gremlin*(1) diagrams;
chem	draws chemical structural formulæ using *pic*;
gperl	populates *groff* registers and strings using *perl*(1);
glilypond	embeds *LilyPond* sheet music; and
gpinyin	eases Mandarin Chinese input using Hanyu Pinyin.

A preprocessor unique to GNU *roff* is *preconv*(1), which converts various input encodings to something GNU *troff* can understand. When used, it is run before any other preprocessors.

Most preprocessors enclose content between a pair of characteristic tokens. Such a token must occur at the beginning of an input line and use the dot control character. Spaces and tabs must not follow the control character or precede the end of the input line. Deviating from these rules defeats a token's recognition by the preprocessor. Tokens are generally preserved in preprocessor output and interpreted as macro calls subsequently by *troff*. The *ideal* preprocessor is not yet available in *groff*.

preprocessor	starting token	ending token
chem	`.cstart`	`.cend`
eqn	`.EQ`	`.EN`
grap	`.G1`	`.G2`
grn	`.GS`	`.GE`
ideal	`.IS`	`.IE`
		`.IF`
pic	`.PS`	`.PE`
		`.PF`
		`.PY`
refer	`.R1`	`.R2`
tbl	`.TS`	`.TE`
glilypond	`.lilypond start`	`.lilypond stop`
gperl	`.Perl start`	`.Perl stop`
gpinyin	`.pinyin start`	`.pinyin stop`

Macro packages

Macro files are *roff* input files designed to produce no output themselves but instead ease the preparation of other *roff* documents. When a macro file is installed at a standard location and suitable for use by a general audience, it is termed a *macro package*.

The **−m** option loads a macro package prior to any *roff* input documents, and after performing any string and register assignments directed by **−d** and **−r** options. The GNU *roff* system implements most well-known macro packages for AT&T *troff* in a compatible way and extends them. These have one- or two-letter names arising from intense practices of naming economy in early Unix culture, a laconic approach that led to many of the packages being identified in general usage with the *nroff* and *troff* option letter used to invoke them, sometimes to punning effect, as with "man" (short for "manual"), and even with the option dash, as in the case of the *s* package, much better known as *ms* or even *−ms*.

Macro packages serve a variety of purposes. Some are "full-service" packages, adopting responsibility for page layout among other fundamental tasks, and defining their own lexicon of macros for document composition; each such package stands alone and a given document can use at most one.

an is used to compose man pages in the format originating in Version 7 Unix (1979); see *groff_man*(7). It can be specified on the command line as **−man**.

doc is used to compose man pages in the format originating in 4.3BSD-Reno (1990); see *groff_mdoc*(7). It can be specified on the command line as **−mdoc**.

e is the Berkeley general-purpose macro suite, developed as an alternative to AT&T's *s*; see *groff_me*(7). It can be specified on the command line as **−me**.

m implements the format used by the second-generation AT&T macro suite for general documents, a successor to *s*; see *groff_mm*(7). It can be specified on the command line as **−mm**.

om (invariably called "mom") is a modern package written by Peter Schaffter specifically for GNU *roff*. Consult the *mom* HTML manual for extensive documentation. She—for *mom* takes the female pronoun—can be specified on the command line as **−mom**.

s is the original AT&T general-purpose document format; see *groff_ms*(7). It can be specified on the command line as **−ms**.

Others are supplemental. For instance, *andoc* is a wrapper package specific to GNU *roff* that recognizes whether a document uses *man* or *mdoc* format and loads the corresponding macro package. It can be specified on the command line as **−mandoc**. A *man*(1) librarian may use this macro file to delegate loading of the correct macro package; it is thus unnecessary for *man* itself to scan the contents of a document to decide the issue.

Many macro files augment the function of the full-service packages, or of *roff* documents that do not employ such a package—the latter are sometimes characterized as "raw". These auxiliary packages are described, along with details of macro file naming and placement, in *groff_tmac*(5).

Formatters

The formatter, the program that interprets *roff* language input, is *troff*(1). It provides the features of the AT&T *troff* and *nroff* programs as well as many extensions. The command-line option **−C** switches *troff* into *compatibility mode*, which tries to emulate AT&T *troff* as closely as is practical to enable the formatting of documents written for the older system.

A shell script, *nroff*(1), emulates the behavior of AT&T *nroff*. It attempts to correctly encode the output based on the locale, relieving the user of the need to specify an output device with the **−T** option and is therefore convenient for use with terminal output devices, described in the next subsection.

GNU *troff* generates output in a device-independent, but not device-agnostic, page description language detailed in *groff_out*(5).

Output devices

troff output is formatted for a particular *output device*, typically specified by the **−T** option to the formatter or a front end. If neither this option nor the *GROFF_TYPESETTER* environment variable is used, the default output device is **ps**. An output device may be any of the following.

ascii for terminals using the ISO 646 1991:IRV character set and encoding, also known as US-ASCII.

dvi for TeX DVI format.

html
xhtml for HTML and XHTML output, respectively.

latin1 for terminals using the ISO Latin-1 (8859-1) character set and encoding.

lbp for Canon CaPSL printers (LBP-4 and LBP-8 series laser printers).

lj4 for HP LaserJet4-compatible (or other PCL5-compatible) printers.

pdf for PDF output.

ps for PostScript output.

utf8 for terminals using the ISO 10646 (“Unicode”) character set in UTF-8 encoding.

X75 for previewing with *gxditview* using 75 dpi resolution and a 10-point base type size.

X75−12 for previewing with *gxditview* using 75 dpi resolution and a 12-point base type size.

X100 for previewing with *gxditview* using 100 dpi resolution and a 10-point base type size.

X100−12 for previewing with *gxditview* using 100 dpi resolution and a 12-point base type size.

Postprocessors

Any program that interprets the output of GNU *troff* is a postprocessor. The postprocessors provided by GNU *roff* are *output drivers*, which prepare a document for viewing or printing. Postprocessors for other purposes, such as page resequencing or statistical measurement of a document, are conceivable.

An output driver supports one or more output devices, each with its own device description file. A device determines its postprocessor with the **postpro** directive in its device description file; see *groff_font*(5). The **−X** option overrides this selection, causing *gxditview* to serve as the output driver.

grodvi(1)
: provides **dvi**.

grohtml(1)
: provides **html** and **xhtml**.

grolbp(1)
: provides **lbp**.

grolj4(1)
: provides **lj4**.

gropdf (1)
: provides **pdf**.

grops(1)
: provides **ps**.

grotty(1)
: provides **ascii**, **latin1**, and **utf8**.

gxditview(1)
: provides **X75**, **X75−12**, **X100**, and **X100−12**, and additionally can preview **ps**.

Utilities

GNU *roff* includes a suite of utilities.

gdiffmk(1)
: marks differences between a pair of *roff* input files.

grog(1)
: infers the *groff* command a document requires.

Several utilities prepare descriptions of fonts, enabling the formatter to use them when producing output for a given device.

addftinfo(1)
: adds information to AT&T *troff* font description files to enable their use with GNU *troff*.

afmtodit(1)
: creates font description files for PostScript Type 1 fonts.

pfbtops(1)
: translates a PostScript Type 1 font in PFB (Printer Font Binary) format to PFA (Printer Font ASCII), which can then be interpreted by *afmtodit*.

hpftodit(1)
: creates font description files for the HP LaserJet 4 family of printers.

tfmtodit(1)
: creates font description files for the TeX DVI device.

xtotroff(1)
: creates font description files for X Window System core fonts.

A trio of tools transform material constructed using *roff* preprocessor languages into graphical image files.

eqn2graph(1)
: converts an *eqn* equation into a cropped image.

grap2graph(1)
: converts a *grap* diagram into a cropped image.

pic2graph(1)
: converts a *pic* diagram into a cropped image.

Another set of programs works with the bibliographic data files used by the *refer*(1) preprocessor.

indxbib(1)
: makes inverted indices for bibliographic databases, speeding lookup operations on them.

lkbib(1)
: searches the databases.

lookbib(1)
: interactively searches the databases.

Exit status

groff exits successfully (with status **0**) if either of the options **−h** or **−−help** is specified, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation. Otherwise, *groff* runs a pipeline to process its input; if all commands within the pipeline exit successfully, *groff* does likewise. If not, *groff*'s exit status encodes a summary of problems encountered, setting bit 2 if a command exited with a failure status, bit 3 if a command was terminated with a signal, and bit 4 if a command could not be executed. (Thus, if all three misfortunes befall one's pipeline, *groff* exits with status 2^2 + 2^3 + 2^4 = 4+8+16 = 28.) To troubleshoot pipeline problems, re-run the *groff* command with the **−V** option and break the reported pipeline down into separate stages, inspecting the exit status of, and diagnostic messages emitted by, each command.

Environment

Environment variables in the host system affect the behavior of programs supplied by *groff* as follows. Normally, the path separator in environment variables ending with *PATH* is the colon; this may vary depending on the operating system. For example, Windows uses a semicolon instead.

GROFF_BIN_PATH
: Locate *groff* commands in these directories, followed by those in *PATH*. If not set, the installation directory of GNU *roff* executables, */usr/bin*, is searched before *PATH*.

GROFF_COMMAND_PREFIX
: Apply a prefix to certain GNU *roff* commands. *groff* can be configured at compile time to apply a prefix to the names of programs it provides that had counterparts in AT&T *troff*, so that name collisions are avoided at run time. The default prefix is empty.

  When used, this prefix is conventionally the letter "g". For example, GNU *troff* would be installed as *gtroff*. Besides *troff*, the prefix applies to the formatter wrapper *nroff*; the preprocessors *eqn*, *grn*, *pic*, *refer*, *tbl*, and *soelim*; and the utilities *indxbib* and *lookbib*.

GROFF_ENCODING
: Specify the assumed character encoding of the input. *groff* passes its value as an argument to *preconv*(1) preprocessor's **−e** option. This variable's existence implies the *groff* option **−k**. If set but empty, *groff* runs *preconv* without an **−e** option. *groff*'s **−K** option overrides *GROFF_ENCODING*.

GROFF_FONT_PATH
: Seek the selected output device's directory of device and font description files in this list of directories. See *troff*(1) and *groff_font*(5).

GROFF_TMAC_PATH
: Seek macro packages in this list of directories. See *troff*(1) and *groff_tmac*(5).

GROFF_TMPDIR
: Create temporary files in this directory. If not set, but *TMPDIR* is, the latter is used instead. On Windows systems, if neither of the foregoing are set, the environment variables *TMP* and *TEMP* (in that order) are checked also. Otherwise, temporary files are created in */tmp*. The *refer*(1), *grohtml*(1), and *grops*(1) commands use temporary files.

GROFF_TYPESETTER
: Set the default output device. If empty or not set, **ps** is used. The **−T** option overrides *GROFF_TYPESETTER*.

SOURCE_DATE_EPOCH
: Declare a time stamp (expressed as seconds since the Unix epoch) to use as the output creation time stamp in place of the current time. The time is converted to human-readable form using *gmtime*(3) and *asctime*(3) when the formatter starts up and stored in registers usable by documents and macro packages.

TZ
: Declare the time zone to use when converting the current time to human-readable form; see *tzset*(3). If *SOURCE_DATE_EPOCH* is used, it is always converted to human-readable form using UTC.

Examples

roff systems are best known for formatting man pages. A *man*(1) librarian program, having located a page, might render it with a *groff* command.

```
groff -t -man -Tutf8 /usr/share/man/man1/groff.1
```

The librarian will also pipe the output through a pager, which might not interpret terminal escape sequences *groff* emits for boldface, underlining, italics, or hyperlinking; see section "Limitations" below.

To process a *roff* input file using the preprocessors *tbl* and *pic* and the *me* macro package in the way to which AT&T *troff* users were accustomed, one would type (or script) a pipeline.

```
pic foo.me | tbl | troff -me -Tutf8 | grotty
```

Shorten this pipeline to an equivalent command using *groff*.

```
groff -p -t -me -T utf8 foo.me
```

An even easier way to do this is to use *grog*(1) to guess the preprocessor and macro options and execute the result by using the command substitution feature of the shell.

```
$(grog -Tutf8 foo.me)
```

Each command-line option to a postprocessor must be specified with any required leading dashes "**-**" because *groff* passes the arguments as-is to the postprocessor; this permits arbitrary arguments to be transmitted. For example, to pass a title to the *gxditview* postprocessor, the shell commands

```
groff -X -P -title -P 'trial run' mydoc.t
```

and

```
groff -X -Z mydoc.t | gxditview -title 'trial run' -
```

are equivalent.

Limitations

When paging output for the **ascii**, **latin1**, and **utf8** devices, programs like *more*(1) and *less*(1) may require command-line options to correctly handle some terminal escape sequences; see *grotty*(1).

Installation directories

GNU *roff* installs files in varying locations depending on its compile-time configuration. On this installation, the following locations are used.

/usr/share/X11/app-defaults
: Application defaults directory for *gxditview*(1).

/usr/bin
: Directory containing *groff*'s executable commands.

/usr/share/groff/1.24.2/eign
: List of common words for *indxbib*(1).

/usr/share/groff/1.24.2
: Directory for data files.

/usr/dict/papers/Ind
: Default index for *lkbib*(1) and *refer*(1).

/usr/share/doc/groff
: Documentation directory.

/usr/share/doc/groff/examples
: Example directory.

/usr/share/groff/1.24.2/font
: Font directory.

/usr/share/doc/groff/html
: HTML documentation directory.

/usr/lib/font
: Legacy font directory.

/etc/groff/site-font
: Local font directory.

/etc/groff/site-tmac
: Local macro package (*tmac* file) directory.

/usr/share/groff/1.24.2/tmac
: Macro package (*tmac* file) directory.

/usr/share/groff/1.24.2/oldfont
: Font directory for compatibility with old versions of *groff*; see *grops*(1).

/usr/share/doc/groff/pdf
: PDF documentation directory.

groff macro directory

Most macro files supplied with GNU *roff* are stored in */usr/share/groff/1.24.2/tmac* for the installation corresponding to this document. As a rule, multiple directories are searched for macro files; see *troff*(1). For a catalog of macro files GNU *roff* provides, see *groff_tmac*(5).

groff device and font description directory

Device and font description files supplied with GNU *roff* are stored in */usr/share/groff/1.24.2/font* for the installation corresponding to this document. As a rule, multiple directories are searched for device and font description files; see *troff*(1). For the formats of these files, see *groff_font*(5).

Availability

Obtain links to *groff* releases for download, its source repository, discussion mailing lists, a support ticket tracker, and further information from the *groff* page of the GNU website.

A free implementation of the *grap* preprocessor, written by Ted Faber, can be found at the *grap* website. *groff* supports only this *grap*.

Authors

groff (both the front-end command and the overall system) was primarily written by James Clark. Contributors to this document include Clark, Trent A. Fisher, Werner Lemberg, Bernd Warken, and G. Branden Robinson.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

A list of all *groff* man pages follows. A few (*grohtml*, *gropdf*, *gxditview*, and *xtotroff*) will be unavailable if their corresponding programs were disabled during compilation.

Introduction, history, and further reading:
: *roff*(7)

Viewer for *groff* (and AT&T device-independent *troff*) documents:
: *gxditview*(1)

Preprocessors:
: *chem*(1), *eqn*(1), *neqn*(1), *glilypond*(1), *grn*(1), *preconv*(1), *gperl*(1), *pic*(1), *gpinyin*(1), *refer*(1), *soelim*(1), *tbl*(1)

Macro packages:
: *groff_hdtbl*(7), *groff_man*(7), *groff_man_style*(7), *groff_mdoc*(7), *groff_me*(7), *groff_mm*(7), *groff_mmse*(7), *groff_mom*(7), *groff_ms*(7), *groff_rfc1345*(7), *groff_trace*(7), *groff_www*(7)

Bibliographic database management tools:
: *indxbib*(1), *lkbib*(1), *lookbib*(1)

Language, conventions, and GNU extensions:
: *groff*(7), *groff_char*(7), *groff_diff*(7), *groff_font*(5), *groff_tmac*(5)

Device-independent page description language:
: *groff_out*(5)

Formatter program:
: *troff*(1)

Formatter wrappers:
nroff (1), *mmroff* (1), *pdfmom*(1)

Postprocessors for output devices:
grodvi(1), *grohtml*(1), *grolbp*(1), *grolj4*(1), *gropdf* (1), *grops*(1), *grotty*(1)

Font support utilities:
addftinfo(1), *afmtodit*(1), *hpftodit*(1), *pfbtops*(1), *tfmtodit*(1), *xtotroff* (1)

Graphics conversion utilities:
eqn2graph(1), *grap2graph*(1), *pic2graph*(1)

Difference-marking utility:
gdiffmk(1)

“groff guess” utility:
grog(1)

Name

grog – "groff guess"—infer the *groff* command a document requires

Synopsis

grog [*groff-option* ...] [**--**] [*file* ...]

grog -h
grog --help

grog -v
grog --version

Description

grog reads its input and guesses which *groff*(1) options are needed to render it. If no operands are given, or if *file* is "**-**", *grog* reads the standard input stream. The corresponding *groff* command is normally written to the standard output stream.

Options

-h and **--help** display a usage message, whereas **-v** and **--version** display version information; all exit afterward.

All other specified short options (that is, arguments beginning with a minus sign "**-**" followed by a letter) are interpreted as *groff* options or option clusters with or without an option argument. Such options are included in the constructed *groff* command line.

Details

grog reads each *file* operand, pattern-matching strings that are statistically likely to be characteristic of *roff*(7) documents. It tries to guess which of the following *groff* options are required to correctly render the input: **-e**, **-g**, **-G**, **-j**, **-p**, **-R**, **-t** (preprocessors); and **-man**, **-mdoc**, **-mdoc-old**, **-me**, **-mm**, **-mom**, and **-ms** (macro packages). The inferred *groff* command including these options and any *file* parameters is written to the standard output stream.

It is possible to specify arbitrary *groff* options on the command line. These are included in the inferred command without change. Choices of *groff* options include **-C** to enable AT&T *troff* compatibility mode and **-T** to select a non-default output device. If the input is not encoded in ISO 646:1991 IRV (US-ASCII) or ISO Latin-1 (8859-1), we advise specifying a *groff* option to run *preconv*(1); see the **-D**, **-k**, and **-K** options of *groff*(1). For UTF-8 input, **-k** is a good choice.

groff may issue diagnostic messages when an inappropriate **-m** option, or multiple conflicting ones, are specified. Consequently, it is best to specify no **-m** options to *grog* unless it cannot correctly infer all of the **-m** arguments a document requires. A *roff* document can also be written without recourse to any macro package. In such cases, *grog* infers a *groff* command without an **-m** option.

Limitations

grog presumes that the input does not change the escape, control, or no-break control characters. *grog* does not parse *roff* input line continuation or control structures (brace escape sequences and the "**if**", "**ie**", and "**el**" requests) nor *groff*'s "**while**". Thus the input

```
.if \
t .NH 1
.if n .SH
Introduction
```

conceals the use of the *ms* macros **NH** and **SH** from *grog*. Such constructions are regarded by *grog*'s implementors as insufficiently common to cause many inference problems. Preprocessors can be even stricter when matching macro calls that bracket the regions of an input file they replace. *pic*, for example, requires **PS**, **PE**, and **PF** calls to immediately follow the default control character at the beginning of a line.

Detection of the **-s** option (the *soelim*(1) preprocessor) is tricky; to correctly infer its necessity would require *grog* to recursively open all files given as arguments to the **.so** request under the same conditions that *soelim* itself does so; see its man page. Recall that *soelim* is necessary only if sourced files need to be preprocessed. Therefore, as a workaround, you may want to run the input through *soelim* manually, piping it to *grog*, and

compare the output to running *grog* on the input directly. If the "*soelim*"ed input causes *grog* to infer additional preprocessor options, then **–s** is likely necessary.

```
$ printf ".TS\nl.\nI'm a table.\n.TE\n" > 3.roff
$ printf ".so 3.roff\n" > 2.roff
$ printf ".XP\n.so 2.roff\n" > 1.roff
$ grog 1.roff
groff -ms 1.roff
$ soelim 1.roff | grog
groff -t -ms -
```

In the foregoing example, we see that this procedure enabled *grog* to detect *tbl*(1) macros, so we would add **–s** as well as the detected **–t** option to a revised *grog* or *groff* command.

```
$ grog -st 1.roff
groff -st -ms 1.roff
```

Exit status

grog exits with status **1** if a macro package appears to be in use by the input document, but *grog* was unable to infer which one, or **2** if there were problems handling an option or operand. It otherwise exits with status **0**. Inferring no preprocessors or macro packages is not an error condition; a valid *roff* document need not use either. Even plain text is valid input, if one is mindful of the syntax of the control and escape characters.

Examples

Running

```
grog /usr/share/doc/groff/meintro.me
```

at the command line results in

```
groff -me /usr/share/doc/groff/meintro.me
```

because *grog* recognizes that the file *meintro.me* is written using macros from the *me* package. The command

```
grog /usr/share/doc/groff/pic.ms
```

outputs

```
groff -e -p -t -ms /usr/share/doc/groff/pic.ms
```

on the other hand. Besides discerning the *ms* macro package, *grog* recognizes that the file *pic.ms* additionally needs the combination of **–t** for *tbl*, **–e** for *eqn*, and **–p** for *pic*.

Consider a file *doc/grnexampl.me*, which uses the *grn* preprocessor to include a *gremlin*(1) picture file in an *me* document. Let's say we want to suppress color output, produce a DVI file, and get backtraces for any errors that *troff* encounters. The command

```
grog -bc -Idoc -Tdvi doc/grnexmpl.me
```

is processed by *grog* into

```
groff -bc -Idoc -Tdvi -e -g -me doc/grnexmpl.me
```

where we can see that *grog* has inferred the *me* macro package along with the *eqn* and *grn* preprocessors. (The input file is located in */usr/share/doc/groff* if you'd like to try this example yourself.)

Authors

grog was originally written in Bourne shell by James Clark. The current implementation in Perl was written by Bernd Warken and heavily revised by G. Branden Robinson.

See also

groff(1)

Name

grohtml, post−grohtml, pre−grohtml − *groff* output driver for HTML

Synopsis

pre−grohtml [**−epV**] [**−a** *anti-aliasing-text-bits*] [**−D** *image-directory*] [**−F** *font-directory*] [**−g** *anti-aliasing-graphic-bits*] [**−i** *resolution*] [**−I** *image-stem*] [**−o** *image-vertical-offset*] [**−x** *html-dialect*] *troff-command troff-argument* . . .

pre−grohtml −−help

pre−grohtml −v
pre−grohtml −−version

post−grohtml [**−bCGhlnrVy**] [**−F** *font-directory*] [**−j** *output-stem*] [**−k** *encoding*] [**−s** *base-point-size*] [**−S** *heading-level*] [**−x** *html-dialect*] [*file* . . .]

post−grohtml −−help

post−grohtml −v
post−grohtml −−version

Description

The GNU *roff* system's HTML support consists of a preprocessor, *pre−grohtml*, and an output driver, *post−grohtml*; together, they translate *roff*(7) documents to HTML. Because a preprocessor is (uniquely) required for this output driver, users should invoke *grohtml* via the *groff*(1) command with the **−Thtml** or **−Txhtml** options. (In this installation, **ps** is the default output device.) Use *groff*'s **−P** option to pass any options shown above to *grohtml*. If no operands are given, or if *file* is "**−**", *grohtml* reads the standard input stream. It writes to the standard output stream.

grohtml invokes *groff* twice. In the first pass, the preprocessor *pre−grohtml* renders pictures, equations, and tables as images in PostScript format using the **ps** output device. In the second pass, the output driver *post−grohtml* translates the output of *troff*(1) to HTML.

grohtml writes UTF-8-encoded output (but see the **−k** option) and produces HTML character references for most non-composite, non-basic Latin Unicode characters. In spite of this, *groff* may issue warnings about unknown special characters if they can't be found during the first pass. You can ignore these warnings unless the special characters appear inside a table or equation.

Typefaces

grohtml supports the standard four styles: **R** (roman), **I** (*italic*), **B** (**bold**), and **BI** (***bold-italic***). Fonts are grouped into families **T** and **C** having members in each style.

TR	Times roman
TI	Times italic
TB	Times bold
TBI	Times bold-italic
CR	Courier roman
CI	Courier italic
CB	Courier bold
CBI	Courier bold-italic

A special font, **S**, is also provided to accommodate *roff* documents that expect it to always be available.

grohtml furthermore supports a naming scheme for East Asian typefaces shared with *gropdf*(1), *grops*(1), and *grotty*(1).

CSH	Simplified Chinese, Hei style
CSS	Simplified Chinese, Song style
CTH	Traditional Chinese, Hei style

CTS Traditional Chinese, Song style
JPG Japanese, Gothic style
JPM Japanese, Mincho style
KOG Korean, Gothic style
KOM Korean, Mincho style

Font description files

The font description files used with *grohtml* expose the same glyph repertoire in their **charset** sections. See *groff_font*(5).

Dependencies

pre-grohtml generates an image whenever an *eqn* equation, *tbl* table, or *pic* picture is encountered in the input. *grohtml* therefore may run several commands as part of its operation. These include the Netpbm tools *pamcut*, *pnmcrop*, and *pnmtopng*, as well as Ghostscript's *gs* and *ps2ps*.

Options

--help displays a usage message, while **-v** and **--version** show version information; all exit afterward.

-a *anti-aliasing-text-bits*
Number of bits of antialiasing information to be used by text when generating PNG images. The default is **4** but **0**, **1**, and **2** are also valid. Your system's version of *gs* must support the **-dTextAlphaBits** option in order to exploit antialiasing. A value of **0** stops *grohtml* from issuing antialiasing commands to *gs*.

-b Initialize the background color to white.

-C Suppress output of "CreationDate:" HTML comment.

-D *image-directory*
Instruct *grohtml* to place all image files into directory *image-directory*.

-e Direct *eqn* to produce MathML.

This option should not be manually specified; it is synthesized by *groff* depending on whether it was given the **-Thtml** or **-Txhtml** option.

-F *font-directory*
Prepend directory font-directory*/dev*name to the search path for font and device description files; *name* is the name of the device, usually **html**.

-g *anti-aliasing-graphic-bits*
Number of bits of antialiasing information to be used by graphics when generating PNG images. The default is **4** but **0**, **1**, and **2** are also valid. Your system's version of *gs* must support the **-dGraphicAlphaBits** option in order to exploit antialiasing. A value of **0** stops *grohtml* from issuing antialiasing commands to *gs*.

-G Suppress output of "Creator:" HTML comment.

-h Generate section headings by using HTML **B** elements and increasing the font size, rather than HTML **H** elements.

-i *resolution*
Set the image resolution in pixels per inch; the default is **100**.

-I *image-stem*
Determine the image file name stem. If omitted, *grohtml* uses *grohtml*-XXXXX (where *XXXXX* is the process ID). A dash is appended to the stem to separate it from the following image number.

-j *output-stem*
Instruct *grohtml* to split the HTML output into multiple files. Output is written to a new file at each section heading (but see option **-S** below) named *output-stem*-n.*html*.

−k *encoding*
: Select the character encoding used in the generated document, affecting the declared encoding in the preamble and the form of character entity references. Valid values are “ASCII” and “UTF−8”. The default is “UTF-8”.

−l
: Turn off the production of automatic section links at the top of the document.

−n
: Generate simple heading anchors whenever a section/number heading is found. Without the option the anchor value is the textual heading. This can cause problems when a heading contains a “?” on older versions of some browsers. This feature is automatically enabled if a heading contains an image.

−o *image-vertical-offset*
: Specify the vertical offset of images in points.

−p
: Display page rendering progress to the standard error stream. *grohtml* displays a page number only when an image is required.

−r
: Turn off the automatic header and footer line (HTML rule).

−s *base-type-size*
: Set the document’s base type size in points. When this size is used in the source, it corresponds to the HTML base type size. Every increase of two points in the source will produce a “**big**” element, and conversely when a decrease of two points is seen, a “**small**” element is emitted.

−S *heading-level*
: When splitting HTML output (see option **−j** above), split at each nested heading level defined by *heading-level*, or higher). The default is **1**.

−V
: Create an XHTML or HTML validator button at the bottom of each page of the document.

−x *html-dialect*
: Select HTML dialect. Currently, *html-dialect* should be either the digit **4** or the letter **x**, which indicates whether *grohtml* should generate HTML 4 or XHTML, respectively.

  This option should not be manually specified; it is synthesized by *groff* depending on whether it was given the **−Thtml** or **−Txhtml** option.

−y
: Produce a right-aligned *groff* signature at the end of the document (only if **−V** is also specified).

Exit status

pre−grohtml and *post−grohtml* each exit with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Environment

GROFF_FONT_PATH
: lists directories in which to search for *devhtml*, *grohtml*’s directory of device and font description files. See *troff*(1) and *groff_font*(5).

SOURCE_DATE_EPOCH
: A timestamp (expressed as seconds since the Unix epoch) to use as the output creation timestamp in place of the current time. The time is converted to human-readable form using *gmtime*(3) and *asctime*(3), and recorded in an HTML comment.

TZ
: The time zone to use when converting the current time to human-readable form; see *tzset*(3). If *SOURCE_DATE_EPOCH* is used, it is always converted to human-readable form using UTC.

Files

/usr/share/groff/1.24.2/font/devhtml/DESC
: describes the **html** output device.

*/usr/share/groff/1.24.2/font/devhtml/*F
: describes the font known as *F* on device **html**.

/usr/share/groff/1.24.2/tmac/html.tmac
: defines font mappings, special characters, and colors for use with the **html** output device. It is automatically loaded by *troffrc* when either of the **html** or **xhtml** output devices is selected.

/usr/share/groff/1.24.2/tmac/html−end.tmac
: finalizes setup of the **html** output device. It is automatically loaded by *troffrc−end* when either of the **html** or **xhtml** output devices is selected.

grohtml uses temporary files. See *groff* (1) for details about where such files are created.

Bugs

grohtml is still beta code.

grohtml does not truly support hyphenation, but you can fool it into hyphenating long input lines, which can appear in HTML output with a hyphenated word followed by a space but no line break.

See also

groff (1), *troff* (1), *groff_font*(5)

Name

grolbp – *groff* output driver for Canon CaPSL printers

Synopsis

grolbp [**–l**] [**–c** *num-copies*] [**–F** *font-directory*] [**–o** *orientation*] [**–p** *paper-format*] [**–w** *width*] [*file* ...]
grolbp [**––copies=***num-copies*] [**––fontdir=***font-directory*] [**––landscape**] [**––linewidth=***width*]
[**––orientation=***orientation*] [**––papersize=***paper-format*] [*file* ...]

grolbp –h
grolbp ––help

grolbp –v
grolbp ––version

Description

This GNU *roff* output driver translates the output of *troff*(1) into a CaPSL and VDM format suitable for Canon LBP-4 and LBP-8 printers. Normally, *grolbp* is invoked by *groff*(1) when the latter is given the "**–T lbp**" option. (In this installation, **ps** is the default output device.) Use *groff*'s **–P** option to pass any options shown above to *grolbp*. If no *file* arguments are given, or if *file* is "–", *grolbp* reads the standard input stream. It writes to the standard output stream.

Typefaces

The driver supports the Dutch, Swiss, and Swiss-Narrow scalable typefaces, each in the regular, bold, italic, and bold-italic styles. Additionally, the bitmapped, monospaced Courier and Elite typefaces are available in regular, bold, and italic styles; Courier at 8 and 12 points, Elite at 8 and 10 points. The following chart summarizes the *groff* font names used to access them.

Typeface	Roman	Bold	Italic	Bold-Italic
Dutch	TR	TB	TI	TBI
Swiss	HR	HB	HI	HBI
Swiss Narrow	HNR	HNB	HNI	HNBI
Courier	CR	CB	CI	
Elite	ER	EB	EI	

Paper format, orientation, and device description file

grolbp supports paper formats "**A4**", "**letter**", "**legal**", and "**executive**". These are matched case-insensitively. The **–p**, **––papersize** option overrides any setting in the device description file *DESC*. If neither specifies a paper format, A4 is assumed.

In its *DESC* file, *grolbp* (case-insensitively) recognizes an **orientation** directive accepting one mandatory argument, **portrait** or **landscape**. The first valid orientation directive encountered controls. The **–l**, **–o**, and **––orientation** command-line options override any setting in *DESC*. If none of the foregoing specify the orientation, portrait is assumed.

Font description files

In addition to the font description file directives documented in *groff_font*(5), *grolbp* recognizes **lbpname**, which maps the *groff* font name to the font name used internally by the printer. Its syntax is as follows.

```
lbpname printer-font-name
```

lbpname's argument is case-sensitive. The printer's font names are encoded as follows.

For bitmapped fonts, *printer-font_name* has the form **N**⟨*base-font-name*⟩⟨*font-style*⟩. *base-font-name* is the font name as it appears in the printer's font listings without the first letter, up to (but not including) the font size. *font-style* can be one of the letters **R**, **I**, or **B**, indicating the roman, italic, and bold styles, respectively. For instance, if the printer's "font listing A" shows "Nelite12I.ISO_USA", the corresponding entry in the *groff* font description file is "**lbpname NeliteI**". You may need to modify *grolbp* to add support for new bitmapped fonts, since the available font names and font sizes of bitmapped fonts (as documented above) are hard-coded into the program.

For scalable fonts, *printer-font-name* is identical to the font name as it appears in the printer's "font listing A". For instance, to select the "Swiss" font in bold-italic style, which appears in the font listing as "Swiss−BoldOblique", "**lbpname Swiss−BoldOblique**" is the required directive, and this is what we find in the *groff* font description file *HBI* for the **lbp** device.

Drawing commands

For compatibility with *grolj4*(1), an additional drawing command is available.

\D'R *dh dv*'
: Draw a rule (solid black rectangle) with one corner at the drawing position, and the diagonally opposite corner at the drawing position +(*dh*,*dv*).

Options

−h and **−−help** display a usage message, while **−v** and **−−version** show version information; all exit afterward.

−c *num-copies*
−−copies=*num-copies*
: Produce *num-copies* copies of each page.

−F *font-directory*
−−fontdir= *font-directory*
: Prepend directory font-directory*/dev*name to the search path for font and device description files; *name* is the name of the device, usually **lbp**.

−l
−−landscape
: Format the document in landscape orientation.

−o *orientation*
−−orientation=*orientation*
: Format the document in the given *orientation*, which must be " **portrait**" or "**landscape**".

−p *paper-format*
−−papersize=*paper-format*
: Set the paper format to *paper-format*, which must be a valid paper format as described above.

−w *width*
−−linewidth=*width*
: Set the default line thickness to *width* thousandths of an em; the default is **40** (0.04 em).

Exit status

grolbp exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Environment

GROFF_FONT_PATH
: lists directories in which to seek the selected output device's directory of device and font description files. See *troff*(1) and *groff_font*(5).

Files

/usr/share/groff/1.24.2/font/devlbp/DESC
: describes the **lbp** output device.

*/usr/share/groff/1.24.2/font/devlbp/*F
: describes the font known as *F* on device **lbp**.

/usr/share/groff/1.24.2/tmac/lbp.tmac
: defines macros for use with the **lbp** output device. It is automatically loaded by *troffrc* when the **lbp** output device is selected.

See also

groff(1), *troff*(1), *groff_out*(5), *groff_font*(5), *groff_char*(7)

Name

grolj4 − *groff* output driver for HP LaserJet 4 and compatible printers

Synopsis

grolj4 [**−l**] [**−c** *num-copies*] [**−d** [*n*]] [**−F** *font-directory*] [**−p** *paper-format*] [**−w** *line-width*] [*file* . . .]

grolj4 −−help

grolj4 −v
grolj4 −−version

Description

This GNU *roff* output driver translates the output of *troff*(1) into a PCL5 format suitable for a Hewlett-Packard LaserJet 4 printer. Normally, *grolj4* is invoked by *groff*(1) when the latter is given the "**−T lj4**" option. (In this installation, **ps** is the default output device.) Use *groff*'s **−P** option to pass any options shown above to *grolj4*. If no *file* arguments are given, or if *file* is "−", *grolj4* reads the standard input stream. It writes to the standard output stream.

Typefaces

grolj4 supports the standard four styles: **R** (roman), **I** (*italic*), **B** (**bold**), and **BI** (***bold-italic***). Fonts are grouped into families **A**, **C**, **G**, **O**, **T**, **TN**, **U**, and **UC** having members in each style. For the convenience of a consistent font repertoire in *groff*, *lj4.tmac* (see section "Files" below) uses **ftr** requests to remap **H** ("Helvetica") font names to the **U** names shown below.

AB	Arial Bold
ABI	Arial Bold Italic
AI	Arial Italic
AR	Arial Roman
CB	Courier Bold
CBI	Courier Bold Italic
CI	Courier Italic
CR	Courier Roman
GB	Garamond Halbfett
GBI	Garamond Kursiv Halbfett
GI	Garamond Kursiv
GR	Garamond Antiqua
OB	CG Omega Bold
OBI	CG Omega Bold Italic
OI	CG Omega Italic
OR	CG Omega Roman
OB	CG Omega Bold
OBI	CG Omega Bold Italic
OI	CG Omega Italic
OR	CG Omega Roman
TB	CG Times Bold
TBI	CG Times Bold Italic
TI	CG Times Italic
TR	CG Times Roman
TNRB	M Times Bold
TNRBI	M Times Bold Italic
TNRI	M Times Italic
TNRR	M Times Roman
UB	Univers Bold
UBI	Univers Bold Italic
UI	Univers Medium Italic

UR	Univers Medium
UCB	Univers Condensed Bold
UCBI	Univers Condensed Bold Italic
UCI	Univers Condensed Medium Italic
UCR	Univers Condensed Medium

The following fonts are not members of a family.

ALBB	Albertus Extra Bold
ALBR	Albertus Medium
AOB	Antique Olive Bold
AOI	Antique Olive Italic
AOR	Antique Olive Roman
CLARENDON	Clarendon
CORONET	Coronet
LGB	Letter Gothic Bold
LGI	Letter Gothic Italic
LGR	Letter Gothic Roman
MARIGOLD	Marigold

The special font is **S** (PostScript Symbol); **SYMBOL** (M Symbol), and **WINGDINGS** (Wingdings) are also available but not mounted by default.

Paper format and device description file

grolj4 supports paper formats "**A4**", "**B5**", "**C5**", "**com10**", "**DL**", "**executive**", "**legal**", "**letter**", and "**monarch**". These are matched case-insensitively. The **−p** option overrides any setting in the device description file *DESC*. If neither specifies a paper format, "letter" is assumed.

Font description files

grolj4 recognizes four font description file directives in addition to those documented in *groff_font*(5).

pclweight *n*
: Set the stroke weight to *n*, an integer in the range −7 to +7; the default is 0.

pclstyle *n*
: Set the style to *n*, an integer in the range 0 to 32767; the default is 0.

pclproportional *n*
: Set the proportional spacing Boolean flag to *n*, which can be either 0 or 1; the default is 0.

pcltypeface *n*
: Set the typeface family to *n*, an integer in the range 0 to 65535; the default is 0.

Drawing commands

An additional drawing command is recognized as an extension to those documented in *groff*(7).

\D'R *dh dv*'
: Draw a rule (solid black rectangle) with one corner at the drawing position, and the diagonally opposite corner at the drawing position +(*dh*,*dv*), at which the drawing position will be afterward. This generates a PCL fill rectangle command, and so will work on printers that do not support HP-GL/2, unlike the other **\D** commands.

Fonts

Nominally, HP LaserJet 4-series and newer printers have the same internal fonts. 45 fonts are scalable from 0.25 to 999.75 points in 0.25-point increments. A Lineprinter font is available only at 8.5 points.

The LaserJet font files included with *groff* assume that all printers since the LaserJet 4 are identical. There are some differences between fonts in the earlier and more recent printers, however. The LaserJet 4 printer used Agfa Intellifont technology for 35 of the internal scalable fonts; the remaining 10 scalable fonts were TrueType. Beginning with the LaserJet 4000-series printers introduced in 1997, all scalable internal fonts have been TrueType. The number of printable glyphs differs slightly between Intellifont and TrueType fonts

(generally, the TrueType fonts include more glyphs), and there are some minor differences in glyph metrics. Differences among printer models are described in the *PCL 5 Comparison Guide* and the *PCL 5 Comparison Guide Addendum* (for printers introduced since approximately 2001).

LaserJet printers reference a glyph by a combination of a 256-glyph symbol set and an index within that symbol set. Many glyphs appear in more than one symbol set; all combinations of symbol set and index that reference the same glyph are equivalent. For each glyph, *hpftodit*(1) searches a list of symbol sets, and selects the first set that contains the glyph. The printing code generated by *hpftodit* is an integer that encodes a numerical value for the symbol set in the high byte(s), and the index in the low byte. See *groff_font*(5) for a complete description of the font file format; symbol sets are described in greater detail in the *PCL 5 Printer Language Technical Reference Manual*.

Two of the scalable fonts, Symbol and Wingdings, are bound to 256-glyph symbol sets; the remaining scalable fonts, as well as the Lineprinter font, support numerous symbol sets, sufficient to enable printing of more than 600 glyphs.

The metrics generated by *hpftodit* assume that the *DESC* file contains values of 1200 for *res* and 6350 for *unitwidth*, or any combination (e.g., 2400 and 3175) for which *res* × *unitwidth* = 7 620 000. Although HP PCL 5 LaserJet printers support an internal resolution of 7200 units per inch, they use a 16-bit signed integer for positioning; if **devlj4** is to support U.S. ledger paper (11 in × 17 in; in = inch), the maximum usable resolution is 32 767 ÷ 17, or 1927 units per inch, which rounds down to 1200 units per inch. If the largest required paper dimension is less (e.g., 8.5 in × 11 in, or A5), a greater *res* (and lesser *unitwidth*) can be specified.

Font metrics for Intellifont fonts were provided by Tagged Font Metric (TFM) files originally developed by Agfa/Compugraphic. The TFM files provided for these fonts supported 600+ glyphs and contained extensive lists of kerning pairs.

To accommodate developers who had become accustomed to TFM files, HP also provided TFM files for the 10 TrueType fonts included in the LaserJet 4. The TFM files for TrueType fonts generally included less information than the Intellifont TFMs, supporting fewer glyphs, and in most cases, providing no kerning information. By the time the LaserJet 4000 printer was introduced, most developers had migrated to other means of obtaining font metrics, and support for new TFM files was very limited. The TFM files provided for the TrueType fonts in the LaserJet 4000 support only the ISO Latin-2 (8859-2) symbol set, and include no kerning information; consequently, they are of little value for any but the most rudimentary documents.

Because the Intellifont TFM files contain considerably more information, they generally are preferable to the TrueType TFM files even for use with the TrueType fonts in the newer printers. The metrics for the TrueType fonts are very close, though not identical, to those for the earlier Intellifont fonts of the same names. Although most output using the Intellifont metrics with the newer printers is quite acceptable, a few glyphs may fail to print as expected. The differences in glyph metrics may be particularly noticeable with composite parentheses, brackets, and braces used by *eqn*(1). A script, located in */usr/share/groff/1.24.2/font/devlj4/generate*, can be used to adjust the metrics for these glyphs in the special font "S" for use with printers that have all TrueType fonts.

At the time HP last supported TFM files, only version 1.0 of the Unicode standard was available. Consequently, many glyphs lacking assigned code points were assigned by HP to the Private Use Area (PUA). Later versions of the Unicode standard included code points outside the PUA for many of these glyphs. The HP-supplied TrueType TFM files use the PUA assignments; TFM files generated from more recent TrueType font files require the later Unicode values to access the same glyphs. Consequently, two different mapping files may be required: one for the HP-supplied TFM files, and one for more recent TFM files.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−c *num-copies*

Format *num-copies* copies of each page.

−d [*n*] Use duplex mode *n*: 1 is long-side binding (default), and 2 is short-side binding.

−F *font-directory*
Prepend directory *font-directory*/dev*name* to the search path for font and device description files; *name* is the name of the device, usually **lj4**.

−l Format the document in landscape orientation.

−p *paper-format*
Set the paper format to *paper-format*, which must be a valid paper format as described above.

−w *line-width*
Set the default line thickness to *line-width* thousandths of an em; the default is **40** (0.04 em).

Exit status

grolj4 exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Environment

GROFF_FONT_PATH
lists directories in which to seek the selected output device's directory of device and font description files. See *troff*(1) and *groff_font*(5).

Files

/usr/share/groff/1.24.2/font/devlj4/DESC
describes the **lj4** output device.

*/usr/share/groff/1.24.2/font/devlj4/*F
describes the font known as *F* on device **lj4**.

/usr/share/groff/1.24.2/tmac/lj4.tmac
defines macros for use with the **lj4** output device. It is automatically loaded by *troffrc* when the **lj4** output device is selected.

Bugs

Small dots.

See also

HP PCL/PJL Reference: PCL 5 Printer Language Technical Reference Manual, Part I

hpftodit(1), *groff*(1), *troff*(1), *groff_out*(5), *groff_font*(5), *groff_char*(7)

Name

gropdf – *groff* output driver for Portable Document Format

Synopsis

gropdf [**–delsW**] [{**–f**|**––format-options**} *bit-vector*] [**–F** *font-directory*] [**–I** *inclusion-directory*]
[**–p** *paper-format*] [**––pdfver** {**1.4**|**1.7**}] [**–u** [*cmap-file*]] [**–y** *foundry*] [*file* ...]

gropdf ––help

gropdf –v
gropdf ––version

Description

The GNU *roff* PDF output driver translates the output of *troff*(1) into Portable Document Format. Normally, *gropdf* is invoked by *groff*(1) when the latter is given the "**–T pdf**" option. (In this installation, **ps** is the default output device.) Use *groff*'s **–P** option to pass any options shown above to *gropdf*. If no *file* arguments are given, or if *file* is "–", *gropdf* reads the standard input stream. It writes to the standard output stream.

See section "Font installation" below for a guide to installing fonts for *gropdf*.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–d Include debug information as comments within the PDF. Also produces an uncompressed PDF.

–e Forces *gropdf* to embed *all* fonts (even the 14 base PDF fonts).

––format–options *bit-vector*
–f *bit-vector*

Specify advanced options for *gropdf*. Familiarity with the ISO 32000 PDF standard is helpful. The *bit-vector* argument is an integer that configures characteristics of the generated PDF. Add the following values to combine them.

Value	Meaning
1	Subset included Type 1 fonts.
2	Use more compact format for text by including space as a character. Fonts that do not include space as a glyph may conflict with this feature.
4	Compress all data streams.
8	Don't embed font files. (A font required by the document is not embedded; usually not useful.)

The default feature combination is 7. To mimic what *gropdf* from *groff* 1.23 produced, specify "6" to turn off subsetting.

–F *dir* Prepend directory *dir*/dev*name* to the search path for font, and device description files; *name* is the name of the device, usually **pdf**.

–I *dir* Search the directory *dir* for files named in **\X'pdf: pdfpic'** device extension commands. **–I** may be specified more than once; each *dir* is searched in the given order. To search the current working directory before others, add "**–I .**" at the desired place; it is otherwise searched last.

–l Orient the document in landscape format.

–p *paper-format*

Set the physical dimensions of the output medium. This overrides the **papersize**, **paperlength**, and **paperwidth** directives in the *DESC* file; it accepts the same arguments as the **papersize** directive. See *groff_font*(5) for details.

––pdfver {**1.4**|**1.7**}

PDF version 1.7 introduced a more compact object format; this is now the default. If you require the original format (as produced by *gropdf* 1.23) set the version to **1.4**.

−s Append a comment line to end of PDF showing statistics, i.e. number of pages in document. Ghostscript's **ps2pdf** complains about this line if it is included, but works anyway.

−u [*cmap-file*]
gropdf normally includes a ToUnicode CMap with any font created using *text.enc* as the encoding file, this makes it easier to search for words which contain ligatures. You can include your own CMap by specifying a *cmap-file* or have no CMap at all by omitting the argument.

−W Exit with failure status if any warnings are issued.

−y *foundry*
Set the foundry to use for selecting fonts of the same name.

Usage

gropdf's input must be in the format produced by *troff*(1) and described in *groff_out*(5). Further, its device and font description files must meet certain requirements. The device **resolution** must be an integer multiple of 72 times **sizescale**. By default, *gropdf* uses a **resolution** of 72000 and a **sizescale** of 1000. A valid paper format is mandatory; see *groff_font*(5). While the PDF standard allows several font file formats (like TrueType), at present *gropdf* accepts only the same Type 1 Adobe PostScript format as *grops*(1). Fewer Type 1 fonts are supported natively in PDF documents than the standard 35 fonts supported by *grops* and PostScript printers, but all are available since *gropdf* automatically embeds any that aren't specified by the PDF standard.

gropdf supports *foundries* that permit multiple providers to supply the same *groff* font names. *groff*'s compilation process attempts to locate Type 1 fonts on the system, populates a *Foundry* file with their locations, and generates font description files corresponding to them. Font description files can also be added after installation. Each such file must contain a directive

internalname *psname*

that maps the *groff* font name (such as "TR") to a PostScript name (such as "Times−Roman"). Lines starting with **#** and blank lines are ignored. The code for each character given in the font file must correspond to the code in the default encoding for the font. This code can be used with the **\N** escape sequence in *troff* to select the character even if it lacks a special character name. Every character in the font description must exist in the font file, and the widths given in the description must match those used in the font file. See *groff_font*(5).

gropdf can automatically *embed* any downloadable fonts necessary to print the document. Any fonts thus required must be listed in the file */usr/share/groff/1.24.2/font/devpdf/download*, which should comprise lines of the form

foundry font file-name

where *foundry* is the foundry name, or blank for the default foundry; *font* is the PostScript name of the font, and *file-name* is the name of the PFA or PFB font file, and can be a pathname (can contain slashes). Any lines beginning with **#** and blank lines are ignored; fields must be separated by tabs (spaces are **not** allowed); if *file-name* is not a pathname, it is sought using the same mechanism as that used for font metric files. The *download* file itself is also sought using this mechanism. Foundry names are usually a single character (such as 'U' for the URW foundry) or empty for the default foundry. This default uses the same fonts as *Ghostscript* uses when it embeds fonts in a PDF file.

The default stroke and fill colors are black.

Typefaces

Styles called **R**, **I**, **B**, and **BI** mounted at font positions 1 to 4. Text fonts are grouped into families **A**, **BM**, **C**, **H**, **HN**, **N**, **P**, and **T**, each having members in each of these styles.

AR AvantGarde-Book
AI *AvantGarde-BookOblique*
AB **AvantGarde-Demi**
ABI ***AvantGarde-DemiOblique***
BMR Bookman-Light

BMI *Bookman-LightItalic*
BMB **Bookman-Demi**
BMBI ***Bookman-DemiItalic***
CR Courier
CI *Courier-Oblique*
CB **Courier-Bold**
CBI ***Courier-BoldOblique***
HR Helvetica
HI *Helvetica-Oblique*
HB **Helvetica-Bold**
HBI ***Helvetica-BoldOblique***
HNR Helvetica-Narrow
HNI *Helvetica-Narrow-Oblique*
HNB **Helvetica-Narrow-Bold**
HNBI ***Helvetica-Narrow-BoldOblique***
NR NewCenturySchlbk-Roman
NI *NewCenturySchlbk-Italic*
NB **NewCenturySchlbk-Bold**
NBI ***NewCenturySchlbk-BoldItalic***
PR Palatino-Roman
PI *Palatino-Italic*
PB **Palatino-Bold**
PBI ***Palatino-BoldItalic***
TR Times-Roman
TI *Times-Italic*
TB **Times-Bold**
TBI ***Times-BoldItalic***

Another text font is not a member of a family.

ZCMI *ZapfChancery-MediumItalic*

Special fonts include **S**, the PostScript Symbol font; **SS**, a subset of **S** with slanted lowercase Greek letters; **EURO**, which offers a Euro glyph in several styles for use with old devices lacking it; and **ZD**, Zapf Dingbats. In contrast to *grops*, *gropdf* does not require a reversed variant of it (**ZDR**); the "hand pointing left" glyph (**\[lh]**) is available nevertheless, since *pdf.tmac* defines it using the **\X'pdf: xrev'** device extension command (see below). Some glyphs in these fonts are unnamed and must be accessed as indexed characters, using the **\N** escape sequence.

The fonts corresponding to **EURO** and **SS** are unknown to the PDF standard; *groff* therefore provides their AFM files (font metrics) and PFA or PFB files so that they can be used with other software and embedded in PDF output.

Feature service levels and URW font support

The traditional PostScript Type 1 fonts are limited in their glyph repertoire, and the original versions from the Adobe foundry are not free software. Historically, because their presence was mandated by the PostScript standard, one could expect to find support for them in any conforming device or software PostScript renderer. PostScript ("Level 1") initially standardized 14 typefaces: Times, Helvetica, and Courier each in four styles (which *groff* groups into "families"); a symbol font; and a dingbats font. PostScript Level 2 increased the number to 35, adding the families Avant Garde, Bookman, Helvetica Narrow, New Century Schoolbook, and Palatino; and a text font in one style, Zapf Chancery medium italic. A document could be small because it did not need to embed font resources unless it had unusual (for the time) glyph or typeface requirements. This situation carried over into the early years of PostScript's successor page description language, PDF. Nowadays, it is common to embed fonts in PDFs, and authorities widely recommend this practice, which increases the reliability of document rendering, and many free software fonts are available with much greater glyph coverage than Adobe's Type 1 fonts for PostScript.

gropdf attempts to work in variety of scenarios, and delivers better results when configured with supporting digital font files (for embedding) and font metrics files describing those fonts to the formatter.

- *Full service* is available when *gropdf* can locate all 35 fonts of the PostScript Level 2 standard on the file system along with their corresponding font metrics (AFM) files. The Adobe-compatible unnamed (default) foundry supports up to 256 glyphs in each typeface. Fonts from the URW foundry ("U") are compatible extensions of the Adobe fonts with extended glyph coverage, including support for Cyrillic script. *groff*'s build process uses *afmtodit*(1) to generate font description files from the URW foundry's AFM files; see section "Files" below.
- *Intermediate service* is available when *gropdf* can locate all 35 fonts of the PostScript Level 2 standard but not their corresponding font metrics (AFM) files. *groff*'s build process copies the font description files from those for the *grops*(1) driver, reusing them for *gropdf*; this reduces glyph coverage to 256 glyphs maximum from each face, and the "U" foundry is unavailable.
- *Basic service* results when *gropdf* cannot locate all 35 fonts of the PostScript Level 2 standard. Only the base 14 fonts of the PDF standard are available, and only in the sense that the formatter can use their metrics (copied from *grops* font descriptions as described above). Use of the **−e** option to embed fonts in the generated PDF results in an error.

Device extension commands

gropdf supports many device extensions, accessed with the *groff* request **device** or *roff* **\X** escape sequence. First, it understands many of the device extensions supported by *grops*(1).

\X'ps: invis'
Suppress output.

\X'ps: endinvis'
Stop suppressing output.

\X'ps: exec gsave currentpoint 2 copy translate *n* **rotate neg exch neg exch translate'**
where *n* is the angle of rotation. This is to support the **align** command in *pic*(1).

\X'ps: exec grestore'
Used by *pic*(1) to restore state after rotation.

\X'ps: exec *n* **setlinejoin'**
where *n* can be one of the following values.

0 = Miter join
1 = Round join
2 = Bevel join

\X'ps: exec *n* **setlinecap'**
where *n* can be one of the following values.

0 = Butt cap
1 = Round cap, and
2 = Projecting square cap

gropdf also supports a subset of the commands introduced in *gpresent*'s *present.tmac*.

PAUSE
BLOCKS
BLOCKE

These allow you to create presentation PDFs. Many of the other commands are already available in other macro packages.

These commands are implemented with *groff* X commands:-

\X'ps: exec %%%%PAUSE'
: The section before this is treated as a block and is introduced using the current **BLOCK** transition setting (see "**\X'pdf: transition'**" below). Equivalently, **.pdfpause** is available as a macro.

\X'ps: exec %%%%BEGINONCE'
: Any text following this command (up to %%%%ENDONCE) is shown only once, the next %%%%PAUSE will remove it. If producing a non-presentation PDF, i.e. ignoring the pauses, see *GROPDF_NOSLIDE* below, this text is ignored.

\X'ps: exec %%%%ENDONCE'
: This terminates the block defined by %%%%BEGINONCE. This pair of commands is what implements the .BLOCKS Once/.BLOCKE commands in *present.tmac*.

The *mom* macro package already integrates these extensions, so you can build slides with *mom*.

If you use *present.tmac* with *gropdf* there is no need to run the program *presentps*(1) since the output will already be a presentation PDF.

All other **ps:** tags are silently ignored.

gropdf also recognizes a device extension used by the DVI driver.

\X'papersize=*width***,***length***'**
: Set the page dimensions in centimeters to *width* by *length.* If the **–l** option was specified, these dimensions are swapped. Changes to the paper dimensions should occur prior to the first page, or during page ejection before starting a subsequent one.

  Caution: the ordering of dimensions differs from that used by *papersize.tmac* and *troff*(1)'s "**–d paper**" option.

\X'pdf: markstart */ANN-definition*'
\X'pdf: markend'
: Macros that support PDF features use these extension commands internally to bracket *hotspot* text (a hyperlink). User documents should call the **.pdfhref** macro instead. Their application is found in other macro packages (like *groff_man*(7) or *groff_mdoc*(7)) that call **.pdfhref** with a **–S** argument, then indicate the end of hotspot text with **\X'pdf: markend'\m[*[pdf:curcol]]**.

\X'pdf: xrev'
: Toggle the reversal of glyph direction. This feature works by reversing all following text. Each separate letter is also mirrored. One application is the reversal of glyphs in the Zapf Dingbats font. To restore the normal glyph orientation, repeat the command.

gropdf supports several more device extensions using the **pdf:** tag. The following have counterpart *convenience macros* that take the same arguments and behave equivalently.

.pdfbackground *cmd left top right bottom weight*
.pdfbackground off
.pdfbackground footnote *bottom*
\X'pdf: background *cmd left top right bottom weight*'
\X'pdf: background off'
\X'pdf: background footnote *bottom*'
: Produce a background rectangle on the page.

  cmd is the command, which can be any of "**page**|**fill**|**box**" in combination. Thus, "**pagefill**" would draw a rectangle which covers the whole current page size (in which case the rest of the parameters can be omitted because the box dimensions are taken from the current media size). "**boxfill**", on the other hand, requires the given dimensions to place the box. Including "**fill**" in the command paints the rectangle with the current fill colour (as with **\M[]**) and including "**box**" gives the rectangle a border in the current stroke colour (as with **\m[]**).

cmd may also be "**off**" on its own, which terminates drawing the current box. If you have specified a page colour with "**pagefill**", it is always the first box in the stack, and if you specify it again, it replaces the first entry. Be aware that the "**pagefill**" box renders the page opaque, so tools that "watermark" PDF pages are unlikely to be successful. To return the background to transparent, issue an "**off**" command with no other boxes open.

Finally, *cmd* may be "**footnote**" followed by a new value for *bottom*, which is used for all open boxes on the current page. This is to allow room for footnote areas that grow while a page is processed (to accommodate multiple footnotes, for instance). (If the value is negative, it is used as an offset from the bottom of the page.)

left
top
right
bottom are the coordinates of the box. The *top* and *bottom* coordinates are the minimum and maximum for the box, since the actual start of the box is *groff*'s drawing position when you issue the command, and the bottom of the box is the point where you turn the box "**off**". The top and bottom coordinates are used only if the box drawing extends onto the next page; ordinarily, they would be set to the header and footer margins.

weight provides the line width for the border if "**box**" is included in the command.

An *sboxes* macro file is also available; see *groff_tmac*(5).

.pdfmarksuspend
.pdfmarkrestart
\X'pdf: marksuspend'
\X'pdf: markrestart'

If you use a page location trap to produce a header or footer, or otherwise interrupt a document's text, you need to use these commands if a PDF *hotspot* crosses a trap boundary; otherwise any text output by the trap will be marked as part of the hotspot. To prevent this error, place these device extension escape sequences or their corresponding convenience macros **.pdfmarksuspend** and **.pdfmarkrestart** at the start and end of the trap macro, respectively.

.pdfpagename *name*
\X'pdf: pagename *name*'

Assign the current page a *name*. All documents bear two default names, ' **top**' and '**bottom**'.

.pdfpagenumbering *type prefix start*
\X'pdf: pagenumbering *type prefix start*'

Control the page numbering shown in a PDF reader's outline (which also contains bookmarks). Normally, the page number associated with each bookmark is its sequence number in the file, but this might not match the desired numbering scheme. A document may bear a cover sheet (which has no page number); front matter (possibly including a table of contents) that uses lowercase roman numerals; the main matter, which uses arabic numerals; and back matter, which may include appendices that are each prefixed with a letter and independently numbered. Place this command prior to breaking the page to which the new numbering scheme is to apply. It then persists until changed again.

type specifies the numbering system to use. It should be one of "**Decimal**", "**Roman**", "**roman**", "**Alpha**", or "**alpha**". This parameter may be abbreviated to the first letter, whose lettercase determines that used for the numbers where applicable. The ordering used by the alphabetic numbering systems is A-Z ... AA-AZ ... ZA-ZZ. *type* can also be "**.**", which selects no numbering system; you may still provide a *prefix*.

prefix specifies text to precede the page number. For example, to number the pages of an appendix "A-1", "A-2", and so forth, use a *prefix* of "**A-**" and a *type* of "**Decimal**".

start determines the page number. It defaults to 1.

.pdfpic *file alignment width height line-length*
\X'pdf: pdfpic *file alignment width height line-length*'
Place an image from file *file* of desired *width* and *height* (if *height* is missing or zero then it is scaled proportionally). If *alignment* is **−L** the drawing is left-aligned. If it is **−C** or **−R** a *line-length* greater than the width of the drawing is required as well. If *width* is specified as zero then the width is scaled in proportion to the height. If both *width* and *height* are non-zero the image is scaled to 'best fit'.

The availability of other software on the system, such as *PerlMagick*, influences the types of image files *gropdf* can embed in its output.

	none	*file*(1)	*identify*(1)	*Image::Magick*(3pm)
.pdf	✓	✓	✓	✓
.jpg	✗	✓	✓	✓
.jp2	✗	✗	✓	✓
other	✗	✗	✗	✓

See *groff_tmac*(5) for a description of the **PDFPIC** macro, which provides a convenient high-level interface for inclusion of various graphic file formats.

.pdfswitchtopage *when name*
\X'pdf: switchtopage *when name*'
Normally each new page is appended to the end of the document, this command allows following pages to be inserted at a *'named'* position within the document (see pagename command above). *'when'* can be either '*after*' or ' *before*'. If it is omitted it defaults to ' *before*'. It should be used at the end of the page before you want the switch to happen. This allows pages such as a TOC to be moved to elsewhere in the document, but more esoteric uses are possible.

.pdftransition *scope mode duration dimension motion direction scale bool*
\X'pdf: transition *scope mode duration dimension motion direction scale bool*'
Configure the style of page transitions, as used in "slides" (or "foils"). *scope* can be either **SLIDE** or **BLOCK**. **SLIDE** applies the transition when a new slide is introduced to the screen; **BLOCK** applies it to the individual blocks making up the slide.

mode is the transition type between slides:-

Split - Two lines sweep across the screen, revealing the new page. The lines may be either horizontal or vertical and may move inward from the edges of the page or outward from the center, as specified by the *dimension* and *motion* entries, respectively.
Blinds - Multiple lines, evenly spaced across the screen, synchronously sweep in the same direction to reveal the new page. The lines may be either horizontal or vertical, as specified by the *dimension* entry. Horizontal lines move downward; vertical lines move to the right.
Box - A rectangular box sweeps inward from the edges of the page or outward from the center, as specified by the *motion* entry, revealing the new page.
Wipe - A single line sweeps across the screen from one edge to the other in the direction specified by the *direction* entry, revealing the new page.
Dissolve - The old page dissolves gradually to reveal the new one.
Glitter - As Dissolve, except that the effect sweeps across the page in a wide band moving from one side of the screen to the other in the direction specified by the *direction* entry.
R - The new page simply replaces the old one with no special transition effect; the *direction* entry shall be ignored.
Fly - (PDF 1.5) Changes are flown out or in (as specified by *motion*), in the direction specified by *direction*, to or from a location that is offscreen except when *direction* is **None**.
Push - (PDF 1.5) The old page slides off the screen while the new page slides in, pushing the old page out in the direction specified by *direction*.

Cover - (PDF 1.5) The new page slides on to the screen in the direction specified by *direction*, covering the old page.
Uncover - (PDF 1.5) The old page slides off the screen in the direction specified by *direction*, uncovering the new page in the direction specified by *direction*.
Fade - (PDF 1.5) The new page gradually becomes visible through the old one.

duration is the length of the transition in seconds (default 1).

dimension (Optional; **Split** and **Blinds** transition styles only) The dimension in which the specified transition effect shall occur: **H** Horizontal, or **V** Vertical.

motion (Optional; **Split**, **Box** and **Fly** transition styles only) The direction of motion for the specified transition effect: **I** Inward from the edges of the page, or **O** Outward from the center of the page.

direction (Optional; **Wipe**, **Glitter**, **Fly**, **Cover**, **Uncover** and **Push** transition styles only) The direction in which the specified transition effect shall moves, expressed in degrees counterclockwise starting from a left-to-right direction. If the value is a number, it shall be one of: **0** = Left to right, **90** = Bottom to top (Wipe only), **180** = Right to left (Wipe only), **270** = Top to bottom, **315** = Top-left to bottom-right (Glitter only) The value can be **None**, which is relevant only for the **Fly** transition when the value of *scale* is not 1.0.

scale (Optional; PDF 1.5; **Fly** transition style only) The starting or ending scale at which the changes shall be drawn. If *motion* specifies an inward transition, the scale of the changes drawn shall progress from *scale* to 1.0 over the course of the transition. If *motion* specifies an outward transition, the scale of the changes drawn shall progress from 1.0 to *scale* over the course of the transition

bool (Optional; PDF 1.5; **Fly** transition style only) If **true**, the area that shall be flown in is rectangular and opaque.

Any of the parameters may be replaced with a "." which signifies the parameter retains its previous value, also any trailing missing parameters are ignored.

Note: not all PDF Readers support any or all these transitions.

Macros

gropdf's support macros in *pdf.tmac* define the convenience macros described above. Some features have no direct device extension escape sequence counterpart.

.pdfbookmark [**−T** *tag-name*] *level text*

Mark the nearest page location as a bookmark, and optionally a named destination as well. *Bookmarks* populate the outline pane of the reader. They are organized into a hierarchical tree; each level of the tree is numbered, starting at 1, and named as *text* in the outline. *Named destinations* permit hyperlink-style navigation within the document. Specifying **−T** followed by *tag-name* creates a named destination making the page location eligible as a target named by "**.pdfhref L** . . .".

.pdfhref L −D *dest* [**−S**] [**−P** *prefix-text*] [**−A** *suffix-text*] [*link-text*]

Create a hotspot link to *dest,* (the *tag-name*) which a " **.pdfbookmark** . . ." or "**.pdfhref M** . . ." call elsewhere in the document should define. (If the document employs forward references, it must be processed twice; see *pdfmom*(1).) If *link-text* is omitted the text associated with *dest*, when it was created, is formatted as the link text. The **−P** and **−A** arguments format their successors as text before and after the link text, respectively, without intervening space. Specifying **−S** prevents **pdfhref** from "closing" the hotspot, requiring the document (or macro package wrapping **pdfhref**) to do so itself with "**\X'pdf: markend'\m[*[pdf:curcol]]**".

.pdfhref M [**−E**] [**−N** *tag-name*] *dest*

Mark the nearest page location as a destination named (the first word of) *dest,* which should be unique within a document. Specifying **−T** followed by *tag-name* overrides this default. Specifying **−E** formats *dest* as text in the document as well.

.pdfhref W −D *uri* [**−S**] [**−P** *prefix-text*] [**−A** *suffix-text*] *link-text*

Create a hotspot link to *uri,* a World Wide Web Universal Resource Identifier (URI). The **−P** and **−A** arguments format their successors as text before and after the link text, respectively, without intervening space. Specifying **−S** prevents **pdfhref** from "closing" the hotspot, requiring the document (or macro package wrapping **pdfhref**) to do so itself with "**\X'pdf: markend'\m[*[pdf:curcol]]**".

.pdfinfo / *field content* . . .

Define PDF metadata. *field* may be one of **Title**, **Author**, **Subject**, **Keywords**, or another datum supported by the PDF standard or your reader. *field* must be prefixed with a slash.

.pdfnote [**−T** *title*] *text*

Create an annotation in the document. Reader support for this feature varies. Some place an icon at the current position on the page; hovering over the icon reveals any *title*, while clicking on the icon pops up a window containing *text*.

Parameters

The following parameters, shown as *roff* control lines, affect the operation of *gropdf*.

Parameter	Purpose	Default
`.nr PDFNOTE.WIDTH`	Set width of annotation icon.	1c
`.nr PDFNOTE.HEIGHT`	Set height of annotation icon.	1c
`.ds PDFNOTE.COLOR`	Set RGB color of annotation icon (RGB)	1.00 1.00 0.00
`.ds PDFNOTE.OPACITY`	Set opacity of annotation icon (decimal value in [0, 1]).	0.6
`.nr PDFOUTLINE.FOLDLEVEL`	Set depth of visible bookmark hierarchy.	10000
`.nr PDFHREF.VIEW.LEADING`	Set position adjustment when clicking bookmark or internal hotspot.	5p
`.nr PDFHREF.LEADING`	Configure size of increased clickable area around a hotspot.	2.0p
`.ds PDFHREF.BORDER`	Configure the border width around a hotspot by specifying two zeroes followed by the desired width in points. Do not use a scaling unit.	0 0 0
`.ds PDFHREF.COLOR`	Set RGB color of link text.	0.00 0.35 0.60

In the foregoing, you can also spell "COLOR" in string names as "COLOUR".

Importing PDF graphics

If you are importing an image as a PDF file, it must be a single page and the drawing must just fit inside the media size of the PDF file. In *inkscape*(1) or *gimp*(1), for example, make sure the canvas size just fits the image.

The PDF parser *gropdf* implements has not been rigorously tested with all applications that produce PDF. If you find a single-page PDF which fails to import properly, try processing it with the *pdftk*(1) program.

```
pdftk existing-file output new-file
```

You may find that *new-file* imports successfully.

TrueType and other font formats

gropdf does not yet support any font formats besides Adobe Type 1 (PFA or PFB).

Font installation

For your convenience, *groff* offers *install−font.bash*, a shell script that interactively assists the configuration of fonts for use with the GNU *troff* formatter and the *gropdf* output driver. See section "Files" below.

The following is a step-by-step font installation guide for *gropdf.*

- Convert your font to something *groff* understands. This is a PostScript Type 1 font in PFA or PFB format, together with an AFM file. A PFA file begins as follows.

  ```
  %!PS-AdobeFont-1.0:
  ```

  A PFB file contains this string as well, preceded by some non-printing bytes. In the following steps, we

will consider the use of CTAN's BrushScriptX-Italic font in PFA format.

- Convert the AFM file to a *groff* font description file with the *afmtodit*(1) program. For instance,

  ```
  $ afmtodit BrushScriptX-Italic.afm text.map BSI
  ```

  converts the Adobe Font Metric file *BrushScriptX-Italic.afm* to the *groff* font description file *BSI*.

  If you have a font family which provides regular upright (roman), bold, italic, and bold-italic styles, (where "italic" may be "oblique" or "slanted"), we recommend using **R**, **B**, **I**, and **BI**, respectively, as suffixes to the *groff* font family name to enable *groff*'s font family and style selection features. An example is *groff*'s built-in support for Times: the font family name is abbreviated as **T**, and the *groff* font names are therefore **TR**, **TB**, **TI**, and **TBI**. In our example, however, the BrushScriptX font is available in a single style only, italic.

- Install the *groff* font description file(s) in a *devpdf* subdirectory in the search path that *groff* uses for device and font file descriptions. See the *GROFF_FONT_PATH* entry in section "Environment" of *troff*(1) for the current value of the font search path. While *groff* doesn't directly use AFM files, it is a good idea to store them alongside its font description files.

- Register fonts in the *devpdf/download* file so they can be located for embedding in PDF files *gropdf* generates. Only the first *download* file encountered in the font search path is read. If in doubt, copy the default *download* file (see section "Files" below) to the first directory in the font search path and add your fonts there. The PostScript font name used by *gropdf* is stored in the **internalname** field in the *groff* font description file. (This name does not necessarily resemble the font's file name.) If the font in our example had originated from a foundry named **Z**, we would add the following line to *download*.

  ```
  Z→BrushScriptX-Italic→BrushScriptX-Italic.pfa
  ```

  A tab character, depicted as →, separates the fields. The default foundry has no name: its field is empty and entries corresponding to it start with a tab character, as will the one in our example.

- Test the selection and embedding of the new font.

  ```
  printf "\\f[BSI]Hello, world!\n" | groff -T pdf -P -e >hello.pdf
  see hello.pdf
  ```

Exit status

0 *gropdf* successfully produced a PDF document.

1 *gropdf* experienced a critical error, or warnings were emitted and the **-W** option was specified.

2 *gropdf* could not interpret its command-line arguments.

Environment

GROFF_FONT_PATH
: A list of directories in which to seek the selected output device's directory of device and font description files. If, in the *download* file, the font file has been specified with a full path, no directories are searched. See *troff*(1) and *groff_font*(5).

GROPDF_NOSLIDE
: If set and evaluates to a true value (to Perl), *gropdf* ignores commands specific to presentation PDFs, producing a normal PDF instead.

GROPDF_OPTIONS
: *gropdf* interprets the contents of this environment variable as a space-separated list of command-line options. Explicit command-line options override any settings from this environment variable.

SOURCE_DATE_EPOCH
: A timestamp (expressed as seconds since the Unix epoch) to use as the output creation timestamp in place of the current time. The time is converted to human-readable form using Perl's *gmtime()* function and recorded in a PDF comment.

TZ The time zone to use when converting the current time to human-readable form; see *tzset*(3). If *SOURCE_DATE_EPOCH* is used, it is always converted to human-readable form using UTC.

Files

/usr/share/groff/1.24.2/font/devpdf/DESC
: describes the **pdf** output device.

*/usr/share/groff/1.24.2/font/devpdf/*F
: describes the font known as *F* on device **pdf**.

*/usr/share/groff/1.24.2/font/devpdf/U−*F
: describes the font from the URW foundry (versus the Adobe default) known as *F* on device **pdf**.

/usr/share/groff/1.24.2/font/devpdf/download
: lists fonts available for embedding within the PDF document (by analogy to the **ps** device's downloadable font support).

/usr/share/groff/1.24.2/font/devpdf/Foundry
: is a data file used by the *groff* build system to locate PostScript Type 1 fonts.

/usr/share/groff/1.24.2/font/devpdf/symbolsl.afm
: provides metrics for the slanted symbol font known to *groff* as **SS**. These data facilitate use of the font with non-*groff* software.

/usr/share/groff/1.24.2/font/devpdf/symbolsl.pfb
: supplies the slanted symbol font known to *groff* as **SS**.

/usr/share/groff/1.24.2/font/devpdf/enc/text.enc
: describes the encoding scheme used by most PostScript Type 1 fonts; the **encoding** directive of font description files for the **pdf** device refers to it.

/usr/share/groff/1.24.2/font/devpdf/generate/symbolsl.sfd
: is the source form of the *symbolsl.pfb* font, in spline font database (SFD) format.

/usr/share/groff/1.24.2/tmac/pdf.tmac
: defines macros for use with the **pdf** output device. It is automatically loaded by *troffrc* when the **pdf** output device is selected.

/usr/share/groff/1.24.2/tmac/pdfpic.tmac
: defines the **PDFPIC** macro for embedding images in a document; see *groff_tmac*(5). It is automatically loaded by *troffrc*.

/usr/share/doc/groff/examples/install−font.bash
: This script, contributed by *mom* macro package author Peter Schaffter and long available at his web site, assists with making TrueType (*.ttf*), OpenType (*.otf*), and PostScript Type 1 (*.pfa*, *.pfb*) fonts available to *groff*.

  Change to its directory and run "**bash install−font.bash −H**" for a man page-like description of its features and operation.

Authors

gropdf was written and is maintained by Deri James.

See also

/usr/share/doc/groff/sboxes/msboxes.ms
/usr/share/doc/groff/sboxes/msboxes.pdf
: "Using PDF boxes with *groff* and the *ms* macros", by Deri James.

present.tmac
: is part of *gpresent*, a software package by Bob Diertens that works with *groff* to produce presentations ("foils", or "slide decks").

afmtodit(1), *groff*(1), *troff*(1), *groff_font*(5), *groff_out*(5)

Name

grops – *groff* output driver for PostScript

Synopsis

grops [**–glm**] [**–b** *brokenness-flags*] [**–c** *num-copies*] [**–F** *font-directory*] [**–I** *inclusion-directory*] [**–p** *paper-format*] [**–P** *prologue-file*] [**–w** *rule-thickness*] [*file* . . .]

grops ––help

grops –v
grops ––version

Description

The GNU *roff* PostScript output driver translates the output of *troff* (1) into PostScript. Normally, *grops* is invoked by *groff* (1) when the latter is given the "**–T ps**" option. (In this installation, **ps** is the default output device.) Use *groff*'s **–P** option to pass any options shown above to *grops*. If no *file* arguments are given, or if *file* is "–", *grotty* reads the standard input stream. It writes to the standard output stream.

When called with multiple *file* arguments, *grops* doesn't produce a valid document structure (one conforming to the Document Structuring Conventions). To print such catenated output, it is necessary to deactivate DSC handling in the printing program or previewer.

See section "Font installation" below for a guide to installing fonts for *grops*.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–b *n* Work around problems with spoolers, previewers, and older printers. Normally, *grops* produces output at PostScript LanguageLevel 2 that conforms to version 3.0 of the Document Structuring Conventions. Some software and devices can't handle such a data stream. The value of *n* determines what *grops* does to make its output acceptable to such consumers. If *n* is **0**, *grops* employs no workarounds, which is the default; it can be changed by modifying the **broken** directive in *grops*'s *DESC* file.

Add 1 to suppress generation of **%%BeginDocumentSetup** and **%%EndDocumentSetup** comments; this is needed for early versions of TranScript that get confused by anything between the **%%EndProlog** comment and the first **%%Page** comment.

Add 2 to omit lines in included files beginning with **%!**, which confuse Sun's *pageview* previewer.

Add 4 to omit lines in included files beginning with **%%Page**, **%%Trailer** and **%%EndProlog**; this is needed for spoolers that don't understand **%%BeginDocument** and **%%EndDocument** comments.

Add 8 to write **%!PS–Adobe–2.0** rather than **%!PS–Adobe–3.0** as the first line of the PostScript output; this is needed when using Sun's Newsprint with a printer that requires page reversal.

Add 16 to omit media size information (that is, output neither a **%%DocumentMedia** comment nor the **setpagedevice** PostScript command). This was the behavior of *groff* 1.18.1 and earlier; it is needed for older printers that don't understand PostScript LanguageLevel 2, and is also necessary if the output is further processed to produce an EPS file; see subsection "Escapsulated PostScript" below.

–c *n* Output *n* copies of each page.

–F *dir* Prepend directory dir*/devname* to the search path for font and device description and PostScript prologue files; *name* is the name of the device, usually **ps**.

–g Generate PostScript code to guess the page length. The guess is correct only if the imageable area is vertically centered on the page. This option allows you to generate documents that can be printed on both U.S. letter and A4 paper formats without change.

–I *dir* Search the directory *dir* for files named in **\X'ps: file'** and **\X'ps: import'** escape sequences. **–I** may be specified more than once; each *dir* is searched in the given order. To search the current working directory before others, add "**–I .**" at the desired place; it is otherwise searched last.

–l Use landscape orientation rather than portrait.

–m Turn on manual feed for the document.

–p *fmt* Set physical dimensions of output medium, overriding the **papersize**, **paperlength**, and **paperwidth** directives in the *DESC* file. *fmt* can be any argument accepted by the **papersize** directive; see *groff_font*(5).

–P *prologue*
Use the file *prologue*, sought in the *groff* font search path, as the PostScript prologue, overriding the default (see section "Files" below) and the environment variable *GROPS_PROLOGUE*.

–w *n* Draw rules (lines) with a thickness of *n* thousandths of an em. The default thickness is **40** (0.04 em).

Usage

The input to *grops* must be in the format output by *troff*(1), described in *groff_out*(5). In addition, the device and font description files for the device used must meet certain requirements. The device resolution must be an integer multiple of 72 times the **sizescale**. The device description file must contain a valid paper format; see *groff_font*(5). Each font description file must contain a directive

```
internalname psname
```

which says that the PostScript name of the font is *psname*.

A font description file may also contain a directive

```
encoding enc-file
```

which says that the PostScript font should be reencoded using the encoding described in *enc-file*; this file should consist of a sequence of lines of the form

```
pschar code
```

where *pschar* is the PostScript name of the character, and *code* is its position in the encoding expressed as a decimal integer; valid values are in the range 0 to 255. Lines starting with **#** and blank lines are ignored. The code for each character given in the font description file must correspond to the code for the character in encoding file, or to the code in the default encoding for the font if the PostScript font is not to be reencoded. This code can be used with the **\N** escape sequence in *troff* to select the character, even if it does not have a *groff* glyph name. Every character in the font description file must exist in the PostScript font, and the widths given in the font description file must match the widths used in the PostScript font. *grops* assumes that a character with a *groff* name of **space** is blank (makes no marks on the page); it can make use of such a character to generate more efficient and compact PostScript output.

grops is able to display all glyphs in a PostScript font; it is not limited to 256 of them. *enc-file* (or the default encoding if no encoding file is specified) just defines the order of glyphs for the first 256 characters; all other glyphs are accessed with additional encoding vectors which *grops* produces on the fly.

grops can embed fonts in a document that are necessary to render it; this is called "downloading". Such fonts must be in PFA format. Use *pfbtops*(1) to convert a Type 1 font in PFB format. Downloadable fonts must be listed a *download* file containing lines of the form

```
psname file
```

where *psname* is the PostScript name of the font, and *file* is the name of the file containing it. Blank lines and those beginning with **#** are ignored; fields are separated by tabs. *file* is sought using the same mechanism as for *groff* font description files. The *download* file itself is as well; currently, the first matching file found in the device and font description search path is used.

If the file containing a downloadable font or imported document conforms to the Adobe Document Structuring Conventions, then *grops* interprets any comments in the files sufficiently to ensure that its own output is conforming. It also supplies any needed font resources that are listed in the *download* file as well as any needed file resources. It is also able to handle inter-resource dependencies. For example, suppose that you have a

downloadable font called Garamond, and also a downloadable font called Garamond-Outline which depends on Garamond (typically it would be defined to copy Garamond's font dictionary, and change the PaintType), then it is necessary for Garamond to appear before Garamond-Outline in the PostScript document. *grops* handles this automatically provided that the downloadable font file for Garamond-Outline indicates its dependence on Garamond by means of the Document Structuring Conventions, for example by beginning with the following lines.

```
%!PS-Adobe-3.0 Resource-Font
%%DocumentNeededResources: font Garamond
%%EndComments
%%IncludeResource: font Garamond
```

In this case, both Garamond and Garamond-Outline would need to be listed in the *download* file. A downloadable font should not include its own name in a **%%DocumentSuppliedResources** comment.

grops does not interpret **%%DocumentFonts** comments. The **%%DocumentNeededResources**, **%%DocumentSuppliedResources**, **%%IncludeResource**, **%%BeginResource**, and **%%EndResource** comments (or possibly the old **%%DocumentNeededFonts**, **%%DocumentSuppliedFonts**, **%%IncludeFont**, **%%BeginFont**, and **%%EndFont** comments) should be used.

The default stroke and fill colors are black. For colors defined in the "rgb" color space, **setrgbcolor** is used; for "cmy" and "cmyk", **setcmykcolor**; and for "gray", **setgray**. **setcmykcolor** is a PostScript LanguageLevel 2 command and thus not available on some older printers.

Typefaces

Styles called **R**, **I**, **B**, and **BI** mounted at font positions 1 to 4. Text fonts are grouped into families **A**, **BM**, **C**, **H**, **HN**, **N**, **P**, and **T**, each having members in each of these styles.

AR	AvantGarde-Book
AI	*AvantGarde-BookOblique*
AB	**AvantGarde-Demi**
ABI	***AvantGarde-DemiOblique***
BMR	Bookman-Light
BMI	*Bookman-LightItalic*
BMB	**Bookman-Demi**
BMBI	***Bookman-DemiItalic***
CR	Courier
CI	*Courier-Oblique*
CB	**Courier-Bold**
CBI	***Courier-BoldOblique***
HR	Helvetica
HI	*Helvetica-Oblique*
HB	**Helvetica-Bold**
HBI	***Helvetica-BoldOblique***
HNR	Helvetica-Narrow
HNI	*Helvetica-Narrow-Oblique*
HNB	**Helvetica-Narrow-Bold**
HNBI	***Helvetica-Narrow-BoldOblique***
NR	NewCenturySchlbk-Roman
NI	*NewCenturySchlbk-Italic*
NB	**NewCenturySchlbk-Bold**
NBI	***NewCenturySchlbk-BoldItalic***
PR	Palatino-Roman
PI	*Palatino-Italic*
PB	**Palatino-Bold**

PBI ***Palatino-BoldItalic***
TR Times-Roman
TI *Times-Italic*
TB **Times-Bold**
TBI ***Times-BoldItalic***

Another text font is not a member of a family.

ZCMI *ZapfChancery-MediumItalic*

Special fonts include **S**, the PostScript Symbol font; **ZD**, Zapf Dingbats; **SS** (slanted symbol), which contains oblique forms of lowercase Greek letters derived from Symbol; **EURO**, which offers a Euro glyph for use with old devices lacking it; and **ZDR**, a reversed version of Zapf Dingbats (with symbols flipped about the vertical axis). Most glyphs in these fonts are unnamed and must be accessed using **\N**. The last three are not standard PostScript fonts, but supplied by *groff* and therefore included in the default *download* file.

grops furthermore supports a naming scheme for East Asian typefaces shared with *grohtml*(1), *gropdf*(1), and *grotty*(1).

CSH Simplified Chinese, Hei style
CSS Simplified Chinese, Song style
CTH Traditional Chinese, Hei style
CTS Traditional Chinese, Song style
JPG Japanese, Gothic style
JPM Japanese, Mincho style
KOG Korean, Gothic style
KOM Korean, Mincho style

Device extension commands

grops recognizes device extension commands produced by the *groff* request **device** or *roff* **\X** escape sequence, but interprets only those that begin with a "**ps:**" tag.

\X'ps: exec *code*'

Execute the arbitrary PostScript commands *code*. The PostScript *currentpoint* is set to the *groff* drawing position when the **\X** escape sequence is interpreted before executing *code*. The origin is at the top left corner of the page; *x* coordinates increase to the right, and *y* coordinates down the page. A procedure **u** is defined that converts *groff* basic units to the coordinate system in effect (provided the user doesn't change the scale). For example,

```
.nr x 1i
\X'ps: exec \nx u 0 rlineto stroke'
```

draws a horizontal line one inch long. *code* may make changes to the graphics state, but any changes persist only to the end of the page. A dictionary containing the definitions specified by the **def** and **mdef** commands is on top of the dictionary stack. If your code adds definitions to this dictionary, you should allocate space for them using "**\X'ps: mdef** *n*'". Any definitions persist only until the end of the page. If you use the **\Y** escape sequence with an argument that names a macro, *code* can extend over multiple lines. For example,

```
.nr x 1i
.de y
ps: exec
\nx u 0 rlineto
stroke
..
\Yy
```

is another way to draw a horizontal line one inch long. The single backslash before "**nx**"—the only reason to use a register while defining the macro "**y**"—is to convert a user-specified dimension "**1i**" to *groff* basic units which are in turn converted to PostScript units with the **u** procedure.

grops wraps user-specified PostScript code into a dictionary, nothing more. In particular, it doesn't start and end the inserted code with **save** and **restore**, respectively. This must be supplied by the user, if necessary.

\X'ps: file *name*'
This is the same as the **exec** command except that the PostScript code is read from file *name*.

\X'ps: def *code*'
Place a PostScript definition contained in *code* in the prologue. There should be at most one definition per **\X** command. Long definitions can be split over several **\X** commands; all the *code* arguments are simply joined together separated by newlines. The definitions are placed in a dictionary which is automatically pushed on the dictionary stack when an **exec** command is executed. If you use the **\Y** escape sequence with an argument that names a macro, *code* can extend over multiple lines.

\X'ps: mdef *n code*'
Like **def**, except that *code* may contain up to *n* definitions. *grops* needs to know how many definitions *code* contains so that it can create an appropriately sized PostScript dictionary to contain them.

\X'ps: import *file llx lly urx ury width* [*height*]'
Import a PostScript graphic from *file*. The arguments *llx*, *lly*, *urx*, and *ury* give the bounding box of the graphic in the default PostScript coordinate system. They should all be integers: *llx* and *lly* are the *x* and *y* coordinates of the lower left corner of the graphic; *urx* and *ury* are the *x* and *y* coordinates of the upper right corner of the graphic; *width* and *height* are integers that give the desired width and height in *groff* basic units of the graphic.

The graphic is scaled so that it has this width and height and translated so that the lower left corner of the graphic is located at the position associated with **\X** command. If the height argument is omitted it is scaled uniformly in the *x* and *y* axes so that it has the specified width.

The contents of the **\X** command are not interpreted by *troff*, so vertical space for the graphic is not automatically added, and the *width* and *height* arguments are not allowed to have attached scaling indicators.

If the PostScript file complies with the Adobe Document Structuring Conventions and contains a **%%BoundingBox** comment, then the bounding box can be automatically extracted from within *groff* input by using the **psbb** request.

See *groff_tmac*(5) for a description of the **PSPIC** macro which provides a convenient high-level interface for inclusion of PostScript graphics.

\X'ps: invis'
\X'ps: endinvis'
No output is generated for text and drawing commands that are bracketed with these **\X** commands. These commands are intended for use when output from *troff* is previewed before being processed with *grops*; if the previewer is unable to display certain characters or other constructs, then other substitute characters or constructs can be used for previewing by bracketing them with these **\X** commands.

For example, *gxditview* is not able to display a proper **\[em]** character because the standard X11 fonts do not provide it; this problem can be overcome with the following request.

```
.char \[em] \X'ps: invis'\
\Z'\v'-.25m'\h'.05m'\D'l .9m 0'\h'.05m''\
\X'ps: endinvis'\[em]
```

In this case, *gxditview* is unable to display the **\[em]** character and draws the line, whereas *grops* prints the **\[em]** character and ignores the line (this code is already in file *Xps.tmac*, which is loaded if a document intended for *grops* is previewed with *gxditview*).

If a PostScript procedure **BPhook** has been defined via a "**ps: def**" or "**ps: mdef**" device extension command, it is executed at the beginning of every page (before anything is drawn or written by *groff*). For example, to underlay the page contents with the word "DRAFT" in light gray, you might use the following.

```
.de XX
ps: def
/BPhook
{ gsave .9 setgray clippath pathbbox exch 2 copy
  .5 mul exch .5 mul translate atan rotate pop pop
  /NewCenturySchlbk-Roman findfont 200 scalefont setfont
  (DRAFT) dup stringwidth pop -.5 mul -70 moveto show
  grestore }
def
..
.devicem XX
```

Or, to cause lines and polygons to be drawn with square linecaps and mitered linejoins instead of the round linecaps and linejoins normally used by *grops*, use the following.

```
.de XX
ps: def
/BPhook { 2 setlinecap 0 setlinejoin } def
..
.devicem XX
```

(Square linecaps, as opposed to butt linecaps ("**0 setlinecap**"), give true corners in boxed tables even though the lines are drawn unconnected.)

Encapsulated PostScript

grops itself doesn't emit bounding box information. The following script, *groff2eps*, produces an EPS file.

```
#!/bin/sh
groff -P-b16 "$1" > "$1".ps
gs -dNOPAUSE -sDEVICE=bbox -- "$1".ps 2> "$1".bbox
sed -e "/^%%Orientation/r $1.bbox" \
    -e "/^%!PS-Adobe-3.0/s/$/ EPSF-3.0/" "$1".ps > "$1".eps
rm "$1".ps "$1".bbox
```

You can then use "**groff2eps foo**" to convert file *foo* to *foo.eps*.

TrueType and other font formats

To use TrueType fonts with *grops*, convert them first to Type 42 format, a PostScript wrapper equivalent to the PFA format described in *pfbtops*(1). Several methods exist to generate a Type 42 wrapper; some of them involve the use of a PostScript interpreter such as Ghostscript—see *gs*(1).

FontForge converts most outline font formats. Here's how we'd set up Roboto Slab Serif for use with *groff*.

```
MAP=/usr/share/groff/1.24.2/font/devps/generate/text.map
TTF=/usr/share/fonts/truetype/roboto/slab/RobotoSlab-Regular.ttf
BASE=$(basename "$TTF")
INT=${BASE%.ttf}
PFA=$INT.pfa
AFM=$INT.afm
GFN=RSR
DIR=$HOME/.local/groff/font
mkdir -p "$DIR"/devps
fontforge -lang=ff -c "Open(\"$TTF\");\
Generate(\"$DIR/devps/$PFA\");"
afmtodit "$DIR/devps/$AFM" "$MAP" "$DIR/devps/$GFN"
printf "$BASE\t$PFA\n" >> "$DIR/devps/download"
```

fontforge and *afmtodit* may produce warnings depending on font file attributes. Test the font as follows.

```
printf ".ft RSR\nHello, world!\n" | groff -F "$DIR" > hello.ps
```

Once you're satisfied that the font works, you may want to generate any available related styles (for instance, Roboto Slab also has "Bold", "Light", and "Thin" styles) and set up *GROFF_FONT_PATH* in your environment to include the directory you keep the generated fonts in so that you don't have to use the **-F** option.

Font installation

For your convenience, *groff* offers *install-font.bash*, a shell script that interactively assists the configuration of fonts for use with the GNU *troff* formatter and the *grops* output driver. See section "Files" below.

The following is a step-by-step font installation guide for *grops*.

- Convert your font to something *groff* understands. This is a PostScript Type 1 font in PFA format or a PostScript Type 42 font, together with an AFM file. A PFA file begins as follows.

  ```
  %!PS-AdobeFont-1.0:
  ```

  A PFB file contains this string as well, preceded by some non-printing bytes. If your font is in PFB format, use *groff*'s *pfbtops*(1) program to convert it to PFA. For TrueType and other font formats, we recommend *fontforge*, which can convert most outline font formats. A Type 42 font file begins as follows.

  ```
  %!PS-TrueTypeFont
  ```

  This is a wrapper format for TrueType fonts. Old PostScript printers might not support them (that is, they might not have a built-in TrueType font interpreter). In the following steps, we will consider the use of CTAN's BrushScriptX-Italic font in PFA format.

- Convert the AFM file to a *groff* font description file with the *afmtodit*(1) program. For instance,

  ```
  $ afmtodit BrushScriptX-Italic.afm text.map BSI
  ```

  converts the Adobe Font Metric file *BrushScriptX-Italic.afm* to the *groff* font description file *BSI*.

  If you have a font family which provides regular upright (roman), bold, italic, and bold-italic styles (where "italic" may be "oblique" or "slanted"), we recommend using the letters **R**, **B**, **I**, and **BI**, respectively, as suffixes to the *groff* font family name to enable *groff*'s font family and style selection features. An example is *groff*'s built-in support for Times: the font family name is abbreviated as **T**, and the *groff* font names are therefore **TR**, **TB**, **TI**, and **TBI**. In our example, however, the BrushScriptX font is available in a single style only, italic.

- Install the *groff* font description file(s) in a *devps* subdirectory in the search path that *groff* uses for device and font file descriptions. See the *GROFF_FONT_PATH* entry in section "Environment" of *troff*(1) for the current value of the font search path. While *groff* doesn't directly use AFM files, it is a good idea to store them alongside its font description files.

- Register fonts in the *devps/download* file so they can be located for embedding in PostScript files *grops* generates. Only the first *download* file encountered in the font search path is read. If in doubt, copy the default *download* file (see section "Files" below) to the first directory in the font search path and add your fonts there. The PostScript font name used by *grops* is stored in the **internalname** field in the *groff* font description file. (This name does not necessarily resemble the font's file name.) We add the following line to *download*.

  ```
  BrushScriptX-Italic→BrushScriptX-Italic.pfa
  ```

  A tab character, depicted as →, separates the fields.

- Test the selection and embedding of the new font.

  ```
  printf "\\f[BSI]Hello, world!\n" | groff -T ps >hello.ps
  see hello.pdf
  ```

Old fonts

groff versions 1.19.2 and earlier contained descriptions of a slightly different set of the base 35 PostScript level 2 fonts defined by Adobe. The older set has 314 glyphs and a larger set of kerning pairs; the newer one has only 229 glyphs, but includes the Euro sign. For backward compatibility, these old font descriptions are also installed in the */usr/share/groff/1.24.2/oldfont/devps* directory.

To use them, make sure that *grops* finds the fonts before the default system fonts (with the same names): either give *grops* the **−F** command-line option,

```
$ groff -Tps -P-F -P/usr/share/groff/1.24.2/oldfont ...
```

or add the directory to *groff*'s font and device description search path environment variable,

```
$ GROFF_FONT_PATH=/usr/share/groff/1.24.2/oldfont \
      groff -Tps ...
```

when the command runs.

Exit status

grops exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Environment

GROFF_FONT_PATH
: A list of directories in which to seek the selected output device's directory of device and font description files. See *troff*(1) and *groff_font*(5).

GROPS_PROLOGUE
: If this is set to *foo*, then *grops* uses the file *foo* (in the font path) instead of the default prologue file *prologue*. The option **−P** overrides this environment variable.

SOURCE_DATE_EPOCH
: A timestamp (expressed as seconds since the Unix epoch) to use as the output creation timestamp in place of the current time. The time is converted to human-readable form using *gmtime*(3) and *asctime*(3), and recorded in a PostScript comment.

TZ
: The time zone to use when converting the current time to human-readable form; see *tzset*(3). If *SOURCE_DATE_EPOCH* is used, it is always converted to human-readable form using UTC.

Files

/usr/share/groff/1.24.2/font/devps/DESC
: describes the **ps** output device.

*/usr/share/groff/1.24.2/font/devps/*F
: describes the font known as *F* on device **ps**.

/usr/share/groff/1.24.2/font/devps/download
: lists fonts available for embedding within the PostScript document (or download to the device).

/usr/share/groff/1.24.2/font/devps/prologue
: is the default PostScript prologue prefixed to every output file.

/usr/share/groff/1.24.2/font/devps/text.enc
: describes the encoding scheme used by most PostScript Type 1 fonts; the **encoding** directive of font description files for the **ps** device refers to it.

/usr/share/groff/1.24.2/tmac/ps.tmac
: defines macros for use with the **ps** output device. It is automatically loaded by *troffrc* when the **ps** output device is selected.

/usr/share/groff/1.24.2/tmac/pspic.tmac
: defines the **PSPIC** macro for embedding images in a document; see *groff_tmac*(5). It is automatically loaded by *troffrc*.

/usr/share/groff/1.24.2/tmac/psold.tmac
: provides replacement glyphs for text fonts that lack complete coverage of the ISO Latin-1 character set; using it, *groff* can produce glyphs like eth (ð) and thorn (þ) that older PostScript printers do not natively support.

/usr/share/doc/groff/examples/install−font.bash

This script, contributed by *mom* macro package author Peter Schaffter and long available at his web site, assists with making TrueType (*.ttf*), OpenType (*.otf*), and PostScript Type 1 (*.pfa*, *.pfb*) fonts available to *groff*.

Change to its directory and run "**bash install−font.bash −H**" for a man page-like description of its features and operation.

grops creates temporary files using the template "*grops*XXXXXX"; see *groff*(1).

See also

PostScript Language Document Structuring Conventions Specification

afmtodit(1), *groff*(1), *troff*(1), *pfbtops*(1), *groff_char*(7), *groff_font*(5), *groff_out*(5), *groff_tmac*(5)

Name

grotty − *groff* output driver for typewriter-like (terminal) devices

Synopsis

grotty [**−dfhot**] [**−i**|**−r**] [**−F** *font-directory*] [*file* ...]

grotty −c [**−bBdfhouU**] [**−F** *font-directory*] [*file* ...]

grotty −−help

grotty −v
grotty −−version

Description

The GNU *roff* TTY ("Teletype") output driver translates the output of *troff*(1) into a form suitable for typewriter-like devices, including video terminal emulators. Normally, *grotty* is invoked by *groff*(1) when the latter is given one of the "**−T ascii**", "**−T latin1**", or "**−T utf8**" options. (In this installation, **ps** is the default output device.) Use *groff*'s **−P** option to pass any options shown above to *grotty*. If no *file* arguments are given, or if *file* is "−", *grotty* reads the standard input stream. It writes to the standard output stream.

By default, *grotty* emits SGR escape sequences (from ISO 6429, popularly called "ANSI escapes") to change text attributes (bold, italic, underline, reverse video ["negative image"] and colors). Devices supporting SGR 30–37 and 40–47 sequences can view *roff* documents using eight different background and foreground colors. *grotty*'s *tty.tmac* file defines the eight color names of ISO 6429: black, white, red, green, blue, yellow, magenta, and cyan. Unrecognized colors map to the default color, the value of which depends on the settings of the terminal. Also see the **−t** option below.

By default, *grotty* produces OSC 8 hyperlinks on devices employing SGR escape sequences.

In accord with long-standing practice and terminals (emulators) that lack support for slanted (oblique or italic) faces, *grotty* marks italicized character cells with underlines instead by default—but see the **−i** option below.

SGR and OSC support in pagers

When paging *grotty*'s output with *less*(1), the latter program must be instructed to pass SGR and OSC sequences through to the device; its **−R** option is one way to achieve this (*less* version 566 or later is required for OSC 8 support). Consequently, programs like *man*(1) that page *roff* documents with *less* must call it with an appropriate option.

Legacy output format

The **−c** option tells *grotty* to use an output format compatible with paper terminals, like the Teletype machines for which *roff* and *nroff* were first developed but which are no longer in wide use. SGR escape sequences are not emitted; bold, italic, and underlining character attributes are thus not manipulated. Instead, *grotty* overstrikes, representing a bold character *c* with the sequence "*c* BACKSPACE *c*", an italic character *c* with the sequence "_ BACKSPACE *c*", and bold italics with " _ BACKSPACE *c* BACKSPACE *c*". This rendering is inherently ambiguous when the character *c* is itself the underscore.

The legacy output format can be rendered on a video terminal (or emulator) by piping *grotty*'s output through *ul*(1), which may render bold italics as reverse video. Some implementations of *more*(1) also are able to display these sequences; you may wish to experiment with that command's **−b** option. *less* renders legacy bold and italics without requiring options. In contrast to the terminal output drivers of some other *roff* implementations, *grotty* never outputs reverse line feeds. You need not filter its output through *col*(1) to remove them.

Device extension commands

grotty recognizes a device extension command produced by the *groff* request **device** or *roff* **\X** escape sequence.

\X'tty: link [*uri* [*key*=*value*] ...]'

Embed a hyperlink using the OSC 8 terminal escape sequence. Specifying *uri* starts hyperlinked text, and omitting it ends the hyperlink. When *uri* is present, any number of additional key/value pairs can be specified; their interpretation is the responsibility of the pager or terminal. Spaces or tabs cannot appear literally in *uri*, *key*, or *value*; they must be represented in an alternate form.

Device description files

If the *DESC* file for the character encoding contains the "**unicode**" directive, *grotty* emits Unicode characters in UTF-8 encoding. Otherwise, it emits characters in a single-byte encoding depending on the data in the font description files. See *groff_font*(5).

A font description file may contain a directive "**internalname** *n*" where *n* is a decimal integer. If the 01 bit in *n* is set, *grotty* treats the font as slanted; if the 02 bit is set, *grotty* treats the font as bold.

Typefaces

grotty supports the standard four styles: **R** (roman), **I** (*italic*), **B** (**bold**), and **BI** (***bold-italic***). Because the output driver operates in *nroff* mode, attempts to set or change the font family or type size are ignored.

grotty shares a naming scheme for East Asian typefaces with *grohtml*(1), *gropdf*(1), and *grops*(1).

CSH	Simplified Chinese, Hei style
CSS	Simplified Chinese, Song style
CTH	Traditional Chinese, Hei style
CTS	Traditional Chinese, Song style
JPG	Japanese, Gothic style
JPM	Japanese, Mincho style
KOG	Korean, Gothic style
KOM	Korean, Mincho style

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−b Suppress the use of overstriking for bold characters in legacy output format.

−B Use only overstriking for bold-italic characters in legacy output format.

−c Use *grotty*'s legacy output format (see subsection "Legacy output format" above). SGR and OSC escape sequences are not emitted.

−d Ignore all drawing commands in the input. By default, *grotty* renders "**D l**" commands that have at least one zero argument (and so are either horizontal or vertical) using Unicode box drawing characters (for the **utf8** device) or the **−**, **|**, and **+** characters (for all other devices). *grotty* handles "**D p**" commands that consist entirely of horizontal and vertical lines similarly. See *groff_out*(5).

−f Emit a form feed at the end of each page having no output on its last line.

−F *dir* Prepend directory dir*/dev*name to the search path for font and device description files; *name* describes the output device's character encoding, one of **ascii**, **latin1**, or **utf8**.

−h Use literal horizontal tab characters in the output. Tabs are assumed to be set every 8 columns.

−i Render fonts marked as slanted with the SGR attribute for italic text rather than underlined text. Many terminals don't support this attribute; however, *xterm*(1), since patch #314 (2014-12-28), does. Ignored if **−c** is also specified.

−o Suppress overstriking (other than for bold and/or underlined characters when the legacy output format is in use; see options **−b** and **−u**).

−r Render fonts marked as slanted with the SGR attribute for reverse video text rather than underlined text. Ignored if **−c** or **−i** is also specified.

−t Assume that the output device supports SGR 38 and 48 escape sequences, which permit specification of character cell foreground and background colors in the RGB color space with 8 bits per channel.

−u Suppress the use of underlining for italic characters in legacy output format.

−U Use only underlining for bold-italic characters in legacy output format.

Exit status

grotty exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Environment

GROFF_FONT_PATH
: A list of directories in which to seek the selected output device's directory of device and font description files. See *troff*(1) and *groff_font*(5).

GROFF_NO_SGR
: If set, *grotty*'s legacy output format is used just as if the **−c** option were specified; see subsection "Legacy output format" above.

Files

/usr/share/groff/1.24.2/font/devascii/DESC
: describes the **ascii** output device.

*/usr/share/groff/1.24.2/font/devascii/*F
: describes the font known as *F* on device **ascii**.

/usr/share/groff/1.24.2/font/devlatin1/DESC
: describes the **latin1** output device.

*/usr/share/groff/1.24.2/font/devlatin1/*F
: describes the font known as *F* on device **latin1**.

/usr/share/groff/1.24.2/font/devutf8/DESC
: describes the **utf8** output device.

*/usr/share/groff/1.24.2/font/devutf8/*F
: describes the font known as *F* on device **utf8**.

/usr/share/groff/1.24.2/tmac/tty.tmac
: defines macros for use with the **ascii**, **latin1**, and **utf8** output devices. It is automatically loaded by *troffrc* when any of those output devices is selected.

/usr/share/groff/1.24.2/tmac/tty−char.tmac
: defines fallback characters for use with *grotty*. See *nroff*(1).

Limitations

grotty is intended only for simple documents.

- There is no support for horizontal or vertical motions smaller than a character cell.
- Drawing commands producing anything other than horizontal and vertical lines are not supported.
- Color handling differs from other output drivers. The GNU *troff* output commands produced by requests and escape sequences that set the stroke and fill colors instead set the foreground and background character cell colors, respectively.

The commands generated by the **\l** and **\L** escape sequences on one hand, and the **\D'l'** line-drawing escape sequence on the other, make different compromises due to the first two factors. Specifically, (1) **\l** draws horizontal lines with underscore characters; **\D'l'** uses ACS or Unicode line-drawing characters if possible, and hyphen-minus signs if not. (2) **\D'l'** draws vertical lines an extra character cell high, and horizontal lines an extra cell to the right. *grotty* does this to detect intersecting lines so that it can replace them with glyphs of appropriate appearance (like "+"). Observe the difference below.

The input

```
Hello,\L'1v'
world.\l'1n'
.sp 2v
Hello,\D'l 0 1v'
```

```
world.\D'l 1n 0'
.pl \n(nlu \" truncate page for convenience
```

rendered with "**nroff –T ascii**" produces the following output.

```
Hello,
      | world._

Hello,|
      |world.--
```

Examples

The following *groff* document exercises several features for which output device support varies: (1) bold style; (2) italic (underline) style; (3) bold-italic style; (4) character composition by overstriking ("coöperate"); (5) foreground color; (6) background color; and (7) horizontal and vertical line drawing.

```
You might see \f[B]bold\f[] and \f[I]italic\f[].
Some people see \f[BI]both at once\f[].
If the output device does (not) co\z\[ad]operate,
you might see \m[red]red\m[].
Black on cyan can have a \M[cyan]\m[black]prominent\m[]\M[]
\D'l 1i 0'\D'l 0 2i'\D'l 1i 0' look.
.\" If in nroff mode, end page now.
.if n .pl \n[nl]u
```

Given the foregoing input, compare and contrast the output of the following.

```
$ groff -T ascii file
$ groff -T utf8 -P -i file
$ groff -T utf8 -P -c file | ul
```

See also

"Control Functions for Coded Character Sets" (ECMA-48) 5th edition, Ecma International, June 1991. A gratis version of ISO 6429, this document includes a normative description of SGR escape sequences.

"Hyperlinks in Terminal Emulators", Egmont Koblinger.

groff(1), *troff*(1), *groff_out*(5), *groff_font*(5), *groff_char*(7), *ul*(1), *more*(1), *less*(1), *man*(1)

Name

gxditview – display *troff* output in X11

Synopsis

gxditview [*X-toolkit-option* ...] [**–backingStore** *backing-store-type*] [**–filename** *file*] [**–page** *page-number*] [**–printCommand** *command*] [**–resolution** *resolution*] *file*

gxditview –help
gxditview ––help

gxditview –v
gxditview –version
gxditview ––version

Description

gxditview renders the output of GNU *troff* to an X11 display. It uses the standard X11 fonts, which are rendered by the X server.

The output of *troff*(1) is in a page description language documented in *groff_out*(5). Normally, users do not observe this output because the *groff*(1) wrapper automatically runs an output driver to interpret it; *groff*'s **–Z** option leaves it on the standard output stream instead. It can be viewed by explicitly calling "**gxditview** *file*". If the *file* operand is "**–**", *gxditview* will read the standard input stream; *file* cannot be omitted. GNU *troff*'s page description language is device-independent but not device-agnostic. *gxditview* can view it for all typesetter devices, but the quality is device-dependent. *gxditview* will not display output for terminal (*nroff*) devices.

The best results are achieved with the **X*** devices for *troff*'s **–T** option, of which there are four: **–TX75**, **–TX75–12**, **–TX100**, and **–TX100–12**. They differ by the X resolution (75 or 100 dots per inch) and the base point size (10 or 12 points). They are especially built for *gxditview*. When using one of these, *troff* generates output for this device and calls *gxditview* automatically for viewing.

–X produces good results only with **–Tps**, **–TX75**, **–TX75–12**, **–TX100**, and **–TX100–12**. The default resolution for previewing **–Tps** output is 75 dpi; this can be changed with the **–resolution** option.

While *gxditview* is running, the left mouse button brings up a menu with several entries.

Next Page Display the next page.

Previous Page Display the previous page.

Select Page Collect a page number from a dialog box and render it.

Print Collect a command from a dialog box and use it to print *troff* output. The default command initially displayed is controlled by the **printCommand** application resource, and by the **–printCommand** option.

Open Collect a file name from a dialog box and render it. The named file should contain *troff* output. If the file name starts with a bar or pipe symbol, "|" it will be interpreted as a command from which to read.

Quit Exit from **gxditview**.

The menu entries correspond to actions with similar but not identical names, which can also be accessed with keyboard accelerators. The *n*, *Space*, *Return*, and *Next* (*PgDn*) keys are bound to the **NextPage** action. The *p*, *b*, *BackSpace*, *Delete*, and *Prior* (*PgUp*) keys are bound to the **PreviousPage** action. The *p* key with any modifier (Shift, Control, Alt, or Meta) is bound to the **Print** action. The *g* key is bound to the **SelectPage** action. The *o* key is bound to the **OpenFile** action. The *q* key is bound to the **Quit** action. The *r* key is bound to a **Rerasterize** action which rereads the current file, and redisplays the current page; if the current file is a command, the command will be re-executed. Vertical scrolling can be done with the *k* and *j* keys; horizontal scrolling is bound to the *h* and *l* keys. The arrow keys (*up*, *down*, *left*, and *right*) are also bound to the obvious scrolling actions.

The **paperlength** and **paperwidth** commands in the *DESC* file describing a *groff* output device specify the length and width in basic units of the virtual page displayed by *gxditview*; see *groff_font*(5).

X defaults

This program uses the *Dvi* widget from the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
: Specifies the width of the window.

height (class **Height**)
: Specifies the height of the window.

foreground (class **Foreground**)
: Specifies the default foreground color.

font (class **Font**)
: Specifies the font to be used for error messages.

fontMap (class **FontMap**)
: Specifies the mapping from *groff* font names to X font names. This must be a string containing a sequence of lines. Each line contains two whitespace-separated fields: the *groff* font name is first, and the XLFD (X Logical Font Description) second. The default is shown in subsection "Default font map" below.

Default font map

XLFDs are long and unwieldy, so some lines are shown broken and indented below.

```
TR  -adobe-times-medium-r-normal--*-100-*-*-*-*-iso8859-1\n\
TI  -adobe-times-medium-i-normal--*-100-*-*-*-*-iso8859-1\n\
TB  -adobe-times-bold-r-normal--*-100-*-*-*-*-iso8859-1\n\
TBI -adobe-times-bold-i-normal--*-100-*-*-*-*-iso8859-1\n\
CR  -adobe-courier-medium-r-normal--*-100
        -*-*-*-*-iso8859-1\n\
CI  -adobe-courier-medium-o-normal
        --*-100-*-*-*-*-iso8859-1\n\
CB  -adobe-courier-bold-r-normal--*-100-*-*-*-*-iso8859-1\n\
CBI -adobe-courier-bold-o-normal--*-100-*-*-*-*-iso8859-1\n\
HR  -adobe-helvetica-medium-r-normal
        --*-100-*-*-*-*-iso8859-1\n\
HI  -adobe-helvetica-medium-o-normal
        --*-100-*-*-*-*-iso8859-1\n\
HB  -adobe-helvetica-bold-r-normal
        --*-100-*-*-*-*-iso8859-1\n\
HBI -adobe-helvetica-bold-o-normal
        --*-100-*-*-*-*-iso8859-1\n\
NR  -adobe-new century schoolbook-medium-r-normal--*-100
        -*-*-*-*-iso8859-1\n\
NI  -adobe-new century schoolbook-medium-i-normal--*-100
        -*-*-*-*-iso8859-1\n\
NB  -adobe-new century schoolbook-bold-r-normal--*-100
        -*-*-*-*-iso8859-1\n\
NBI -adobe-new century schoolbook-bold-i-normal--*-100
        -*-*-*-*-iso8859-1\n\
S   -adobe-symbol-medium-r-normal--*-100
        -*-*-*-*-adobe-fontspecific\n\
SS  -adobe-symbol-medium-r-normal--*-100
        -*-*-*-*-adobe-fontspecific\n\
```

Options

−help and **−−help** display a usage message, while **−v**, **−version**, and **−−version** show version information; all exit afterward.

gxditview accepts all of the standard X Toolkit command-line options along with the additional options listed below.

−page This option specifies the page number of the document to be displayed.

−backingStore *backing-store-type*
Because redisplay of the rendered *troff* output can take a perceiptible amount of time, this option causes the server to save the window contents so that when it is scrolled around the viewport, the window is painted from contents saved in backing store. *backing-store-type* can be one of **Always**, **WhenMapped** or **NotUseful**.

−printCommand *command*
The default command displayed in the dialog box for the **Print** menu entry will be *command*.

−resolution *res*
troff output file will be displayed at a resolution of *res* dots per inch, unless the *DESC* file contains the **X11** directive, in which case the device resolution will be used. This corresponds to the *Dvi* widget's **resolution** resource. The default is **75**.

−filename *string*
The default file name displayed in the dialog box for the **Open** menu entry will be *string*. This can be either a file name, or a command starting with "**|**".

The following standard X Toolkit command-line arguments are commonly used with *gxditview*.

−bg *color*
This option specifies the color to use for the background of the window. The default is "**white**".

−bd *color*
This option specifies the color to use for the border of the window. The default is "**black**".

−bw *number*
This option specifies the width in pixels of the border surrounding the window.

−fg *color*
This option specifies the color to use for displaying text. The default is "**black**".

−fn *font*
This option specifies the font to be used for displaying widget text. The default is "**fixed**".

−rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

−geometry *geometry*
This option specifies the preferred size and position of the window.

−display *host***:***display*
This option specifies the X server to contact.

−xrm *resourcestring*
This option specifies a resource string to be used.

Exit status

gxditview exits with status **0** on successful operation, and status **2** if the program cannot interpret its command-line arguments.

Environment

GROFF_FONT_PATH
A list of directories in which to seek the selected output device's directory of device and font description files. See *troff*(1) and *groff_font*(5).

Files

/usr/share/X11/app−defaults/GXditview
/usr/share/X11/app−defaults/GXditview−color
define X application defaults for *gxditview*. Users can override these values in the *.Xdefaults* file, normally located in the user's home directory. See *appres*(1) and *xrdb*(1).

/usr/share/groff/1.24.2/font/devX100/DESC
describes the **X100** output device.

*/usr/share/groff/1.24.2/font/devX100/*F
describes the font known as *F* on device **X100**.

/usr/share/groff/1.24.2/font/devX100−12/DESC
describes the **X100−12** output device.

*/usr/share/groff/1.24.2/font/devX100−12/*F
describes the font known as *F* on device **X100−12**.

/usr/share/groff/1.24.2/font/devX75/DESC
describes the **X75** output device.

*/usr/share/groff/1.24.2/font/devX75/*F
describes the font known as *F* on device **X75**.

/usr/share/groff/1.24.2/font/devX75−12/DESC
describes the **X75−12** output device.

*/usr/share/groff/1.24.2/font/devX75−12/*F
describes the font known as *F* on device **X75−12**.

/usr/share/groff/1.24.2/tmac/X.tmac
defines macros for use with the **X100**, **X100−12**, **X75**, and **X75−12** output devices. It is automatically loaded by *troffrc* when any of those output devices is selected.

/usr/share/groff/1.24.2/tmac/Xps.tmac
sets up *troff* to use *gxditview* as a previewer for device-independent output targeting the **ps** output device. It is automatically loaded by *troffrc* when *troff* is given the options **−X** and **−Tps**.

Examples

The following command views this man page with a base point size of 12.

```
groff -TX100-12 -man gxditview.1
```

The quality of the result depends mainly on the chosen point size and display resolution; for rapid previewing, however, something like

```
groff -X -P-resolution -P100 document
```

yields acceptable results.

Authors

gxditview and its predecessor *xditview* were written by Keith Packard (MIT X Consortium), Richard L. Hyde (Purdue), David Slattengren (Berkeley), Malcolm Slaney (Schlumberger Palo Alto Research), Mark Moraes (University of Toronto), and James Clark.

This program is derived from *xditview*; portions of *xditview* originated in *xtroff*, which was derived from *suntroff*.

See also

"X Logical Font Description Conventions", by Jim Flowers and Stephen Gildea.

X(7), *xrdb*(1), *xditview*(1), *groff*(1), *groff_out*(5)

Name

hpftodit – create font description files for use with *groff* and *grolj4*

Synopsis

hpftodit [**–aqs**] [**–i** *n*] *tfm-file map-file font-description*

hpftodit –d *tfm-file* [*map-file*]

hpftodit ––help

hpftodit –v
hpftodit ––version

Description

hpftodit creates a font description file for use with a Hewlett-Packard LaserJet 4-series (or newer) printer and the *grolj4*(1) output driver of *groff*(1), using data from an HP tagged font metric (TFM) file. *tfm-file* is the name of the font's TFM file; Intellifont and TrueType TFM files are supported, but symbol set TFM files are not. *map-file* is a file giving the *groff* special character identifiers for glyphs in the font; this file should consist of a sequence of lines of the form

```
m u c1 c2 ... [# comment]
```

where *m* is a decimal integer giving the glyph's MSL (Master Symbol List) number, *u* is a hexadecimal integer giving its Unicode character code, and *c1*, *c2*, ... are its *groff* glyph names (see *groff_char*(7) for a list). The values can be separated by any number of spaces and/or tabs. The Unicode value must use uppercase hexadecimal digits A–F, and must lack a leading "**0x**", "**u**", or "**U+**". Unicode values corresponding to composite glyphs are decomposed; that is "**u00C0**" becomes "**u0041_0300**". A glyph without a *groff* special character identifier may be named **u***XXXX* if the glyph corresponds to a Unicode value, or as an unnamed glyph "**–––**". If the given Unicode value is in the Private Use Area (PUA) (0xE000–0xF8FF), the glyph is included as an unnamed glyph. Refer to *groff_diff*(1) for additional information about unnamed glyphs and how to access them.

Blank lines and lines beginning with "**#**" are ignored. A "**#**" following one or more *groff* names begins a comment. Because "**#**" is a valid *groff* name, it must appear first in a list of *groff* names if a comment is included, as in

```
3   0023   #   # number sign
```

or

```
3   0023   # sh   # number sign
```

whereas in

```
3   0023   sh #   # number sign
```

the first "**#**" is interpreted as the beginning of the comment.

hpftodit writes output to *font-description,* a file named for the intended *groff* font name; if this operand is "**–**", *hpftodit* writes to the standard output stream. The **–i** option directs *hpftodit* to automatically generate an italic correction, a left italic correction, and a subscript correction for each glyph. See *groff_font*(5).

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–a Include glyphs in the TFM file that are not included in *map-file*. A glyph with corresponding Unicode value is given the name u*XXXX*; a glyph without a Unicode value is included as an unnamed glyph "–––". A glyph with a Unicode value in the Private Use Area (0xE000–0xF8FF) is also included as an unnamed glyph.

This option provides a simple means of adding Unicode-named and unnamed glyphs to a font without including them in the map file, but it affords little control over which glyphs are placed in a regular font and which are placed in a special font. The presence or absence of the **–s** option has some effect on which glyphs are included: without it, only the "text" symbol sets are searched for matching glyphs; with it, only the "mathematical" symbol sets are searched. Nonetheless, restricting the symbol sets searched isn't very selective—many glyphs are placed in both regular and special fonts. Normally, **–a** should be used only as a last resort.

−d Dump information about the TFM file to the standard output stream; use this to ensure that a TFM file is a proper match for a font, and that its contents are suitable. The information includes the values of important TFM tags and a listing (by MSL number for Intellifont TFM files or by Unicode value for TrueType TFM files) of the glyphs included in the TFM file. The unit of measure "DU" for some tags indicates design units; there are 8782 design units per em for Intellifont fonts, and 2048 design units per em for TrueType fonts. Note that the accessibility of a glyph depends on its inclusion in a symbol set; some TFM files list many glyphs but only a few symbol sets.

The glyph listing includes the glyph index within the TFM file, the MSL or Unicode value, and the symbol set and character code that will be used to print the glyph. If *map-file* is given, *groff* names are given for matching glyphs. If only the glyph index and MSL or Unicode value are given, the glyph does not appear in any supported symbol set and cannot be printed.

With the **−d** option, *map-file* is optional, and *output-font* is ignored if given.

−i *n* Generate an italic correction for each glyph so that its width plus its italic correction is equal to *n* thousandths of an em plus the amount by which the right edge of the glyphs's bounding box is to the right of its origin. The minimum value *hpftodit* will write is zero.

Also generate a subscript correction equal to the product of the tangent of the slant of the font and four fifths of the x-height of the font. If a subscript correction greater than the italic correction would result, use a subscript correction equal to the italic correction instead.

Also generate a left italic correction for each glyph equal to *n* thousandths of an em plus the amount by which the left edge of the glyphs's bounding box is to the left of its origin. The left italic correction may be negative.

This option normally is needed only with italic or oblique fonts; a value of 50 (0.05 em) usually is a reasonable choice.

−q Suppress warnings about glyphs in the map file that were not found in the TFM file. Warnings never are given for unnamed glyphs or by glyphs named by their Unicode values. This option is useful when sending the output of *hpftodit* to the standard output stream.

−s Add the **special** directive to the font description file, affecting the order in which HP symbol sets are searched for each glyph. Without this option, the "text" sets are searched before the "mathematical" symbol sets. With it, the search order is reversed.

Exit status

hpftodit exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Files

/usr/share/groff/1.24.2/font/devlj4/DESC
describes the **lj4** output device.

*/usr/share/groff/1.24.2/font/devlj4/*F
describes the font known as *F* on device **lj4**.

/usr/share/groff/1.24.2/font/devlj4/generate/Makefile
is a *make*(1) script that uses *hpftodit*(1) to prepare the *groff* font description files above from HP TFM data; it can be used to regenerate them in the event the TFM files are updated.

/usr/share/groff/1.24.2/font/devlj4/generate/special.awk
is an *awk*(1) script that corrects the Intellifont-based height metrics for several glyphs in the **S** (special) font for TrueType CG Times used in the HP LaserJet 4000 and later.

/usr/share/groff/1.24.2/font/devlj4/generate/special.map
/usr/share/groff/1.24.2/font/devlj4/generate/symbol.map

/usr/share/groff/1.24.2/font/devlj4/generate/text.map
/usr/share/groff/1.24.2/font/devlj4/generate/wingdings.map
map MSL indices and HP Unicode PUA assignments to *groff* special character identifiers.

See also

groff(1), *groff_diff*(1), *grolj4*(1), *groff_font*(5)

Name

indxbib – make inverted index of bibliographic databases

Synopsis

indxbib [**–w**] [**–c** *common-words-file*] [**–d** *dir*] [**–f** *list-file*] [**–h** *min-hash-table-size*] [**–i** *excluded-fields*] [**–k** *max-keys-per-record*] [**–l** *min-key-length*] [**–n** *threshold*] [**–o** *file*] [**–t** *max-key-length*] [*file* ...]

indxbib ––help

indxbib –v
indxbib ––version

Description

indxbib makes an inverted index of the bibliographic databases in each *file* to speed their access by *refer*(1), *lookbib*(1), and *lkbib*(1). The program writes to a temporary file that it later renames to file.*i*. If no *file* operands are present and no **–o** option is given, *indxbib* names the index *Ind.i*.

Bibliographic databases are divided into records by blank lines. Within a record, each field starts with a **%** character at the beginning of a line. Fields have a one-letter name that follows the **%** character.

indxbib stores the values set by the **–c**, **–l**, **–n**, and **–t** options in the index: programs that search the index interpret them, discarding and truncating keys appropriately, and using the original keys to verify that any record found using the index actually contains the keys. This means that a user of an index need not know whether these options were used in the creation of the index, provided that not all the keys to be searched for would have been discarded during indexing and that the user supplies at least the part of each key that would have remained after being truncated during indexing. *indxbib* also stores the value set by the **–i** option in the index for use in verifying records found using it.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–c *common-words-file*
: Read the list of common words from *common-words-file* instead of */usr/share/groff/1.24.2/eign*.

–d *dir*
: Use *dir* as the name of the directory to store in the index, instead of that returned by *getcwd*(2). Typically, *dir* will be a symbolic link whose target is the current working directory.

–f *list-file*
: Read the files to be indexed from *list-file*. If *list-file* is **–**, files will be read from the standard input stream. The **–f** option can be given at most once.

–h *min-hash-table-size*
: Use the first prime number greater than or equal to the argument for the size of the hash table. Larger values will usually make searching faster, but will make the index file larger and cause *indxbib* to use more memory. The default hash table size is 997.

–i *excluded-fields*
: Don't index the contents of fields whose names are in *excluded-fields*. Field names are one character each. If this option is not present, *indxbib* excludes fields **X**, **Y**, and **Z**.

–k *max-keys-per-record*
: Use no more keys per input record than specified in the argument. If this option is not present, the maximum is 100.

–l *min-key-length*
: Discard any key whose length in characters is shorter than the value of the argument. If this option is not present, the minimum key length is 3.

–n *threshold*
: Discard the *threshold* most common words from the common words file. If this option is not present, the 100 most common words are discarded.

−o *basename*
: Name the index basename.*i*.

−t *max-key-length*
: Truncate keys to *max-key-length* in characters. If this option is not present, keys are truncated to 6 characters.

−w
: Index whole files. Each file is a separate record.

Exit status

indxbib exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Files

file.*i* index for *file*

Ind.i default index name

/usr/share/groff/1.24.2/eign
: contains the list of common words. The traditional name, "*eign*", is an abbreviation of "English ignored [word list]".

*indxbib*XXXXXX
: temporary file

See also

"Some Applications of Inverted Indexes on the Unix System", by M. E. Lesk, 1978, AT&T Bell Laboratories Computing Science Technical Report No. 69.

refer(1), *lkbib*(1), *lookbib*(1)

Name

lkbib – search bibliographic databases

Synopsis

lkbib [**−n**] [**−i** *fields*] [**−p** *file*] . . . [**−t** *n*] *key* . . .

lkbib −−help

lkbib −v
lkbib −−version

Description

lkbib searches bibliographic databases for references matching all keywords *key* and writes any references found to the standard output stream. It reads databases given by **−p** options and then (unless **−n** is given) a default database. The default database is taken from the *REFER* environment variable if it is set, otherwise it is */usr/dict/papers/Ind*. For each database *file* to be searched, if an index file.*i* created by *indxbib*(1) exists, then it will be searched instead; each index can cover multiple databases.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−i *string*
: When searching files for which no index exists, ignore the contents of fields whose names are in *string*.

−n Suppress search of default database.

−p *file*
: Search *file*. Multiple **−p** options can be used.

−t *n* Require only the first *n* characters of keys to be given. The default is 6.

Exit status

lkbib exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Environment

REFER
: Assign this variable a file name to override the default database.

Files

/usr/dict/papers/Ind
: Default database to be used if the *REFER* environment variable is not set.

file.*i* Index files.

See also

"Some Applications of Inverted Indexes on the Unix System", by M. E. Lesk, 1978, AT&T Bell Laboratories Computing Science Technical Report No. 69.

refer(1), *lookbib*(1), *indxbib*(1)

Name

lookbib − search bibliographic databases interactively

Synopsis

lookbib [**−i** *string*] [**−t** *n*] *file* ...

lookbib −−help

lookbib −v
lookbib −−version

Description

lookbib writes a prompt to the standard error stream (unless the standard input stream is not a terminal), reads from the standard input a line containing a set of keywords, searches each bibliographic database *file* for references containing those keywords, writes any references found to the standard output stream, and repeats this process until the end of input. For each database *file* to be searched, if an index file.*i* created by *indxbib*(1) exists, then it will be searched instead; each index can cover multiple databases.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−i *string*
: When searching files for which no index exists, ignore the contents of fields whose names are in *string*.

−t *n*
: Require only the first *n* characters of keys to be given. The default is 6.

Exit status

lookbib exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Files

file.*i* Index files.

See also

"Some Applications of Inverted Indexes on the Unix System", by M. E. Lesk, 1978, AT&T Bell Laboratories Computing Science Technical Report No. 69.

refer(1), *lkbib*(1), *indxbib*(1)

Name

mmroff – reference- and PostScript inclusion-handling front end for GNU *roff mm* macro package

Synopsis

mmroff [**–x**] *groff-argument* ...

mmroff ––help

mmroff ––version

Description

mmroff is a simple wrapper for *groff*, used to expand internal references in *mm* documents; see *groff_mm*(7). It runs *groff* with the **–mm** option twice, first with **–z** and **–rRef=1** to populate reference and index files with their corresponding entries, and then without to produce the document. It also handles the inclusion of PostScript images with the **PIC** macro. Documents that do not use these features of *groff mm* (the **INITI**, **IND**, **INDP**, **INITR**, **SETR**, **GETHN**, **GETPN**, **GETR**, **GETST**, and **PIC** macros) do not require *mmroff*.

Options

––help displays a usage message, while **––version** shows version information; both exit afterward.

–x Create or update the reference list file and exit.

Environment

Normally, the path separator in environment variables ending with *PATH* is the colon; this may vary depending on the operating system. For example, Windows uses a semicolon instead.

GROFF_BIN_PATH

Locate *groff* commands in these directories, followed by those in *PATH*. If not set, the installation directory of GNU *roff* executables, */usr/bin*, is searched before *PATH*.

Exit status

mmroff exits with status **0** if the **––help** or **––version** options were specified, or if **–x** was specified and the program successfully created or updated the reference list file. It exits with status **1** if it was unable to create necessary files. Otherwise, it exits with the status of *groff*.

Authors

mmroff was written by Jörgen Hägg of Lund, Sweden.

See also

groff_mm(7), *groff_mmse*(7), *groff*(1), *troff*(1), *tbl*(1), *pic*(1), *eqn*(1)

Name

neqn − format equations for character-cell terminal output

Synopsis

neqn [*eqn-argument* . . .]

Description

neqn invokes the *eqn*(1) command with the **ascii** output device.

GNU *eqn* does not support low-resolution, typewriter-like devices, although it may work adequately for very simple input.

See also

eqn(1)

Name

nroff – format documents with *groff* for terminal ("TTY") devices

Synopsis

nroff [**−abcCEhikpRStUVzZ**] [**−d** *ctext*] [**−d** *string*=*text*] [**−D** *fallback-input-encoding*] [**−I** *inclusion-directory*] [**−K** *input-encoding*] [**−m** *macro-package*] [**−M** *macro-directory*] [**−n** *page-number*] [**−o** *page-list*] [**−P** *postprocessor-argument*] [**−r** *cnumeric-expression*] [**−r** *register*=*numeric-expression*] [**−T** *output-device*] [**−w** *warning-category*] [**−W** *warning-category*] [*file* . . .]

nroff −−help

nroff −v [*other-nroff-option* . . .]
nroff −−version [*other-nroff-option* . . .]

Description

nroff formats documents written in the *groff*(7) language for typewriter-like devices such as terminal emulators. GNU *nroff* emulates the AT&T *nroff* command using *groff*(1). *nroff* generates output via *grotty*(1), *groff*'s terminal output driver, which needs to know the character encoding scheme used by the device. Consequently, acceptable arguments to the **−T** option are **ascii**, **latin1**, and **utf8**; any others are ignored. If neither the *GROFF_TYPESETTER* environment variable nor the **−T** command-line option (which overrides the environment variable) specifies a (valid) device, *nroff* consults the locale to select an appropriate output device. It first tries the *locale*(1) program, then checks several locale-related environment variables; see section "Environment" below. If all of the foregoing fail, *nroff* assumes "**−T ascii**".

The **−a**, **−b**, **−c**, **−C**, **−d**, **−E**, **−i**, **−I**, **−m**, **−M**, **−n**, **−o**, **−r**, **−U**, **−w**, **−W**, and **−z** options have the effects described in *troff*(1). **−c** and **−h** imply "**−P −c**" and "**−P −h**", respectively; **−c** is also meaningful to *troff* itself. Further, GNU *nroff* ignores the AT&T *nroff* options **−e**, **−q**, and **−s**. *groff*(1) documents options **−D**, **−k**, **−K**, **−p**, **−P**, **−R**, **−t**, **−S**, and **−Z**. **−V** causes *nroff* to display the constructed *groff* command on the standard output stream, but does not execute it. **−v** and **−−version** show version information about *nroff* and the programs it runs, while **−−help** displays a usage message; all exit afterward.

Exit status

nroff exits with status **2** if there was a problem parsing its arguments, with status **0** if any of the options **−V**, **−v**, **−−version**, or **−−help** were specified, and with the status of *groff* otherwise.

Environment

Normally, the path separator in environment variables ending with *PATH* is the colon; this may vary depending on the operating system. For example, Windows uses a semicolon instead.

GROFF_BIN_PATH
: Locate *groff* commands in these directories, followed by those in *PATH*. If not set, the installation directory of GNU *roff* executables, */usr/bin*, is searched before *PATH*.

GROFF_TYPESETTER
: specifies the default output device for *groff*.

LC_ALL
LC_CTYPE
LANG
LESSCHARSET
: are pattern-matched in this order for contents matching standard character encodings supported by *groff* in the event no **−T** option is given and *GROFF_TYPESETTER* is unset, or the values specified are invalid.

Files

/usr/share/groff/1.24.2/tmac/tty−char.tmac
: defines fallback definitions of *roff* special characters. These definitions more poorly optically approximate typeset output than those of *tty.tmac* in favor of communicating semantic information. *nroff* loads it automatically.

Notes

Pager programs like *more*(1) and *less*(1) may require command-line options to correctly handle some output sequences; see *grotty*(1).

See also

groff (1), *troff* (1), *grotty*(1), *locale*(1), *roff* (7)

Name

pdfmom – produce PDF documents using the *mom* macro package for *groff*

Synopsis

pdfmom [**−−roff**] [**−Tpdf**] [*groff-options*] [*file* . . .]

pdfmom [**−−roff**] **−Tps** [*pdfroff-options*] [*groff-options*] [*file* . . .]

pdfmom −v
pdfmom −−version

Description

pdfmom is a wrapper around *groff*(1) that facilitates the production of PDF documents from files formatted with the *mom* macros.

If the **−−roff** option is used, the wrapper can be used with macro packages other than *groff_mom*(7). This is also true if the wrapper is renamed or linked as a pseudonym; for example, creating a *pdfms* link pointing to the *pdfmom* executable makes a wrapper for producing PDFs with the *ms* package.

pdfmom prints to the standard output, so output must usually be redirected to a destination file. The size of the final PDF can be reduced by piping the output through *ps2pdf*(1).

If called with the **−Tpdf** option (which is the default), *pdfmom* processes files using *groff*'s native PDF driver, *gropdf*(1). If **−Tps** is given, processing is passed over to *pdfroff*, which uses *groff*'s PostScript driver. In either case, multiple runs of the source file are performed in order to satisfy any forward references in the document.

pdfmom accepts all the same options as *groff*. If **−Tps** is given, the options associated with *pdfroff* are accepted as well. When *pdfmom* calls *pdfroff*, the options " **−mpdfmark −mom −−no−toc**" options are implied and should not be given on the command line. Equally, it is not necessary to supply the **−mom** or **−m mom** options when **−Tps** is absent.

PDF integration with the *mom* macros is discussed in full in the manual "Producing PDFs with *groff* and *mom*", which was itself produced with *pdfmom*.

If called with the **−v** or **−−version** options, *pdfmom* displays its version information and exits. Using the **−−help** option displays a usage message and exits.

Authors

pdfmom was written by Deri James and Peter Schaffter, and is maintained by James.

See also

/usr/share/doc/groff/pdf/mom−pdf.pdf
: "Producing PDFs with *groff* and *mom*", by Deri James and Peter Schaffter. This file, together with its source, *mom−pdf.mom*, is part of the *groff* distribution.

groff(1), *gropdf*(1), *pdfroff*(1), *ps2pdf*(1)

Name

pfbtops − translate PostScript Printer Font Binary files to Printer Font ASCII

Synopsis

pfbtops [*pfb-file*]

pfbtops −−help

pfbtops −v
pfbtops −−version

Description

pfbtops translates a PostScript Type 1 font in Printer Font Binary (PFB) format to Printer Font ASCII (PFA) format, splitting overlong lines in text packets into smaller chunks. If *pfb-file* is omitted, the PFB file will be read from the standard input stream. The PFA font will be written on the standard output stream. PostScript fonts for MS-DOS were historically supplied in PFB format. Use of a PostScript Type 1 font with *groff* requires conversion of its metrics (AFM file) to a *groff* font description file; see *afmtodit*(1).

The **−−help** option displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

See also

grops(1), *gropdf*(1)

Name

pic – compile pictures for *troff* or TeX

Synopsis

pic [**−CnSU**] [*file* . . .]

pic −t [**−cCSUz**] [*file* . . .]

pic −−help

pic −v
pic −−version

Description

The GNU implementation of *pic* is part of the *groff*(1) document formatting system. *pic* is a *troff*(1) preprocessor that translates descriptions of diagrammatic pictures embedded in *roff*(7) or TEX input into the language understood by TEX or *troff*. It copies each *file*'s contents to the standard output stream, translating each *picture* between lines starting with **.PS** and any of **.PE**, **.PF**, or **.PY**. End a *pic* picture with **.PE** to leave the drawing position at the bottom of the picture, and with **.PF** or **.PY** to leave it at the top. Normally, *pic* is not executed directly by the user, but invoked by specifying the **−p** option to *groff*(1). If no *file* operands are present, or if *file* is "**−**", *pic* reads the standard input stream.

It is the user's responsibility to provide appropriate definitions of the **PS**, **PE**, and one or both of the **PF** and **PY** macros. When a macro package does not supply these, obtain simple definitions with the *groff* option **−mpic**; these horizontally center each picture.

GNU *pic* supports **PY** as a synonym of **PF** to work around a name space collision with the *mm* macro package, which defines **PF** as a page footer management macro. Use **PF** preferentially unless a similar problem faces your document.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−c Be more compatible with *tpic*; implies **−t**. Lines beginning with **** are not passed through transparently. Lines beginning with **.** are passed through with the initial **.** changed to ****. GNU *pic* gives a line beginning with **.ps** special treatment: it takes an optional integer argument specifying the line thickness (pen size) in milliinches, a missing argument restores the previous line thickness, and the default line thickness is 8 milliinches. The line thickness thus specified takes effect only when a non-negative line thickness has not been specified by use of the **thickness** attribute or by setting the **linethick** variable.

−C Recognize **.PS**, **.PE**, **.PF**, and **.PY** even when followed by a character other than space or newline.

−n Don't use *groff* extensions to the *troff* drawing commands. Specify this option if a postprocessor you're using doesn't support these extensions, described in *groff_out*(5). This option also causes *pic* not to use zero-length lines to draw dots in *troff* mode.

−S Enable safer mode and ignore any subsequent **−U** option. In *safer mode, pic* ignores **sh** commands, which can be useful when operating on untrustworthy input. *pic* operates in safer mode by default.

−t Produce TEX output.

−U Operate in *unsafe mode;* **sh** commands are interpreted.

−z In TEX mode, draw dots using zero-length lines.

The following options supported by other versions of *pic* are ignored.

−D Draw all lines using the \D escape sequence. GNU *pic* always does this.

−T *dev* Generate output for the *troff* device *dev*. This option is unnecessary in GNU *pic* because the *troff* it generates is device-independent.

Usage

This section primarily discusses the differences between GNU *pic* and the Eighth Edition Unix version of AT&T *pic* (1985). Many of these differences also apply to later versions of AT&T *pic*.

TEX mode

TEX-compatible output is produced when the **−t** option is specified. You must use a TEX driver that supports *tpic* version 2 specials. (*tpic* was a fork of AT&T *pic* by Tim Morgan of the University of California at Irvine that diverged from its source around 1984. It is best known today for lending its name to a group of **\special** commands it produced for TEX.)

Lines beginning with **** are passed through unaltered except for a **%** suffix to avoid unwanted spaces. Use this feature to change fonts or the value of **\baselineskip**. Other applications may produce undesirable results; use at your own risk. By default, lines beginning with a dot are not treated specially—but see the **−c** option.

In TEX mode, *pic* defines a vbox called **\graph** for each picture. Use GNU *pic*'s **figname** command to change the name of the vbox. You must print that vbox yourself using the command

```
\centerline{\box\graph}
```

for instance. Since the vbox has a height of zero—it is defined with **\vtop**—this produces slightly more vertical space above the picture than below it;

```
\centerline{\raise 1em\box\graph}
```

would avoid this. To give the vbox a positive height and a depth of zero (as used by LATEX's *graphics.sty*, for example), define the following macro in your document.

```
\def\gpicbox#1{%
  \vbox{\unvbox\csname #1\endcsname\kern 0pt}}
```

You can then simply say **\gpicbox{graph}** instead of **\box\graph**.

Commands

Several commands new to GNU *pic* accept delimiters, shown in their synopses as braces **{ }**. Nesting of braces is supported. Any other characters (except a space, tab, or newline) may be used as alternative delimiters, in which case the members of a given pair must be identical. GNU *pic* recognizes double-quoted strings within delimiters of either kind; such strings may contain the delimiter character or unbalanced braces.

for *variable* **=** *expr1* **to** *expr2* [**by** [*****]*expr3*] **do** *X body X*

Set *variable* to *expr1*. While the value of *variable* is less than or equal to *expr2*, do *body* and increment *variable* by *expr3*; if **by** is not given, increment *variable* by 1. *expr3* can be negative, in which case *variable* is then tested whether it is greater than or equal to *expr2*. A ***** prefix on *variable* multiplies it by *expr3* (which must be greater than zero) at each iteration rather than incrementing it. If the range constraint on *expr3* isn't met, the loop does not execute. *X* can be any character not in *body*.

if *expr* **then** *X if-true X* [**else** *Y if-false Y*]

Evaluate *expr*; if it is non-zero then do *if-true*, otherwise do *if-false*. *X* can be any character not in *if-true*. *Y* can be any character not in *if-false*.

print *arg* ...

Catenate and write arguments to the standard error stream followed by a newline. Each *arg* must be an expression, a position, or text. This feature is useful for debugging.

command *arg* ...

Catenate arguments and pass them as a line to *troff* or TEX. Each *arg* must be an expression, a position, or text. **command** allows the values of *pic* variables to be passed to the formatter. Thus,

```
.PS
x = 14
command ".ds string x is " x "."
.PE
\*[string]
```

produces

```
x is 14.
```

when formatted with *troff*.

sh *X command X*
: Pass *command* to a shell via *system*(3). Ignored if **–U** option not specified.

copy "*filename***"**
: Include *filename* at this point in the file.

copy ["*filename*"] **thru** *X body X* [**until** "*word*"]
copy ["*filename*"] **thru** *macro* [**until** "*word*"]
: This construct does *body* once for each line of *filename*; the line is split into blank-delimited words, and occurrences of **$***i* in *body*, for *i* between 1 and 9, are replaced by the *i*-th word of the line. If *filename* is not given, lines are taken from the current input up to **.PE**. If an **until** clause is specified, GNU *pic* reads lines only until encountering one beginning with *word*; which it then discards. *X* can be any character not in *body*. For example,

```
.PS
copy thru % circle at ($1,$2) % until "END"
1 2
3 4
5 6
END
box
.PE
```

and

```
.PS
circle at (1,2)
circle at (3,4)
circle at (5,6)
box
.PE
```

are equivalent. The commands to be performed for each line can also be taken from a macro defined earlier by giving the name of the macro as the argument to **thru**. The argument after **thru** is looked up as a macro name first; if not defined, its first character is interpreted as a delimiter.

reset
reset *pvar1*[**,**] *pvar2* ...
: Reset predefined variables *pvar1*, *pvar2* ... to their default values; if no arguments are given, reset all predefined variables to their default values. Variable names may be separated by commas, spaces, or both. Assigning a value to **scale** also causes all predefined variables that control dimensions to be reset to their default values times the new value of **scale**.

plot *expr* ["*text*"]
: Create a text object by using *text* as a format string for *sprintf*(3) with an argument of *expr*. If *text* is omitted, **"%g"** is implied. Attributes can be specified in the same way as for a normal text object. *Caution:* be very careful that you specify an appropriate format string in *text; pic*'s validation of it is limited. **plot** is deprecated in favour of **sprintf**.

var **:=** *expr*
: Update an existing variable. *var* must already be defined, and *expr* is assigned to *var* without creating a variable local to the current block. (By contrast, **=** defines *var* in the current block if it is not already defined there, and then changes the value in the current block only.) For example,

```
.PS
x = 3
y = 3
[
x := 5
y = 5
]
```

```
print x   y
.PE
```

writes

```
5 3
```

to the standard error stream.

Expressions

The syntax for expressions has been significantly extended.

x **^** *y* (exponentiation)
sin(*x*)
cos(*x*)
atan2(*y*, *x*)
log(*x*) (base 10)
exp(*x*) (base 10, i.e. 10^x)
sqrt(*x*)
int(*x*)
rand() (return a random number between 0 and 1)
rand(*x*) (return a random number between 1 and *x*; deprecated)
srand(*x*) (set the random number seed)
max(*e1*, *e2*)
min(*e1*, *e2*)
!*e*
e1 **&&** *e2*
e1 **||** *e2*
e1 **==** *e2*
e1 **!=** *e2*
e1 **>=** *e2*
e1 **>** *e2*
e1 **<=** *e2*
e1 **<** *e2*
"*str1*" **==** "*str2*"
"*str1*" **!=** "*str2*"

String comparison expressions must be parenthesised in some contexts to avoid ambiguity.

Other changes

A bare expression, *expr*, is acceptable as an attribute; it is equivalent to " *dir expr*", where *dir* is the current direction. For example, "**line 2i**" draws a line 2 inches long in the current direction. The 'i' (or 'I') character is ignored; to use another measurement unit, set the **scale** variable to an appropriate value.

The maximum width and height of the picture are taken from the variables **maxpswid** and **maxpsht**. Initially, these have values 8.5 and 11, respectively.

Scientific notation is allowed for numbers, as with "**x = 5e−2**".

Text attributes can be compounded. For example, "**"foo" above ljust**" is valid.

There is no limit to the depth to which blocks can be nested. For example,

```
[A: [B: [C: box ]]] with .A.B.C.sw at 1,2
circle at last [].A.B.C
```

is acceptable.

Arcs have compass points determined by the circle of which the arc is a part.

Circles, ellipses, and arcs can be dotted or dashed. In TEX mode, splines can be dotted or dashed as well.

Boxes can have rounded corners. The **rad** attribute specifies the radius of the quarter-circles at each corner. If no **rad** or **diam** attribute is given, a radius of **boxrad** is used. Initially, **boxrad** has a value of 0. A box with rounded corners can be dotted or dashed.

Boxes can have slanted sides, generalizing them from rectangles to parallelograms. The **xslanted** and **yslanted** attributes specify the *x* and *y* offsets of the box's upper right corner from its default position.

The **.PS** line accepts a second argument specifying a maximum height for the picture. If a width of zero is specified, it is ignored when computing the scaling factor for the picture. GNU *pic* always scales a picture by the same amount vertically and horizontally. This differs from DWB 2.0 *pic*, which may change the picture's aspect ratio if a height is specified.

Each text object has an associated invisible box that determines its compass points and implicit motion. The dimensions of the box are taken from its width and height attributes. If the width attribute is not supplied, the value of **textwid** is assumed. If the height attribute is not supplied, the height defaults to the number of text strings associated with the object times **textht**. Initially, **textwid** and **textht** have values of 0.

In (almost all) places where a quoted text string can be used, an expression of the form

sprintf("*format***",** *arg***, ...)**

can be used instead; it transforms its arguments per *format*, which should be a string as described in *printf*(3), and appropriate to the quantity of arguments supplied. GNU *pic* supports only the conversion specifiers **e**, **E**, **f**, **g**, **G**, and **%**, modifiers "**#**", "**−**", "**+**", and " " [space]), a minimum field width, and an optional precision.

The thickness of the lines used to draw objects is controlled by the **linethick** variable, which is measured in points. A negative value indicates the default thickness. In TEX output mode when the **−c** option is not given, this means 8 milliinches. In *troff* and TEX **−c** output modes, the default thickness corresponds to the type size. (Thus, if the type size is 10 points, a line is 10 points thick.) A **linethick** value of zero draws the thinnest possible line supported by the output device. Initially, **linethick** has a value of −1. A **thick**[**ness**] attribute is also available. For example, "**circle thickness 1.5**" draws a circle with a line thickness of 1.5 points. The thickness of lines is not affected by the value of the **scale** variable, nor by the width or height given in the **.PS** line.

Boxes (including boxes with rounded corners or slanted sides), circles, and ellipses can be filled by giving them an attribute of **fill**[**ed**], which takes an optional expression argument with a value between 0 and 1; 0 fills it with white, 1 with black, and values in between with a proportionally gray shade. A value greater than 1 is interpreted as the shade of gray that is being used for text and lines. Normally this is black, but output devices may provide a mechanism for changing this. Without an argument, the value of the variable **fillval** is used. Initially, **fillval** has a value of 0.5. The **invisible** attribute does not affect the filling of objects. Text associated with a filled object is added after the object is filled, so that the text is not obscured by the filling.

Additional modifiers are available to draw colored objects: **outline**[**d**] sets the color of the outline, **shaded** the fill color, and **colo**[**u**]**r**[**ed**] sets both. All expect a subsequent string argument specifying the color.

```
circle shaded "green" outline "black"
```

Color is not yet supported in TEX mode. Device macro files like *ps.tmac* declare color names; you can define additional ones with the **defcolor** request (see *groff*(7)). *pic* assumes at the beginning of each picture that the stroke and fill colors are set to the device defaults.

To change the name of the vbox in TEX mode, set the pseudo-variable **figname** (which is actually a specially parsed command) within a picture. For example,

```
.PS
figname = foobar;
circle "dig here";
.PE
```

makes the picture available in the box **\foobar**.

Arrow heads are drawn as solid triangles if the variable **arrowhead** is non-zero and either TEX mode is enabled or the **−n** option is not used. Initially, **arrowhead** has a value of 1. Solid arrow heads are always filled with the current outline (stroke) color.

The *troff* output of *pic* is device-independent. The **−T** option is therefore redundant. Except where noted, all measurements and dimensions use inches implicitly; they are never interpreted as *troff* basic units.

Objects can have an **aligned** attribute, but it is supported only by the *grops*(1) and *gropdf*(1) output drivers. Any text associated with an **aligned** object is rotated about the object's center such that it is oriented along a line connecting the start and end points of the object. **aligned** has no effect on objects whose start and end points are coincident.

In places where *n***th** is allowed, `*expr*'**th** is also allowed. "'**th**" is a single token: no space is allowed between the apostrophe and the "**th**". Consider the following example.

```
for i = 1 to 4 do {
   box
   move right 0.5
}
for i = 1 to 3 do {
   arrow from `i'th box.ne to `i+1'th box.sw
}
```

Arbitrary polygons can be drawn using the **polygon** keyword followed by a series of $n-1$ line segments, where n is the number of edges of the polygon. This allows GNU *troff* to interpret the line segments as a complete object such that the **filled** and **shaded** attributes may be used. The last user-specified line segment determines the polygon's final drawing position and direction. For example, draw and fill a triangle with the following.

```
polygon up 1 then right 1 down 0.5 fill 0.5
```

Two reference point suffixes permit the positioning of drawing elements relative to a polygon. "**.vertex** *expr*" locates the vertices, and "**.midpoint** *expr*" for locating the midpoints of edges. GNU **pic** numbers vertices and midpoints in drawing order starting from **1 .** You can abbreviate **.vertex** as **.v** or **.ver**, and **.midpoint** as **.mid**.

For example,

```
arrow down
polygon up 0.5 right 1 \
then down 0.5 right 1 \
then down 0.5 left 1 \
with .v2 at last line.end
arrow down from last polygon.v4
```

creates and correctly places a flowchart decision diamond.

.center (**.c**) is also available, but compass points do not work as expected and should not be used with polygons.

Converting *pic* to other image formats

To create a stand-alone graphics file from a *pic* file, first compose the picture. Bracket your *pic* code with **.PS** and **.PE** tokens. *groff* requests that don't produce formatted output may precede **.PS**, but format no text, not even any injected by a macro package, which may include a page number even on the first page, as *mm* does by default. Consider writing a "raw" *roff* document that uses no macro package.

Next, convert the *roff*/*pic* input into the desired format. *groff* distributes a simple utility, *pic2graph*(1), for this purpose. Other possibilities exist, particularly if you first transform your picture into PostScript format with "**groff −T ps**". However, such a PostScript file lacks bounding box information; *roff* formatters produce page-sized output. Several tools with names beginning "psto" or "ps2" exist that can infer the bounding box and perform a format conversion. One of these is the PostScript interpreter Ghostscript (*gs*(1)), which exposes format converters via its **−sDEVICE=** option. "**gs −−help**" lists available devices.

Alternatively, produce a PDF with "**groff −T pdf**"; *gropdf*(1)'s **−p** option sets the MediaBox of the file.

The Encapsulated PostScript File (EPS) format is still sometimes seen. The aforementioned Ghostscript offers *ps2epsi*(1), and a standalone package and command *ps2eps*(1) is also available.

For raster image formats, use *pstopnm*(1); the resulting *pnm*(5) file can be then converted to virtually any image format using the *netpbm* tools.

GNU *plotutils* offers a *pic2plot*(1) utility for converting *pic* input to a wide variety of other image formats, including SVG.

Exit status

pic exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Files

/usr/share/groff/1.24.2/tmac/pic.tmac

offers simple definitions of the **PS**, **PE**, **PF**, and **PY** macros. Load it with the **mso** request when eschewing a full-service macro package, or using one that doesn't supply its own definitions for them.

Caveats

The spacing of dots and dashes in broken lines scales with other graphics, and is configurable. For example,

```
line from 0,0 to 100,0 dotted 5
```

spaces the dots in a dotted line 5 units apart. The spacing of dashes can be set the same way. The **dashwid** variable sets the length of dashes.

Bugs

Characters that are invalid as input to GNU *troff* (see the *groff* Texinfo manual or *groff_char*(7) for a list) are rejected even in TEX mode.

Research Tenth Edition Unix *pic*'s **fillval** interprets 0 as black and 1 as white, incompatibly with GNU *pic*.

See also

/usr/share/doc/groff/pic.ps

"Making Pictures with GNU pic", by Eric S. Raymond. This file, together with its source, *pic.ms*, is part of the *groff* distribution.

"PIC—A Graphics Language for Typesetting: User Manual", by Brian W. Kernighan, 1984 (revised 1991), AT&T Bell Laboratories Computing Science Technical Report No. 116

ps2eps is available from CTAN mirrors, e.g., ftp://ftp.dante.de/tex-archive/support/ps2eps/.

W. Richard Stevens, *Turning PIC into HTML*

W. Richard Stevens, *Examples of* pic *Macros*

troff(1), *groff_out*(5), *tex*(1), *gs*(1), *ps2eps*(1), *pstopnm*(1), *ps2epsi*(1), *pnm*(5)

Name

pic2graph – convert a *pic* diagram into a cropped image

Synopsis

pic2graph [**–unsafe**] [**–format** *output-format*] [**–eqn** *delimiters*] [*convert-argument* ...]

pic2graph ––help

pic2graph –v
pic2graph ––version

Description

pic2graph reads a *pic*(1) program from the standard input and writes an image file, by default in Portable Network Graphics (PNG) format, to the standard output. It furthermore translates *eqn*(1) constructs, so it can be used to generate images of mathematical formulae.

The input PIC code should *not* be wrapped with the **.PS** and **.PE/.PF/.PY** macros that normally guard it within *groff*(1) documents.

Arguments not recognized by *pic2graph* are passed to the ImageMagick or GraphicsMagick program *convert*(1). By specifying these, you can give your image a border, set the image's pixel density, or perform other useful transformations.

The output image is clipped using *convert*'s **–trim** option to the smallest possible bounding box that contains all the black pixels.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–eqn *delimiters*
: Use *delimiters* as the opening and closing characters that delimit *eqn* directives; the default is "$$". The option argument *delimiters* should be a two-character string, but an empty string ("") is accepted as a directive to disable *eqn* processing.

–format *output-format*
: Write the image in *output-format*, which must be understood by *convert*; the default is PNG.

–unsafe
: Run *groff* in *unsafe* mode, enabling the PIC command **sh** to execute arbitrary Unix shell commands. The *groff* default is to forbid this.

Environment

GROFF_TMPDIR
TMPDIR
TMP
TEMP
: These environment variables are searched in the order shown to determine the directory where temporary files will be created. If none are set, */tmp* is used.

Authors

pic2graph was written by Eric S. Raymond, based on a recipe by W. Richard Stevens.

See also

W. Richard Stevens, *Turning PIC into HTML*

eqn2graph(1), *grap2graph*(1), *pic*(1), *eqn*(1), *groff*(1), *convert*(1)

Name

preconv – prepare files for typesetting with *groff*

Synopsis

preconv [**−dr**] [**−D** *fallback-encoding*] [**−e** *encoding*] [*file* ...]

preconv −h
preconv −−help

preconv −v
preconv −−version

Description

preconv reads each *file*, converts its encoded characters to a form *troff* (1) can interpret, and sends the result to the standard output stream. Currently, this means that code points in the range 0–127 in ISO 646:1991 IRV (US-ASCII), ISO 8859, or Unicode remain as-is and the remainder are converted to the *groff* special character form "**\[u***XXXX***]**", where *XXXX* is a hexadecimal number of four to six digits corresponding to a Unicode code point. By default, *preconv* also inserts a *roff* **.lf** request at the beginning of each *file*, identifying it for the benefit of later processing (including diagnostic messages); the **−r** option suppresses this behavior.

In typical usage scenarios, *preconv* need not be run directly; instead it should be invoked with the **−k** or **−K** options of *groff*. If no *file* operands are present, or if *file* is "**−**", *preconv* reads the standard input stream.

preconv selects an input encoding with the following algorithm, stopping at the first success.

1. If the input encoding has been explicitly specified with option **−e**, use it.
2. If the input starts with a Unicode Byte Order Mark, select UTF-8, UTF-16, or UTF-32 accordingly.
3. If the input stream is seekable, check the first two input lines for a GNU Emacs file-local variable identifying the character encoding, here referred to as the "coding tag". If found, use it.
4. If the input stream is seekable, and if the *uchardet* library is available on the system, use it to try to infer the encoding of the file.
5. If the **−D** option specifies an encoding, use it.
6. Use the encoding specified by the current locale (*LC_CTYPE*), unless the locale is "C", "POSIX", or empty, in which case assume ISO Latin-1 (8859-1).

The coding tag and *uchardet* methods in the above procedure rely upon a seekable input stream; when *preconv* reads from a pipe, the stream is not seekable, and these detection methods are skipped. If character encoding detection of your input is unreliable, arrange for one of the other methods to succeed by using *preconv*'s **−D** or **−e** options, or by configuring your locale appropriately. *groff* also supports a *GROFF_ENCODING* environment variable, which can be overridden by its **−K** option. Valid values for (or parameters to) all of these are enumerated in the lists of recognized coding tags in the next subsection, and are further influenced by *iconv* library support.

Coding tags

Text editors that support more than a single character encoding need tags within the input files to mark the file's encoding. While it is possible to guess the right input encoding with the help of heuristics that produce good results for a preponderance of natural language texts, they are not absolutely reliable. Heuristics can fail on inputs that are too short or don't represent a natural language.

Consequently, *preconv* supports the coding tag convention used by GNU Emacs (with some restrictions). This notation appears in specially marked regions of an input file designated for "file-local variables".

preconv interprets the following syntax if it occurs in a *roff* comment in the first or second line of the input. Both "\"" and "\#" comment forms are recognized, but the control (or no-break control) character must be the default and must begin the line. Similarly, the escape character must be the default.

```
-*- [...;] coding: encoding[; ...] -*-
```

The only variable *preconv* interprets is "coding", which can take the values listed below.

The following list comprises all MIME "charset" parameter values recognized, case-insensitively, by *preconv*.

big5, cp1047, euc–jp, euc–kr, gb2312, iso–8859–1, iso–8859–2, iso–8859–5, iso–8859–7, iso–8859–9, iso–8859–13, iso–8859–15, koi8–r, us–ascii, utf–8, utf–16, utf–16be, utf–16le

In addition, the following list of other coding tags is recognized, each of which is mapped to an appropriate value from the list above.

ascii, chinese–big5, chinese–euc, chinese–iso–8bit, cn–big5, cn–gb, cn–gb–2312, cp878, csascii, csisolatin1, cyrillic–iso–8bit, cyrillic–koi8, euc–china, euc–cn, euc–japan, euc–japan–1990, euc–korea, greek–iso–8bit, iso–10646/utf8, iso–10646/utf–8, iso–latin–1, iso–latin–2, iso–latin–5, iso–latin–7, iso–latin–9, japanese–euc, japanese–iso–8bit, jis8, koi8, korean–euc, korean–iso–8bit, latin–0, latin1, latin–1, latin–2, latin–5, latin–7, latin–9, mule–utf–8, mule–utf–16, mule–utf–16be, mule–utf–16–be, mule–utf–16be–with–signature, mule–utf–16le, mule–utf–16–le, mule–utf–16le–with–signature, utf8, utf–16–be, utf–16–be–with–signature, utf–16be–with–signature, utf–16–le, utf–16–le–with–signature, utf–16le–with–signature

Any "–dos", "–unix", or "–mac" suffix on a coding tag (which indicates the end-of-line convention used in the file) is ignored during comparison with the above tags.

iconv support

While *preconv* recognizes all of the coding tags listed above, it is capable on its own of interpreting only two encodings: ISO Latin-1 and and UTF-8. ISO 646:1991 IRV (US-ASCII) is a proper subset of both these encodings. If *iconv*(3) support is configured at compile time and available at run time, all others are passed to *iconv* library functions, which may recognize many additional encoding strings. The command "**preconv –v**" discloses whether *iconv* support is configured.

The use of *iconv* means that characters in the input that encode invalid code points for that encoding may be dropped from the output stream or mapped to the Unicode replacement character (U+FFFD). Compare the following examples using the input "café" (note the "e" with an acute accent), which due to its short length challenges inference of the encoding used.

```
printf 'caf\351\n' | LC_ALL=en_US.UTF-8 preconv
printf 'caf\351\n' | preconv -e us-ascii
printf 'caf\351\n' | preconv -e latin-1
```

The fate of the accented "e" differs in each case. In the first, *uchardet* fails to detect an encoding (though the library on your system may behave differently) and *preconv* falls back to the locale settings, where octal 351 starts an incomplete UTF-8 sequence and results in the Unicode replacement character. In the second, it is not a representable character in the declared input encoding of US-ASCII and is discarded by *iconv*. In the last, it is correctly detected and mapped.

Limitations

preconv cannot perform any transformation on input that it cannot see. Examples include files that are interpolated by preprocessors that run subsequently, including *soelim*(1); files included by *troff* itself through "**so**" and similar requests; and string definitions passed to *troff* through its **–d** command-line option.

preconv assumes that its input uses the default escape character, a backslash ****, and writes special character escape sequences accordingly.

Options

–h and **––help** display a usage message, while **–v** and **––version** show version and configuration information; all exit afterward.

–d Emit debugging messages to the standard error stream.

–D *fbenc* Select the fallback encoding *fbenc* if all detection methods fail.

–e *enc* Skip detection and select encoding *enc*; see *groff*'s **–K** option.

–r Write files "raw"; do not add **.lf** requests.

Exit status

preconv exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

See also

groff (1), *iconv*(3), *locale*(7)

Name

refer – process bibliographic references for *groff*

Synopsis

refer [**−bCenPRS**] [**−a** *n*] [**−B** *field***.***macro*] [**−c** *fields*] [**−f** *n*] [**−i** *fields*] [**−k** *field*] [**−l** *range-expression*] [**−p** *database-file*] [**−s** *fields*] [**−t** *n*] [*file* ...]

refer −−help

refer −v
refer −−version

Description

The GNU implementation of *refer* is part of the *groff*(1) document formatting system. *refer* is a *troff*(1) preprocessor that prepares bibliographic citations by looking up keywords specified in a *roff*(7) input document, obviating the need to type such annotations, and permitting the citation style in formatted output to be altered independently and systematically. It copies the contents of each *file* to the standard output stream, looking up each *citation* between lines starting with **.[** and **.]** and replacing it with a bibliographic reference. GNU *refer* furthermore interprets lines between those starting with **.R1** and **.R2** as instructions directing how citations are to be processed. *refer* interprets and generates *roff* **lf** requests so that file names and line numbers in messages produced by commands that read its output correctly describe the source document. Normally, *refer* is not executed directly by the user, but invoked by specifying the **−R** option to *groff*(1). If no *file* operands are present, or if *file* is "**−**", *refer* reads the standard input stream.

A citation identifies a work by reference to a *bibliographic record* detailing it. Select a work from a database of records by listing keywords that uniquely identify its entry. Alternatively, a document can specify a record for the work at the point its citation occurs. A document can use either or both strategies as desired.

For each citation, *refer* produces a *mark* in the text, like a superscripted footnote number or "[Lesk1978a]". A mark consists of a *label* between *brackets.* The mark can be separated from surrounding text and from other labels in various ways. *refer* produces *roff* language requests usable by a document or a macro package such as *me*, *mm*, *mom*, or *ms* to produce a formatted reference for each citation. A citation's reference can be output immediately after it occurs (as with footnotes), or references may *accumulate*, with corresponding output appearing later in the document (as with endnotes). When references accumulate, multiple citations of the same reference produce a single formatted entry.

Interpretation of lines between **.R1** and **.R2** tokens as preprocessor commands is a GNU *refer* extension. Documents employing this feature can still be processed by AT&T *refer* by adding the lines

```
.de R1
.ig R2
..
```

to the beginning of the document. The foregoing input causes *troff* to ignore everything between **.R1** and **.R2**. The effects of some *refer* commands can be achieved by command-line options; these are supported for compatibility with AT&T *refer*. It is usually more convenient to use commands.

Bibliographic records

A bibliographic record describes a referenced work in sufficient detail that it may be cited to accepted standards of scholarly and professional clarity. The record format permits annotation and extension that a document may use or ignore. A record is a plain text sequence of *fields,* one per line, each consisting of a percent sign **%**, an alphanumeric character classifying it, one space, and its contents. If a field's contents are empty, the field is ignored.

Frequently, such records are organized into a *bibliographic database,* with each entry separated by blank lines or file boundaries. This practice relieves documents of the need to maintain bibliographic data themselves. The programs *lookbib*(1) and *lkbib*(1) consult a bibliographic database, and *indxbib*(1) indexes one to speed retrieval from it, reducing document processing time. Use of these tools is optional.

The conventional uses of the bibliographic field entries are as follows. Within a record, fields other than **%A** and **%E** replace previous occurrences thereof. The ordering of multiple **%A** and **%E** fields is significant.

%A names an author. If the name contains a suffix such as "Jr." or "III", it should be separated from the surname by a comma. We recommend always supplying an **%A** field or a **%Q** field.

%B records the title of the book within which a cited article is collected. See **%J** and **%T**.

%C names the city or other place of publication.

%D indicates the date of publication. Specify the year in full. If the month is specified, use its name rather than its number; only the first three letters are required. We recommend always supplying a **%D** field; if the date is unknown, use "**in press**" or "**unknown**" as its contents.

%E names an editor of the book within which a cited article is collected. Where a work has editors but no authors, name the editors in **%A** fields and append "**, (ed.)**" or "**, (eds.)**" to the last of these.

%G records the U.S. government ordering number, ISBN, DOI, or other unique identifier.

%I names the publisher (issuer).

%J records the name of the journal within which a cited article is collected. See **%B** and **%T**.

%K lists keywords intended to aid searches.

%L is a label; typically unused in database entries, it can override the label format otherwise determined.

%N records the issue number of the journal within which a cited article is collected.

%O presents additional ("other") information, typically placed at the end of the reference.

%P lists the page numbers of a cited work that is part of a larger collection. Specify a range with *m*−*n*.

%Q names an institutional author when no **%A** fields are present. Only one **%Q** field is permitted.

%R is an identifier for a report, thesis, memorandum, or other unpublished work.

%S records the title of a series to which the cited work belongs.

%T is the work's title. See **%B** and **%J**.

%V is the volume number of the journal or book containing the cited work.

%X is an annotation. By convention, it is not formatted in the citing document.

If the obsolescent "accent strings" feature of the *ms* or *me* macro packages is used, such strings should follow the character to be accented; an *ms* document must call the **AM** macro before using them. Do not quote accent strings: use one **** rather than two. See *groff_char*(7) for a modern approach to the problem of diacritics.

Citations

Citations have a characteristic format.

```
.[opening-text
flags keyword ...
field
...
.]closing-text
```

opening-text, *closing-text*, and *flags* are optional, and only one *keyword* or *field* need be specified. If keywords are present, *refer* searches the bibliographic database(s) for a unique reference matching them. Multiple matches are an error; add more keywords to disambiguate the reference. In the absence of keywords, *fields* constitute the bibliographic record. Otherwise, *fields* specify additional data to replace or supplement those in the reference. When references are accumulating and keywords are present, specify additional *fields* at most on the first citation of a particular reference; they apply to all further citations thereof.

opening-text and *closing-text* are *roff* input used to bracket the label, overriding the **bracket−label** command. Leading and trailing spaces are significant. If either of these is non-empty, the corresponding argument to the **bracket−label** command is not used; alter this behavior with the **[** and **]** *flags.*

flags is a list of non-alphanumeric characters each of which modifies the treatment of the particular citation. AT&T *refer* ignores them since they are non-alphanumeric. They direct GNU *refer* as follows.

Use the label specified by the **short−label** command, if any. *refer* otherwise uses the normal label. Typically, a short label implements author-date citation styles consisting of a name, a year, and a disambiguating letter if necessary. "**#**" is meant to suggest such a (quasi-)numeric label.

[Precede *opening-text* with the first argument given to the **bracket−label** command.

] Follow *closing-text* with the second argument given to the **bracket−label** command.

An advantage of the **[** and **]** flags over use of *opening-text* and *closing-text* is that you can update the document's bracketing style in one place using the **bracket−label** command. Another is that sorting and merging of citations is not necessarily inhibited if the flags are used.

refer appends any label resulting from a citation to the *roff* input line preceding the **.[** token. If there is no such line, *refer* issues a warning diagnostic.

There is no special notation for citing multiple references in series. Use a sequence of citations, one for each reference, with nothing between them. *refer* attaches all of their labels to the line preceding the first. These labels may be sorted or merged. See the description of the **<>** label expression, and of the **sort−adjacent−labels** and **abbreviate−label−ranges** commands. A label is not merged if its citation has a non-empty *opening-text* or *closing-text*. However, the labels for two adjacent citations, the former using the **]** flag and without any *closing-text,* and the latter using the **[** flag and without any *opening-text,* may be sorted and merged even if the former's *opening-text* or the latter's *closing-text* is non-empty. (To prevent these operations, use the dummy character escape sequence **\&** as the former's *closing-text*.)

Commands

Commands are contained between lines starting with **.R1** and **.R2**. The **−R** option prevents recognition of these lines. When *refer* encounters a **.R1** line, it flushes any accumulated references. Neither **.R1** nor **.R2** lines, nor anything between them, is output.

Commands are separated by newlines or semicolons. A number sign (**#**) introduces a comment that extends to the end of the line, but does not conceal the newline. Each command is broken up into words. Words are separated by spaces or tabs. A word that begins with a (neutral) double quote (**"**) extends to the next double quote that is not followed by another double quote. If there is no such double quote, the word extends to the end of the line. Pairs of double quotes in a word beginning with a double quote collapse to one double quote. Neither a number sign nor a semicolon is recognized inside double quotes. A line can be continued by ending it with a backslash "\"; this works everywhere except after a number sign.

Each command *name* that is marked with * has an associated negative command **no−***name* that undoes the effect of *name*. For example, the **no−sort** command specifies that references should not be sorted. The negative commands take no arguments.

In the following description each argument must be a single word; *field* is used for a single upper or lower case letter naming a field; *fields* is used for a sequence of such letters; *m* and *n* are used for a non-negative numbers; *string* is used for an arbitrary string; *file* is used for the name of a file.

abbreviate* *fields string1 string2 string3 string4*

Abbreviate the first names of *fields*. An initial letter will be separated from another initial letter by *string1*, from the surname by *string2*, and from anything else (such as "von" or "de") by *string3*. These default to a period followed by a space. In a hyphenated first name, the initial of the first part of the name will be separated from the hyphen by *string4*; this defaults to a period. No attempt is made to handle any ambiguities that might result from abbreviation. Names are abbreviated before sorting and before label construction.

abbreviate−label−ranges* *string*

Three or more adjacent labels that refer to consecutive references will be abbreviated to a label consisting of the first label, followed by *string*, followed by the last label. This is mainly useful with numeric labels. If *string* is omitted, it defaults to "**−**".

accumulate*

Accumulate references instead of writing out each reference as it is encountered. Accumulated references will be written out whenever a reference of the form

```
.[
$LIST$
.]
```

is encountered, after all input files have been processed, and whenever a **.R1** line is recognized.

annotate* *field string*

field is an annotation; print it at the end of the reference as a paragraph preceded by the line

.*string*

If *string* is omitted, it will default to **AP**; if *field* is also omitted it will default to **X**. Only one field can be an annotation.

articles *string* ...

Each *string* is a definite or indefinite article, and should be ignored at the beginning of **T** fields when sorting. Initially, "a", "an", and "the" are recognized as articles.

bibliography *file* ...

Write out all the references contained in each bibliographic database *file*. This command should come last in an **.R1/.R2** block.

bracket−label *string1 string2 string3*

In the text, bracket each label with *string1* and *string2*. An occurrence of *string2* immediately followed by *string1* will be turned into *string3*. The default behavior is as follows.

```
bracket-label \*([. \*(.] ", "
```

capitalize *fields*

Convert *fields* to caps and small caps.

compatible*

Recognize **.R1** and **.R2** even when followed by a character other than space or newline.

database *file* ...

Search each bibliographic database *file*. For each *file*, if an index file.*i* created by *indxbib*(1) exists, then it will be searched instead; each index can cover multiple databases.

date−as−label* *string*

string is a label expression that specifies a string with which to replace the **D** field after constructing the label. See subsection "Label expressions" below for a description of label expressions. This command is useful if you do not want explicit labels in the reference list, but instead want to handle any necessary disambiguation by qualifying the date in some way. The label used in the text would typically be some combination of the author and date. In most cases you should also use the **no−label−in−reference** command. For example,

```
date-as-label D.+yD.y%a*D.-y
```

would attach a disambiguating letter to the year part of the **D** field in the reference.

default−database*

The default database should be searched. This is the default behavior, so the negative version of this command is more useful. *refer* determines whether the default database should be searched on the first occasion that it needs to do a search. Thus a **no−default−database** command must be given before then, in order to be effective.

discard* *fields*

When the reference is read, *fields* should be discarded; no string definitions for *fields* will be output. Initially, *fields* are **XYZ**.

et−al* *string m n*

Configure use of "**et al**" in the evaluation of @ expressions in label expressions. If *u* is the number of authors needed to make the author sequence unambiguous and the total number of authors is *t,* then the last $t - u$ authors will be replaced by *string* provided that $t - u$ is not less than *m* and *t* is not less than *n*. The default behavior is as follows.

```
et−al " et al" 2 3
```

Note the absence of a dot from the end of the abbreviation, which is arguably not correct. (*Et al*[.] is short for *et alli*, as *etc.* is short for *et cetera*.)

include *file*

Include *file* and interpret the contents as commands.

join−authors *string1 string2 string3*

Join multiple authors together with *string*s. When there are exactly two authors, they will be joined with *string1*. When there are more than two authors, all but the last two will be joined with *string2*, and the last two authors will be joined with *string3*. If *string3* is omitted, it will default to *string1*; if *string2* is also omitted it will also default to *string1*. For example,

```
join−authors " and " ", " ", and "
```

will restore the default method for joining authors.

label−in−reference*

When outputting the reference, define the string **[F** to be the reference's label. This is the default behavior, so the negative version of this command is more useful.

label−in−text*

For each reference output a label in the text. The label will be separated from the surrounding text as described in the **bracket−label** command. This is the default behavior, so the negative version of this command is more useful.

label *string*

string is a label expression describing how to label each reference.

separate−label−second−parts *string*

When merging two-part labels, separate the second part of the second label from the first label with *string*. See the description of the **<>** label expression.

move−punctuation*

In the text, move any punctuation at the end of line past the label. We recommend employing this command unless you are using superscripted numbers as labels.

reverse* *string*

Reverse the fields whose names are in *string*. An optional integer after a field name limits the number of such fields to the given count; no integer means no limit.

search−ignore* *fields*

While searching for keys in databases for which no index exists, ignore the contents of *fields*. Initially, fields **XYZ** are ignored.

search−truncate* *n*

Only require the first *n* characters of keys to be given. In effect when searching for a given key words in the database are truncated to the maximum of *n* and the length of the key. Initially, *n* is 6.

short−label* *string*

string is a label expression that specifies an alternative (usually shorter) style of label. This is used when the **#** flag is given in the citation. When using author-date style labels, the identity of the author or authors is sometimes clear from the context, and so it may be desirable to omit the author or authors from the label. The **short−label** command will typically be used to specify a label containing just a date and possibly a disambiguating letter.

sort* *string*

Sort references according to *string*. References will automatically be accumulated. *string* should be a list of field names, each followed by a number, indicating how many fields with the name should be used for sorting. "**+**" can be used to indicate that all the fields with the name should be used. Also **.** can be used to indicate the references should be sorted using the (tentative) label. (Subsection "Label expressions" below describes the concept of a tentative label.)

sort−adjacent−labels*

Sort labels that are adjacent in the text according to their position in the reference list. This command should usually be given if the **abbreviate−label−ranges** command has been given, or if the label expression contains a **<>** expression. This has no effect unless references are being accumulated.

Label expressions

Label expressions can be evaluated both normally and tentatively. The result of normal evaluation is used for output. The result of tentative evaluation, called the *tentative label*, is used to gather the information that normal evaluation needs to disambiguate the label. Label expressions specified by the **date−as−label** and **short−label** commands are not evaluated tentatively. Normal and tentative evaluation are the same for all types of expression other than **@**, *****, and **%** expressions. The description below applies to normal evaluation, except where otherwise specified.

field [*n*]

is the *n*th part of *field*. If *n* is omitted, it defaults to 1.

'*string***'** The characters in *string* literally.

@ All authors joined as specified by the **join−authors** command. The whole of each author's name is used. However, if the references are sorted by author (that is, the sort specification starts with "**A+**"), then authors' surnames will be used instead, provided that this does not introduce ambiguity, and also an initial subsequence of the authors may be used instead of all the authors, again provided that this does not introduce ambiguity. Given any two referenced works with *n* authors, the use of only the surname for the *n*th author of a reference is regarded as ambiguous if the other reference shares the first $n-1$ authors, the *n*th authors of each reference are not identical, but the *n*th authors' surnames *are* the same. A proper initial subsequence of the sequence of authors for some reference is considered to be ambiguous if there is a reference with some other sequence of authors which also has that subsequence as a proper initial subsequence. When an initial subsequence of authors is used, the remaining authors are replaced by the string specified by the **et−al** command; this command may also specify additional requirements that must be met before an initial subsequence can be used. **@** tentatively evaluates to a canonical representation of the authors, such that authors that compare equally for sorting purposes have the same representation.

%*n*
%a
%A
%i
%I The serial number of the reference formatted according to the character following the **%**. The serial number of a reference is 1 plus the number of earlier references with same tentative label as this reference. These expressions tentatively evaluate to an empty string.

*expr****** If there is another reference with the same tentative label as this reference, then *expr*, otherwise an empty string. It tentatively evaluates to an empty string.

*expr***+***n*
*expr***−***n* The first (**+**) or last (**−**) *n* upper or lower case letters or digits of *expr*. *roff* special characters (such as **\('a**) count as a single letter. Accent strings are retained but do not count toward the total.

*expr***.l** *expr* converted to lowercase.

*expr***.u** *expr* converted to uppercase.

*expr***.c** *expr* converted to caps and small caps.

*expr***.r** *expr* reversed so that the surname is first.

*expr***.a** *expr* with first names abbreviated. Fields specified in the **abbreviate** command are abbreviated before any labels are evaluated. Thus **.a** is useful only when you want a field to be abbreviated in a label but not in a reference.

*expr***.y** The year part of *expr*.

*expr***.+y**
The part of *expr* before the year, or the whole of *expr* if it does not contain a year.

*expr***.−y**
The part of *expr* after the year, or an empty string if *expr* does not contain a year.

*expr***.n** The surname part of *expr*.

expr1~*expr2*
expr1 except that if the last character of *expr1* is **−** then it will be replaced by *expr2*.

expr1 expr2
The catenation of *expr1* and *expr2*.

expr1|*expr2*
If *expr1* is non-empty then *expr1* otherwise *expr2*.

expr1&*expr2*
If *expr1* is non-empty then *expr2* otherwise an empty string.

expr1?*expr2*:*expr3*
If *expr1* is non-empty then *expr2* otherwise *expr3*.

<*expr*> The label is in two parts, which are separated by *expr*. Two adjacent two-part labels which have the same first part will be merged by appending the second part of the second label onto the first label separated by the string specified in the **separate−label−second−parts** command (initially, a comma followed by a space); the resulting label will also be a two-part label with the same first part as before merging, and so additional labels can be merged into it. It is permissible for the first part to be empty; this may be desirable for expressions used in the **short−label** command.

(*expr*) The same as *expr*. Used for grouping.

The above expressions are listed in order of precedence (highest first); **&** and **|** have the same precedence.

Macro interface

Each reference starts with a call to the macro **]−**. The string **[F** will be defined to be the label for this reference, unless the **no−label−in−reference** command has been given. There then follows a series of string definitions, one for each field: string **[***X* corresponds to field *X*. The register **[P** is set to 1 if the **P** field contains a range of pages. The **[T**, **[A** and **[O** registers are set to 1 according as the **T**, **A** and **O** fields end with any of **.?!** (an end-of-sentence character). The **[E** register will be set to 1 if the **[E** string contains more than one name. The reference is followed by a call to the **][** macro. The first argument to this macro gives a number representing the type of the reference. If a reference contains a **J** field, it will be classified as type 1, otherwise if it contains a **B** field, it will be type 3, otherwise if it contains a **G** or **R** field it will be type 4, otherwise if it contains an **I** field it will be type 2, otherwise it will be type 0. The second argument is a symbolic name for the type: **other**, **journal−article**, **book**, **article−in−book**, or **tech−report**. Groups of references that have been accumulated or are produced by the **bibliography** command are preceded by a call to the **]<** macro and followed by a call to the **]>** macro.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−R Don't recognize lines beginning with **.R1/.R2**.

Other options are equivalent to *refer* commands.

−a *n*	**reverse A***n*
−b	**no−label−in−text; no−label−in−reference**
−B	See below.
−c *fields*	**capitalize** *fields*
−C	**compatible**
−e	**accumulate**
−f *n*	**label %***n*
−i *fields*	**search−ignore** *fields*
−k	**label L~%a**
−k *field*	**label** *field***~%a**
−l	**label A.nD.y%a**
−l *m*	**label A.n+***m***D.y%a**
−l ,*n*	**label A.nD.y−***n***%a**
−l *m***,***n*	**label A.n+***m***D.y−***n***%a**
−n	**no−default−database**
−p *db-file*	**database** *db-file*
−P	**move−punctuation**
−s *spec*	**sort** *spec*
−S	**label "(A.n|Q) ', ' (D.y|D)"; bracket-label " (") "; "**
−t *n*	**search−truncate** *n*

The **B** option has command equivalents with the addition that the file names specified on the command line are processed as if they were arguments to the **bibliography** command instead of in the normal way.

−B	**annotate X AP; no−label−in−reference**
−B *field***.***macro*	**annotate** *field macro***; no−label−in−reference**

Exit status

refer exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Environment

REFER Assign this variable a file name to override the default database.

Files

/usr/dict/papers/Ind
Default database.

file.*i* Index files.

/usr/share/groff/1.24.2/tmac/refer.tmac
defines macros and strings facilitating integration with macro packages that wish to support *refer*.

refer uses temporary files. See the *groff* (1) man page for details of where such files are created.

Bugs

In label expressions, **<>** expressions are ignored inside **.***char* expressions.

Examples

We can illustrate the operation of *refer* with a sample bibliographic database containing one entry and a simple *roff* document to cite that entry.

```
$ cat > my-db-file
%A Daniel P.\& Friedman
%A Matthias Felleisen
%C Cambridge, Massachusetts
%D 1996
%I The MIT Press
%T The Little Schemer, Fourth Edition
$ refer -p my-db-file
Read the book
.[
friedman
.]
on your summer vacation.
<Control+D>
.lf 1 -
Read the book\*([.1\*(.]
.ds [F 1
.]-
.ds [A Daniel P. Friedman and Matthias Felleisen
.ds [C Cambridge, Massachusetts
.ds [D 1996
.ds [I The MIT Press
.ds [T The Little Schemer, Fourth Edition
.nr [T 0
.nr [A 0
.][ 2 book
.lf 5 -
on your summer vacation.
```

The foregoing shows us that *refer* (a) produces a label "1"; (b) brackets that label with interpolations of the "**[.**" and "**.]**" strings; (c) calls a macro "**]–**"; (d) defines strings and registers containing the label and bibliographic data for the reference; (e) calls a macro "**][**"; and (f) uses the **lf** request to restore the line numbers of the input. As noted in subsection "Macro interface" above, it is the document's responsibility to employ and format the information usefully. Let us see how we might turn *groff_ms*(7) to this task.

```
$ REFER=my-db-file groff -R -ms
.LP
Read the book
.[
friedman
.]
on your summer vacation.
Commentary is available.\*{*\*}
.FS \*{*\*}
Space reserved for penetrating insight.
.FE
```

ms's automatic footnote numbering mechanism is not aware of *refer*'s label numbering, so we have manually specified a (superscripted) symbolic footnote for our non-bibliographic aside.

See also

"Refer — A Bibliography System", by Bill Tuthill, 1983, Computing Services, University of California, Berkeley.

"Some Applications of Inverted Indexes on the Unix System", by M. E. Lesk, 1978, AT&T Bell Laboratories Computing Science Technical Report No. 69.

indxbib(1), *lookbib*(1), *lkbib*(1)

Name

soelim – recursively interpolate source requests in *roff* or other text files

Synopsis

soelim [**−Crt**] [**−I** *dir*] [*input-file* . . .]

soelim −−help

soelim −v
soelim −−version

Description

GNU *soelim* is a preprocessor for the *groff*(7) document formatting system. *soelim* eliminates source requests in *roff*(7) and other text files; that is, it replaces lines of the form "**.so** *included-file*" within each text *input-file* with the contents of *included-file* recursively, flattening a tree of documents. By default, it writes *roff* **lf** requests as well to record the name and line number of each *input-file* and *included-file*, so that any diagnostics produced by later processing can be accurately traced to the original input. Options allow this information to be suppressed (**−r**) or supplied in TEX comments instead (**−t**). In the absence of *input-file* arguments, *soelim* reads the standard input stream. The program writes to the standard output stream.

soelim reads the *included-file* argument as GNU *troff* does. It ignores spaces immediately after "so"; to embed a sequence of one or more leading spaces in the argument, prefix the sequence with a neutral double quote ("). Non-leading spaces are interpreted literally. A backslash followed by a space ("\ ") also encodes a space, for compatibility with earlier versions of GNU *soelim*. If the included file name requires a backslash, use **** or **\e** to embed it. Any other escape sequence in *included-file*, including " **\[rs]**", prevents *soelim* from replacing the source request. AT&T and descendant versions of *soelim* have no means of embedding spaces in file names; they replace the first space encountered with a newline and stop interpreting the request.

The dot must be at the beginning of a line and must be followed by "**so**" without intervening spaces or tabs for *soelim* to handle it. This convention allows source requests to be "protected" from processing by *soelim*, for instance as part of macro definitions or "**if**" requests.

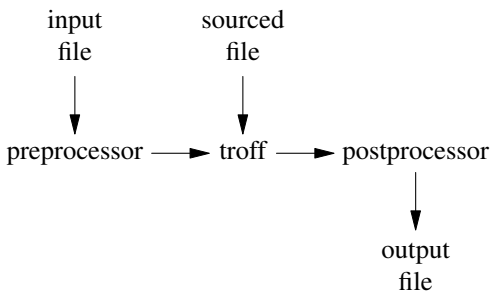
There must also be at least one space between "**so**" and its *included-file* argument. The **−C** option overrides this requirement.

The foregoing is the limit of *soelim*'s understanding of the *roff* language; it does not, for example, replace the input line

```
.if 1 .so otherfile
```

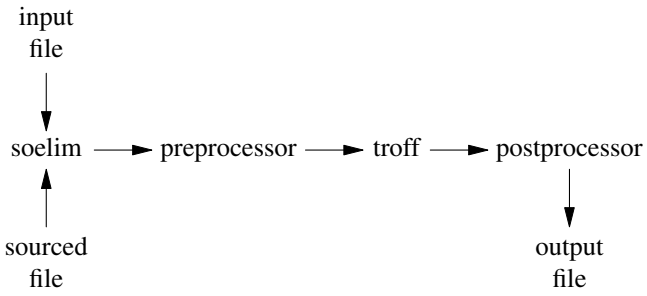
with the contents of *otherfile*. With its **−r** option, therefore, *soelim* can be used to process text files in general.

soelim was designed to handle situations where the target of a *roff* source request requires a preprocessor such as *eqn*(1), *pic*(1), *refer*(1), or *tbl*(1). The usual processing sequence of *groff*(1) is as follows.



That is, files sourced with "**so**" are normally read *only* by the formatter, *troff*(1). *soelim* is *not* required for *troff* to source files.

If a file to be sourced should also be preprocessed, it must already be read *before* the input passes through the preprocessor. *soelim*, normally invoked via *groff*'s **−s** option, handles this.



Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−C Recognize an input line starting with **.so** even if a character other than a space or newline follows.

−I *dir* Search the directory *dir* for *input-* and *included-files*. If specified more than once, each *dir* is searched in the given order. To search the current working directory before others, add "**−I .**" at the desired place; it is otherwise searched last.

−r Write files "raw"; do not add **lf** requests.

−t Emit TEX comment lines starting with "**%**" indicating the current file and line number, rather than **lf** requests for the same purpose.

If both **−r** and **−t** are given, the last one specified controls.

Exit status

soelim exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

See also

groff (1)

Name

tbl – prepare tables for *groff* documents

Synopsis

tbl [**−C**] [*file* . . .]

tbl −−help

tbl −v
tbl −−version

Description

The GNU implementation of *tbl* is part of the *groff* (1) document formatting system. *tbl* is a *troff* (1) preprocessor that translates descriptions of tables embedded in *roff* (7) input into the language understood by *troff*. It copies each *file*'s contents to the standard output stream, transforming each *table region* between lines starting with **.TS** and **.TE** into instructions to the GNU *troff* formatter. While GNU *tbl*'s input syntax is highly compatible with AT&T *tbl*, the output GNU *tbl* produces cannot be processed by AT&T *troff*; GNU *troff* (or a *troff* implementing any GNU extensions employed) must be used. Normally, *tbl* is not executed directly by the user, but invoked by specifying the **−t** option to *groff* (1). If no *file* operands are present, or if *file* is "**−**", *tbl* reads the standard input stream.

Overview

tbl expects to find table descriptions between input lines that begin with **.TS** (table start) and **.TE** (table end). Each such table region encloses one or more table descriptions. Within a table region, table descriptions beyond the first must each be preceded by an input line beginning with **.T&**. This mechanism does not start a new table region; all table descriptions are treated as part of their **.TS/.TE** enclosure, even if they are boxed or have column headings that repeat on subsequent pages (see below).

(Experienced *roff* users should observe that *tbl* is not a *roff* language interpreter: the default control character must be used, and no spaces or tabs are permitted between the control character and the macro name. These *tbl* input tokens remain as-is in the output, where they become ordinary macro calls. Macro packages often define **TS**, **T&**, and **TE** macros to handle issues of table placement on the page. *tbl* produces *troff* requests to define these macros as empty if their definitions do not exist when the formatter encounters a table region.)

Each table region may begin with *region options,* and must contain one or more *table definitions;* each table definition contains a *format specification* followed by one or more input lines (rows) of *entries.* These entries comprise the *table data.*

Region options

The line immediately following the **.TS** token may specify region options, keywords that influence the interpretation or rendering of the region as a whole or all table entries within it indiscriminately. Separate multiple region options with commas, spaces, or tabs. Those that require a parenthesized argument permit spaces and tabs between the option's name and the opening parenthesis. Options accumulate and cannot be unset within a region once declared; if an option that takes a parameter is repeated, the last occurrence controls. If present, the set of region options must be terminated with a semicolon (**;**).

Any of the **allbox**, **box**, **doublebox**, **frame**, and **doubleframe** region options makes a table "boxed" for the purpose of later discussion.

allbox Enclose each table entry in a box; implies **box**.

box Enclose the entire table region in a box. GNU *tbl* recognizes **frame** as a synonym.

center Center the table region with respect to the line length, reducing the indentation if necessary (with a warning at formatting time) to make the table region fit; the default is to left-align it. GNU *tbl* recognizes **centre** as a synonym.

decimalpoint(*c***)** Recognize character *c* as the decimal separator in columns using the **N** (numeric) classifier (see subsection "Column classifiers" below). This is a GNU extension.

delim(*xy*) Recognize characters *x* and *y* as start and end delimiters, respectively, for *eqn*(1) input, and ignore input between them. *x* and *y* need not be distinct.

doublebox Enclose the entire table region in a double box; implies **box**. GNU *tbl* recognizes **doubleframe** as a synonym.

expand Spread the table horizontally to fill the available space (line length minus indentation) by increasing column separation. Ordinarily, a table is made only as wide as necessary to accommodate the widths of its entries and its column separations (whether specified or default). When **expand** applies to a table that exceeds the available horizontal space, *tbl* reduces column separation as far as necessary (even to zero). *tbl* produces *troff* input that issues a diagnostic if such compression occurs. The column modifier **x** (see below) overrides this option.

linesize(*n*) Draw lines or rules (e.g., from **box**) with a thickness of *n* points. The default is the current type size when the region begins. This option has no effect on terminal devices.

nokeep Don't use *roff* diversions to manage page breaks. Normally, *tbl* employs them to avoid breaking a page within a table row. This usage can sometimes interact badly with macro packages' own use of diversions—when footnotes, for example, are employed. This is a GNU extension.

nospaces Ignore leading and trailing spaces in table entries. This is a GNU extension.

nowarn Suppress diagnostic messages produced at document formatting time when the line length and indentation or page length is inadequate to contain a table row. This is a GNU extension.

tab(*c*) Use the character *c* instead of a tab to separate entries in a row of table data.

Table format specification

The table format specification is mandatory: it determines the number of columns in the table and directs how the entries within it are to be typeset. The format specification is a series of column *descriptors.* Each descriptor encodes a *classifier* followed by zero or more *modifiers.* Classifiers are letters (recognized case-insensitively) or punctuation symbols; modifiers consist of or begin with letters or numerals. Spaces, tabs, newlines, and commas separate descriptors. Newlines and commas apply the descriptors following them to a subsequent row of the table (enabling column headings to be centered or emboldened while the table entries for the data are not, for instance). We term the resulting group of column descriptors a *row definition.* Within a row definition, separation between column descriptors by spaces or tabs is often optional; only some modifiers, described below, make separation necessary.

The classifier selects from one of several arrangements. Some determine the positioning of table entries within a rectangular cell: centered, left-aligned, numeric (aligned to a configurable decimal separator), and so on. Others perform special operations like drawing lines or spanning entries from adjacent cells in the table. Except for "|", any classifier can be followed by one or more *modifiers;* some of these accept an argument, which in GNU *tbl* can be parenthesized. Modifiers select fonts, set the type size, and perform other tasks described below.

The format specification can occupy multiple input lines, but must conclude with a dot "**.**" followed by a newline. Each row definition applies in turn to one row of the table. *tbl* applies the last row definition to rows of table data in excess of the row definitions.

For clarity in this document's examples, we write classifiers in uppercase and modifiers in lowercase. Thus, "**CbCb,LR.**" defines two rows of two columns. The first row's entries are centered and boldfaced; the second and any further rows' first and second columns are left- and right-aligned, respectively. Any rows of entries appended to the table data reuse the row definition "**LR**".

The row definition with the most column descriptors determines the number of columns in the table; any row definition with fewer, GNU *tbl* implicitly extends on the right-hand side with **L** descriptors as many times as necessary to make the table rectangular.

Column classifiers

The **L**, **R**, and **C** classifiers are the easiest to understand and use.

A, **a** Center longest entry in this column, left-align remaining entries in the column with respect to the centered entry, then indent all entries by one en. Such "alphabetic" entries (hence the name of the classifier) can be used in the same column as **L**-classified entries, as in "**LL,AR.**". The **A** entries are often termed "sub-columns" due to their indentation.

C, **c** Center entry within the column.

L, **l** Left-align entry within the column.

N, **n** Numerically align entry in the column. *tbl* aligns columns of decimal numbers vertically at the units place. If multiple decimal separators are adjacent to a digit, it uses the rightmost one for vertical alignment. If there is no decimal separator, *tbl* uses the rightmost digit for vertical alignment; if no digits are present, *tbl* centers the entry within the column. The *roff* dummy character **\&** in an entry marks the glyph preceding it (if any) as the units place; if multiple instances occur in the data, *tbl* uses the leftmost for alignment.

If **N**-classified entries share a column with **L** or **R** entries, *tbl* centers the widest **N** entry with respect to the widest **L** or **R** entry, preserving the alignment of **N** entries with respect to each other.

Decimal separators in *eqn* equations within **N**-classified columns can conflict with *tbl*'s use of them for alignment. Specify the **delim** region option to make *tbl* ignore the data within *eqn* delimiters.

R, **r** Right-align entry within the column.

S, **s** Span previous entry on the left into this column.

^ Span entry in the same column from the previous row into this row.

, **−** Replace table entry with a horizontal rule. *tbl* expects an empty table entry to correspond to this classifier; if data are found there, it issues a diagnostic message. If the entire row definition consists of these classifiers (only one is necessary), it is treated as a "****" occupying a row of table entries, and no corresponding data are expected.

= Replace table entry with a double horizontal rule. *tbl* expects an empty table entry to correspond to this classifier; if data are found there, it issues a diagnostic message. If the entire row definition consists of these classifiers (only one is necessary), it is treated as a "**=**" occupying a row of table entries, and no corresponding data are expected.

| Place a vertical rule (line) on the corresponding row of the table (if two of these are adjacent, a double vertical rule). This classifier does not contribute to the column count and no table entries correspond to it. A **|** to the left of the first column descriptor or to the right of the last one produces a vertical rule at the edge of the table; these are redundant (and ignored) in boxed tables.

To change the table format within a *tbl* region, use the **.T&** token at the start of a line. Follow it with a format specification and table data, but *not* region options. The quantity of columns in a format thus introduced cannot increase relative to the previous format; in that case, you must end the table region and start another. If that will not serve because the region uses box options or the columns align in an undesirable manner, you must design the initial table format specification to include the maximum quantity of columns required, and use the **S** horizontal spanning classifier where necessary to achieve the desired columnar alignment.

Spanning horizontally in the first column or vertically on the first row is an error. *tbl* does not support non-rectangular span areas.

Column modifiers

Any number of modifiers can follow a column classifier. Modifier arguments, where accepted, are case-sensitive. If you apply a given modifier to a classifier more than once, or apply conflicting modifiers, only the last occurrence has effect. The modifier **x** is mutually exclusive with **e** and **w**, but **e** is not mutually exclusive with **w**; if these are used in combination, **x** unsets both **e** and **w**, while either **e** or **w** overrides **x**.

b, **B** Typeset entry in boldface, abbreviating **f(B)**.

d, **D** Align a vertically spanned table entry to the bottom ("down"), instead of the center, of its range. This is a GNU extension.

e, **E** Equalize the widths of columns with this modifier to that of the widest. This modifier sets the default line length used in a text block.

f, **F** Select the typeface for the table entry. A font or style name (one or two characters not starting with a digit), font mounting position (a single digit), or a name or mounting position of any length in parentheses, must follow. The last form is a GNU extension. (The parameter corresponds to that accepted by the *troff* **ft** request.) A one-character argument not in parentheses must end the row description or be separated by one or more spaces or tabs from what follows.

i, **I** Typeset entry in an oblique or italic face, abbreviating **f(I)**.

m, **M** Call a *groff* macro before typesetting a text block (see subsection "Text blocks" below). This is a GNU extension. A macro name of one or two characters, or a name of any length in parentheses, must follow. A one-character macro name not in parentheses must be separated by one or more spaces or tabs from what follows. The named macro must be defined before the table region containing this column modifier is encountered. The macro should contain only simple *groff* requests to change text formatting, like adjustment or hyphenation. The macro is called *after* the column modifiers **b**, **f**, **i**, **p**, and **v** take effect; it can thus override other column modifiers.

p, **P** Set the type size. An integer *n* with an optional leading sign must follow. If unsigned, the type size is set to *n* points. Otherwise, the type size is incremented or decremented per the sign by *n* points. The use of a signed multi-digit number is a GNU extension. (The parameter corresponds to that accepted by the *troff* **ps** request.) If a type size modifier is followed by a column separation modifier (see below), they must be separated by at least one space or tab.

t, **T** Align a vertically spanned table entry to the top, instead of the center, of its range.

u, **U** Move the column up one half-line, "staggering" the rows. This is a Documenter's Workbench (DWB) 1.0 and Research Tenth Edition Unix extension.

v, **V** Set the vertical spacing of a text block. An integer *n* with an optional leading sign must follow. If unsigned, the vertical spacing is set to *n* points. Otherwise, the vertical spacing is incremented or decremented per the sign by *n* points. The use of a signed multi-digit number is a GNU extension. (This parameter corresponds to that accepted by the *troff* **vs** request.) If a vertical spacing modifier is followed by a column separation modifier (see below), they must be separated by at least one space or tab.

w, **W** Set the column's minimum width. A number, either a unitless integer, or a *roff* horizontal measurement in parentheses, must follow. Parentheses are required if the width is to be followed immediately by an explicit column separation (alternatively, follow the width with one or more spaces or tabs). If no unit is specified, ens are assumed. This modifier sets the default line length used in a text block.

x, **X** Expand the column. After computing the column widths, distribute any remaining line length evenly over all columns bearing this modifier. This is a Documenter's Workbench (DWB) 3.3 extension. Applying the **x** modifier to more than one column is a GNU extension. This modifier sets the default line length used in a text block.

z, **Z** Ignore the table entries corresponding to this column for width calculation purposes; that is, compute the column's width using only the information in its descriptor. This is a Documenter's Workbench (DWB) 1.0 and Research Tenth Edition Unix extension.

n A numeric suffix on a column descriptor sets the separation distance (in ens) from the succeeding column; the default separation is **3n**. This separation is proportionally multiplied if the **expand** region option is in effect; in the case of tables wider than the output line length, this separation might be zero. A negative separation cannot be specified. A separation amount after the last column in a row is nonsensical and provokes a diagnostic from *tbl*.

Table data

Place table data on lines after the format specification. Each text line corresponds to a table row, except that a backslash at the end of a line of table data continues an entry on the next input line. (Text blocks, discussed below, also spread table entries across multiple input lines.) Table entries within a row are separated in the input by a tab character by default; see the **tab** region option above. Excess entries in a row of table data (those that have no corresponding column descriptor, not even an implicit one arising from rectangularization of the table), *tbl* discards with a diagnostic message. *tbl* passes *roff* control lines through unaltered to its output. If you wish to visibly mark an empty table entry in the document source, populate it with the **\&** *roff* dummy character. The table data are interrupted by a line consisting of the **.T&** input token, and conclude with the line **.TE**.

Ordinarily, a table entry is typeset rigidly. It is not filled, broken, hyphenated, adjusted, or populated with supplemental inter-sentence space. *tbl* instructs the formatter to measure each table entry as it occurs in the input, updating the width required by its corresponding column. If the **z** modifier applies to the entry, this measurement is ignored; if **w** applies and its argument is larger than this width, that argument is used instead. In contrast to conventional *roff* input (within a paragraph, say), changes to text formatting, such as font selection or vertical spacing, do not persist between entries.

tbl interprets several forms of table entry specially.

- If a table row contains only an underscore or equals sign (**_** or **=**), *tbl* draws a a single or double horizontal rule (line), respectively, across the table at that point.
- If a table entry contains only **_** or **=** on an otherwise populated row, *tbl* populates its cell with a single or double horizontal rule, respectively, that joins its neighbors.
- If a table entry contains only **_** or **\=** on an otherwise populated row, *tbl* populates its cell with a single or double horizontal rule, respectively, that does *not* (quite) join its neighbors.
- If a table entry contains only **\R***x*, where *x* is any *roff* ordinary, special, or indexed character, *tbl* populates its cell with enough repetitions of the glyph corresponding to *x* to fill the column without joining its neighbors.
- On any row but the first, a table entry of **\^** causes the entry above it to span down into the current one.

On occasion, these special tokens may be required as literal table data. To use either **_** or **=** literally and alone in an entry, prefix or suffix it with the *roff* dummy character **\&**. To express **_**, **\=**, or **\R**, use a *roff* escape sequence to interpolate the backslash (**\e** or **\[rs]**). A reliable way to emplace the **\^** glyph sequence within a table entry is to use a pair of *groff* special character escape sequences (**\[rs]\[ha]**).

Rows of table entries can be interleaved with *groff* control lines; these do not count as table data. On such lines the default control character (**.**) must be used (and not changed); the no-break control character is not recognized. To start the first table entry in a row with a dot, precede it with the *roff* dummy character **\&**.

Text blocks

An ordinary table entry's contents can make a column, and therefore the table, too wide; the table then exceeds the line length of the page, and becomes ugly or is exposed to truncation by the output device. When a table entry requires more conventional typesetting, breaking across more than one output line (and thereby increasing the height of its row), it can be placed within a *text block*.

tbl interprets a table entry of "**T{**" at the end of an input line specially, as a token starting a text block. Similarly, an entry "**T}**" at the start of an input line ends a text block. Text block tokens can share an input line with other table data (preceding **T{** and following **T}**). Input lines between these tokens are formatted in a diversion by *troff*. Text blocks cannot be nested. Multiple text blocks can occur in a table row.

Text blocks are formatted as was the text prior to the table, modified by applicable column descriptors. Specifically, the classifiers **A**, **C**, **L**, **N**, **R**, and **S** determine a text block's *alignment* within its cell, but not its *adjustment*. Add **na** or **ad** requests to the beginning of a text block to alter its adjustment distinctly from other text in the document. As with other table entries, when a text block ends, any alterations to formatting parameters are discarded. They do not affect subsequent table entries, not even other text blocks.

If **w** or **x** modifiers are not specified for *all* columns of a text block's span, the default length of the text block (more precisely, the line length used to process the text block's diversion) is computed as $L \times C/(N+1)$, where L is the current line length, C the number of columns spanned by the text block, and N the number of columns in the table. If necessary, you can also control a text block's width by including an **ll** (line length) request in it prior to any text to be formatted. Because *tbl* uses a diversion to format the text block, its height and width are subsequently available in the *troff* registers **dn** and **dl**, respectively.

roff interface

The register **TW** stores the width of the table region in basic units; it can't be used within the region itself, but is defined before the **.TE** token is output so that a *troff* macro named **TE** can make use of it. "**T.**" is a Boolean-valued register indicating whether the bottom of the table is being processed. A **#T** register is used internally. Avoid using these names for any other purpose.

tbl also defines a macro **T#** to produce the bottom and side lines of a boxed table. While *tbl* itself arranges for the output to include a call of this macro at the end of such a table, it can also be used by macro packages to create boxes for multi-page tables by calling it from a page footer macro that is itself called by a trap planted near the bottom of the page. See section "Limitations" below for more on multi-page tables.

GNU *tbl* internally employs register, string, macro, and diversion names beginning with the numeral **3**. A document to be preprocessed with GNU *tbl* should not use any such identifiers.

Interaction with *eqn*

Process a document with *tbl* before *eqn*(1). (*groff*(1) automatically arranges preprocessors in the correct order.) Don't call the **EQ** and **EN** macros within tables; instead, set up delimiters in your *eqn* input and use the **delim** region option so that *tbl* will recognize them.

GNU *tbl* enhancements

In addition to extensions noted above, GNU *tbl* removes constraints endured by users of AT&T *tbl*.

- Region options can be specified in any lettercase.
- There is no limit on the number of columns in a table, regardless of their classification, nor any limit on the number of text blocks.
- GNU *tbl* considers all table rows when computing column widths, not just those occurring in the first 200 input lines of a region. Similarly, it recognizes table continuation tokens (**.T&**) outside a region's first 200 input lines.
- Numeric and alphabetic entries may appear in the same column.
- Numeric and alphabetic entries may span horizontally.

Using GNU *tbl* within macros

You can embed a table region inside a macro definition. However, since *tbl* writes its own macro definitions at the beginning of each table region, it is necessary to call end macros instead of ending macro definitions with "**..**". Additionally, the escape character must be disabled.

Not all *tbl* features can be exercised from such macros because *tbl* is a *roff* preprocessor: it sees the input earlier than *troff* does. For example, vertically aligning decimal separators fails if the numbers containing them occur as macro or string parameters; the alignment is performed by *tbl* itself, which sees only **\$1**, **\$2**, and so on, and therefore can't recognize a decimal separator that appears only later when *troff* interpolates a macro or string definition.

Using *tbl* macros within conditional input (that is, contingent upon an **if**, **ie**, **el**, or **while** request) can result in misleading line numbers in subsequent diagnostics. *tbl* unconditionally injects its output into the source document, but the conditional branch containing it may not be taken, and if it is not, the **lf** requests that *tbl* injects to restore the source line number cannot take effect. Consider copying the input line counter register "**c.**" and restoring its value at a convenient location after applicable arithmetic.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−C Enable AT&T compatibility mode: recognize **.TS** and **.TE** even when followed by a character other than space or newline, and interpret the copy-mode leader escape sequence **\a** as a leader character.

Exit status

tbl exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Limitations

Within a *tbl* region (except in text blocks or on *roff* control lines), avoid escape sequences that read the rest of the input line, as **\"** and **\!** do.

Multi-page tables, if boxed and/or if you want their column headings repeated after page breaks, require support at the time the document is formatted. A convention for such support has arisen in macro packages such as *ms*, *mm*, and *me*. To use it, follow the **.TS** token with a space and then "**H**"; this will be interpreted by the formatter as a **TS** macro call with an **H** argument. Then, within the table data, call the **TH** macro; this informs the macro package where the headings end. If your table has no such heading rows, or you do not desire their repetition, call **TH** immediately after the table format specification. If a multi-page table is boxed or has repeating column headings, do not enclose it with keep/release macros, or divert it in any other way. Further, the **bp** request will not cause a page break in a "**TS H**" table. Define a macro to wrap **bp**: invoke it normally if there is no current diversion. Otherwise, pass the macro call to the enclosing diversion using the transparent line escape sequence **\!**; this will "bubble up" the page break to the output device. See section "Examples" below for a demonstration.

grotty(1) does not support double horizontal rules; it uses single rules instead. It also ignores half-line motions, so the **u** column modifier has no effect. On terminal devices ("*nroff* mode"), horizontal rules and box borders occupy a full vee of space; **doublebox** doubles that for borders. Tables using these features thus require more vertical space in *nroff* mode than in *troff* mode: write **ne** requests accordingly. Vertical rules between columns are drawn in the space between columns in *nroff* mode; using double vertical rules and/or reducing the column separation below the default can make them ugly or overstrike them with table data.

A text block within a table must be able to fit on one page.

Using **\a** to put leaders in table entries does not work in GNU *tbl*, except in compatibility mode. This is correct behavior: **\a** is an *uninterpreted* leader. You can still use the *roff* leader character (Control+A) or define a string to use **\a** as it was designed: to be interpreted only in copy mode.

```
.ds a \a
.TS
box center tab(;);
Lw(2i)0 L.
Population\*a;6,327,119
.TE
```

Population.....6,327,119

A leading or trailing "**|**" in a format specification, as in "**|LCR|.**", produces an en space between the rules and the content of adjacent columns. If such space is undesired (the rule should abut the content), you can introduce "dummy" columns with zero separation and empty corresponding table entries before and/or after.

```
.TS
center tab(#);
R0|L C R0|L.
_
#levulose#glucose#dextrose#
_
.TE
```

These dummy columns have zero width and are therefore invisible; unfortunately they usually don't work as intended on terminal devices.

levulose	glucose	dextrose

Examples

It can be easier to acquire the language of *tbl* through examples than formal description, especially at first.

```
.TS
box center tab(#);
Cb Cb
L  L.
Ability#Application
Strength#crushes a tomato
Dexterity#dodges a thrown tomato
Constitution#eats a month-old tomato without becoming ill
Intelligence#knows that a tomato is a fruit
Wisdom#chooses \f[I]not\f[] to put tomato in a fruit salad
Charisma#sells tomato-based fruit salads to hypercarnivores
.TE
```

Ability	Application
Strength	crushes a tomato
Dexterity	dodges a thrown tomato
Constitution	eats a month-old tomato without becoming ill
Intelligence	knows that a tomato is a fruit
Wisdom	chooses *not* to put tomato in a fruit salad
Charisma	sells tomato-based fruit salads to hypercarnivores

The **A** and **N** column classifiers can be easier to grasp in visual rendering than in description.

```
.TS
center tab(;);
CbS,LN,AN.
Daily energy intake (in MJ)
Macronutrients
.\" assume 3 significant figures of precision
Carbohydrates;4.5
Fats;2.25
Protein;3
.T&
LN,AN.
Mineral
Pu-239;14.6
_
.T&
LN.
Total;\[ti]24.4
.TE
```

Daily energy intake (in MJ)	
Macronutrients	
Carbohydrates	4.5
Fats	2.25
Protein	3
Mineral	
Pu-239	14.6
Total	~24.4

Next, we'll lightly adapt a compact presentation of spanning, vertical alignment, and zero-width column modifiers from the *mandoc* reference for its *tbl* interpreter. It rewards close study.

```
.TS
box center tab(:);
Lz  S | Rt
Ld| Cb| ^
^ | Rz  S.
left:r
l:center:
:right
.TE
```

left		r
	center	
l		right

Row staggering is not visually achievable on terminal devices, but a table using it can remain comprehensible nonetheless.

```
.TS
center tab(|);
Cf(BI) Cf(BI) Cf(B), C C Cu.
n|n\f[B]\[tmu]\f[]n|difference
1|1
2|4|3
3|9|5
4|16|7
5|25|9
6|36|11
.TE
```

n	***n*×*n***	**difference**
1	1	
2	4	3
3	9	5
4	16	7
5	25	9
6	36	11

Some *tbl* features cannot be illustrated in the limited environment of a portable man page.

We can define a macro outside of a *tbl* region that we can call from within it to cause a page break inside a multi-page boxed table. You can choose a different name; be sure to change both occurrences of "BP".

```
.de BP
.  ie '\\n(.z'' .bp \\$1
.  el \!.BP \\$1
..
```

See also

"Tbl—A Program to Format Tables", by M. E. Lesk, 1976 (revised 16 January 1979), AT&T Bell Laboratories Computing Science Technical Report No. 49.

The spanning example above was taken from *mandoc*'s man page for its *tbl* implementation.

groff (1), *troff* (1)

Name

tfmtodit – adapt TeX Font Metrics files for use with *groff* and *grodvi*

Synopsis

tfmtodit [**–s**] [**–g** *gf-file*] [**–k** *skew-char*] *tfm-file map-file font-description*

tfmtodit ––help

tfmtodit –v
tfmtodit ––version

Description

tfmtodit creates a font description file for use with *groff*(1)'s **dvi** output device. *tfm-file* is the name of the TEX font metric file for the font. *map-file* assigns *groff* ordinary or special character identifiers to glyph indices in the font; it should consist of a sequence of lines of the form

i c1 . . . cn

where *i* is a position of the glyph in the font in decimal, and *c1* through *cn* are glyph identifiers in the form used by *groff* font descriptions. If a glyph has no *groff* names but exists in *tfm-file,* it is put in the *groff* font description file as an unnamed glyph. Output is written in *groff_font*(5) format to *font-description,* a file named for the intended *groff* font name.

If the font is "special", meaning that *groff* should search it whenever a glyph is not found in the current font, use the **–s** option and name *font-description* in the **fonts** directive in the output device's *DESC* file.

To do a good job of math typesetting, *groff* requires font metric information not present in *tfm-file.* This is because TEX has separate math italic fonts, whereas *groff* uses normal italic fonts for math. The additional information required by *groff* is given by the two arguments to the **math_fit** macro in the Metafont programs for the Computer Modern fonts. In a text font (a font for which **math_fit** is false), Metafont normally ignores these two arguments. Metafont can be made to put this information into the GF ("generic font") files it produces by loading the following definition after **cmbase** when creating *cm.base*.

```
def ignore_math_fit(expr left_adjustment,right_adjustment) =
    special "adjustment";
    numspecial left_adjustment*16/designsize;
    numspecial right_adjustment*16/designsize;
    enddef;
```

For the EC font family, load the following definition after **exbase**; consider patching *exbase.mf* locally.

```
def ignore_math_fit(expr left_adjustment,right_adjustment) =
    ori_special "adjustment";
    ori_numspecial left_adjustment*16/designsize;
    ori_numspecial right_adjustment*16/designsize;
    enddef;
```

The only difference from the previous example is the "ori_" prefix to "special" and "numspecial". The GF file created using this modified *cm.base* or *exbase.mf* should be specified with the **–g** option, which should *not* be given for a font for which **math_fit** is true.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–g *gf-file*
: Use the *gf-file* produced by Metafont containing "**special**" and "**numspecial**" commands to obtain additional font metric information.

–k *skew-char*
: The skew character of this font is at position *skew-char. skew-char* should be an integer; it may be given in decimal, with a leading 0 in octal, or with a leading 0x in hexadecimal. Any kerns whose second component is *skew-char* are ignored.

−s Add the **special** directive to the font description file.

Exit status

tfmtodit exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Files

/usr/share/groff/1.24.2/font/devdvi/DESC
describes the **dvi** output device.

*/usr/share/groff/1.24.2/font/devdvi/*F
describes the font known as *F* on device **dvi**.

/usr/share/groff/1.24.2/font/devdvi/generate/ec.map
/usr/share/groff/1.24.2/font/devdvi/generate/msam.map
/usr/share/groff/1.24.2/font/devdvi/generate/msbm.map
/usr/share/groff/1.24.2/font/devdvi/generate/tc.map
/usr/share/groff/1.24.2/font/devdvi/generate/texb.map
/usr/share/groff/1.24.2/font/devdvi/generate/texex.map
/usr/share/groff/1.24.2/font/devdvi/generate/texi.map
/usr/share/groff/1.24.2/font/devdvi/generate/texitt.map
/usr/share/groff/1.24.2/font/devdvi/generate/texmi.map
/usr/share/groff/1.24.2/font/devdvi/generate/texr.map
/usr/share/groff/1.24.2/font/devdvi/generate/texsy.map
/usr/share/groff/1.24.2/font/devdvi/generate/textex.map
/usr/share/groff/1.24.2/font/devdvi/generate/textt.map
map glyph indices in TEX fonts to *groff* ordinary and special character identifiers. *ec.map* is used for **TREC**, **TIEC**, **TBEC**, **TBIEC**, **HREC**, **HIEC**, **HBEC**, **HBIEC**, **CWEC**, and **CWIEC**; *msam.map* for **SA**; *msbm.map* for **SB**; *tc.map* for **TRTC**, **TITC**, **TBTC**, **TBITC**, **HRTC**, **HITC**, **HBTC**, **HBITC**, **CWTC**, and **CWITC**; *texb.map* for **TB**, **HR**, **HI**, **HB**, and **HBI**; *texex.map* for **EX**; *texi.map* for **TI** and **TBI**; *texitt.map* for **CWI**; *texmi.map* for **MI**; *texr.map* for **TR**; *texsy.map* for **S**; *textex.map* for **SC**; and *textt.map* for **CW**.

See also

groff(1), *grodvi*(1), *groff_font*(5)

Name

troff – GNU *roff* typesetter and document formatter

Synopsis

troff [**−abcCEiRSUz**] [**−d** *ctext*] [**−d** *string*=*text*] [**−f** *font-family*] [**−F** *font-directory*] [**−I** *inclusion-directory*] [**−m** *macro-package*] [**−M** *macro-directory*] [**−n** *page-number*] [**−o** *page-list*] [**−r** *cnumeric-expression*] [**−r** *register*=*numeric-expression*] [**−T** *output-device*] [**−w** *warning-category*] [**−W** *warning-category*] [*file* . . .]

troff −−help

troff −v
troff −−version

Description

GNU *troff* transforms *groff* (7) language input into the device-independent page description language detailed in *groff_out*(5); *troff* is thus the heart of the GNU *roff* document formatting system. If no *file* operands are present, or if *file* is "**−**", *troff* reads the standard input stream.

GNU *troff* is functionally compatible with the AT&T *troff* typesetter and features numerous extensions. Many people prefer to use the *groff* (1) command, a front end which also runs preprocessors and output drivers in the appropriate order and with appropriate options.

Options

−h and **−−help** display a usage message, while **−v** and **−−version** show version information; all exit afterward.

−a Generate a plain text approximation of the typeset output. The read-only register **.A** is set to 1. This option produces a sort of abstract preview of the formatted output.

- Page breaks are marked by a phrase in angle brackets; for example, "<beginning of page>".
- Lines are broken where they would be in formatted output.
- Vertical motion, apart from that implied by a break, is not represented.
- A horizontal motion of any size is represented as one space. Adjacent horizontal motions are not combined. Supplemental inter-sentence space (configured by the second argument to the **.ss** request) is not represented.
- A special character is rendered as its identifier between angle brackets; for example, a hyphen appears as "<hy>".

The above description should not be considered a specification; the details of **−a** output are subject to change.

−b Write a backtrace reporting the state of *troff*'s input parser to the standard error stream with each diagnostic message. The line numbers given in the backtrace might not always be correct, because *troff*'s idea of line numbers can be confused by requests that append to macros.

−c Disable multi-color output and "**color**" request's ability to enable it.

−C Enable AT&T *troff* compatibility mode; implies **−c**. See *groff_diff* (7).

−d *ctext*
−d *string*=*text*
Define *roff* string *c* or *string* as *text*. *c* must be a one-character identifier; *string* can be of arbitrary length. Such assignments happen before any macro file is loaded, including the startup file. Due to *getopt_long*(3) limitations, *c* cannot be, and *string* cannot contain, an equals sign, even though that is a valid character in a *roff* identifier.

−E Inhibit *troff* error messages; implies **−Ww**. This option does *not* suppress messages sent to the standard error stream by documents or macro packages using **tm** or related requests.

−f *fam* Use *fam* as the default font family.

−F *dir* Search in directory *dir* for the selected output device's directory of device and font description files. See the description of *GROFF_FONT_PATH* in section "Environment" below for the default search locations and ordering.

−i Read the standard input stream after all named input files have been processed.

−I *dir* Search the directory *dir* for files (those named on the command line; in **psbb**, **so**, and **soquiet** requests; and in "**\X'ps: import'**", " **\X'ps: file'**", and " **\X'pdf: pdfpic'**" device extension escape sequences). **−I** may be specified more than once; each *dir* is searched in the given order. To search the current working directory before others, add "**−I .**" at the desired place; it is otherwise searched last. **−I** works similarly to, and is named for, the "include" option of Unix C compilers.

−m *mac* Search for the macro package mac.*tmac* and read it prior to any input. If not found, *tmac*.mac is attempted. See the description of *GROFF_TMAC_PATH* in section "Environment" below for the default search locations and ordering.

−M *dir* Search directory *dir* for macro files. See the description of *GROFF_TMAC_PATH* in section "Environment" below for the default search locations and ordering.

−n *num* Begin numbering pages at *num.* The default is **1**.

−o *list* Output only pages in *list,* which is a comma-separated list of inclusive page ranges; *n* means page *n, m−n* means every page between *m* and *n*, **−***n* means every page up to *n*, and *n***−** means every page from *n* on. *troff* stops processing and exits after formatting the last page enumerated in *list.*

−r *cnumeric-expression*
−r *register*=*numeric-expression*
Define *roff* register *c* or *register* as *numeric-expression.* *c* must be a one-character identifier; *register* can be of arbitrary length. Such assignments happen before any macro file is loaded, including the startup file. Due to *getopt_long*(3) limitations, *c* cannot be, and *register* cannot contain, an equals sign, even though that is a valid character in a *roff* identifier.

−R Prevent loading of *troffrc* and *troffrc−end*; this option can be useful for troubleshooting.

−S Enable safer mode and ignore any subsequent **−U** option.

−T *dev* Prepare output for device *dev.* The default is **ps**; see *groff* (1).

−U Operate in *unsafe mode,* enabling the **cf**, **open**, **opena**, **pi**, **pso**, and **sy** requests, which are disabled by default because they allow an untrusted input document to run arbitrary commands, put arbitrary content into *troff* output, or write to arbitrary file names. (GNU *troff* does not, however, accept newlines (line feeds) in file names supplied as arguments to requests.) This option also adds the current directory to the macro package search path; see the **−m** and **−M** options above.

−w *cat*
−W *cat* Enable and inhibit, respectively, warnings in category *cat.* See section "Warnings" below.

−z Suppress formatted output.

Warnings

GNU *troff* divides its warning diagnostics into named, numbered categories. The **−w** and **−W** options use the associated names. A power of two characterizes each category; the **warn** request and the **.warn** register respectively set and report the sum of enabled category codes. Warnings of each category are produced under the following circumstances.

Bit	Code	Category	Bit	Code	Category
0	*1*	**char**	10	*1024*	**reg**
1	*2*	*unused*	11	*2048*	**tab**
2	*4*	**break**	12	*4096*	*unused*
3	*8*	**delim**	13	*8192*	**missing**
4	*16*	*unused*	14	*16384*	**input**
5	*32*	**scale**	15	*32768*	**escape**
6	*64*	**range**	16	*65536*	**space**
7	*128*	**syntax**	17	*131072*	**font**
8	*256*	**di**	18	*262144*	**ig**
9	*512*	**mac**	19	*524288*	**color**
			20	*1048576*	**file**

break 4 A filled output line could not be broken such that its length was less than or equal to, or adjusted such that its length was exactly equal to, the output line length "**\n[.l]**". GNU *troff* reports the amount of overset or underset in the scaling unit configured by the **warnscale** request in **troff** mode, and in ens ('**n**'; character cells) in **nroff** mode. This category is enabled by default.

char 1 No user-defined character of the requested name or index exists and no mounted font defines a glyph for it, or input could not be encoded for device-independent output. This category is enabled by default.

color 524288 An undefined color name was selected, an attempt was made to define a color using an unrecognized color space, an invalid channel value in a color definition was encountered, or an attempt was made to redefine a default color.

delim 8 The selected delimiter character was ambiguous because it is also meaningful when beginning a numeric expression, or the closing delimiter in an escape sequence was missing or mismatched.

A future *groff* release may reject ambiguous delimiters. In compatibility mode, ambiguous delimiters are accepted without warning.

di 256 A **di**, **da**, **box**, or **boxa** request was invoked without an argument when there was no current diversion.

escape 32768 An unsupported escape sequence was encountered.

file 1048576 An attempt was made to read a file that does not exist. or a stream remained open at formatter exit. This category is enabled by default.

font 131072 A non-existent font was selected. This category is enabled by default.

ig 262144 An invalid escape sequence occurred in input ignored using the **ig** request. This warning category diagnoses a condition that is an error when it occurs in non-ignored input.

input 16384 An invalid character occurred on the input stream.

mac 512 An undefined string, macro, or diversion was used. When such an object is dereferenced, an empty one of that name is automatically created. So, unless it is later deleted, *troff* issues at most one warning for each.

troff also uses this category to warn of an attempt to move an unplanted trap macro. In such cases, the unplanted macro is *not* dereferenced, so it is not created if it does not exist.

missing 8192 A request was invoked with a mandatory argument absent.

range	64	A numeric expression was out of range for its context.
reg	1024	An undefined register was used. When an undefined register is dereferenced, the formatter automatically defines it with a value of 0. So, unless it is later deleted, GNU *troff* issues at most one warning for each.
scale	32	A scaling unit inappropriate to its context was used in a numeric expression.
space	65536	A space was missing between a request or macro and its argument. This warning is produced when an undefined name longer than two characters is encountered and the first two characters of the name constitute a defined name. No request is invoked, no macro called, and an empty macro is not defined. This category is enabled by default. It never occurs in compatibility mode.
syntax	128	A self-contradictory hyphenation mode or character flags were requested; an empty or incomplete numeric expression was encountered; an operand to a numeric operator was missing; an attempt was made to format characters or spaces on an input line after an output line continuation escape sequence; a recognized but inapposite escape sequence or unprintable character code was used in a device extension command; an attempt was made to define a recursive, empty, or nonsensical character class; or a *groff* extension escape sequence or conditional expression operator was used while in compatibility mode.
tab	2048	A tab character appeared in a parameterized escape sequence, in an unquoted macro argument, or where a request expected a numeric expression argument.

Two warning names group other warning categories for convenience.

all All warning categories except **di**, **mac**, and **reg**. This shorthand is intended to produce all warnings that are useful with macro packages and documents written for AT&T *troff* and its descendants, which have less fastidious diagnostics than GNU *troff*.

w All warning categories. Authors of documents and macro packages targeting *groff* are encouraged to use this setting.

Exit status

troff exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters a fatal error during operation, or is directed to abort by the input.

Environment

GROFF_FONT_PATH and *GROFF_TMAC_PATH* each accept a search path of directories; that is, a list of directory names separated by the system's path component separator character. On Unix systems, this character is a colon (:); on Windows systems, it is a semicolon (;).

GROFF_FONT_PATH

A list of directories in which to seek the selected output device's directory of device and font description files. *troff* will scan directories given as arguments to any specified **−F** options before these, then in a site-specific directory (*/etc/groff/site−font*), a standard location (*/usr/share/groff/1.24.2/font*), and a compatibility directory (*/usr/lib/font*) after them.

GROFF_TMAC_PATH

A list of directories in which to search for macro files. *troff* will scan directories given as arguments to any specified **−M** options before these, then the current directory (only if in unsafe mode), the user's home directory, a site-specific directory (*/etc/groff/site−tmac*), and a standard location (*/usr/share/groff/1.24.2/tmac*) after them.

GROFF_TYPESETTER

Set the default output device. If empty or not set, **ps** is used. The **−T** option overrides *GROFF_TYPESETTER*.

SOURCE_DATE_EPOCH
: A timestamp (expressed as seconds since the Unix epoch) to use as the output creation timestamp in place of the current time. The time is converted to human-readable form using *gmtime*(3) and *asctime*(3) when the formatter starts up and stored in registers usable by documents and macro packages.

TZ
: The time zone to use when converting the current time to human-readable form; see *tzset*(3). If *SOURCE_DATE_EPOCH* is used, it is always converted to human-readable form using UTC.

Files

/usr/share/groff/1.24.2/tmac/troffrc
: is an initialization macro file loaded before any macro packages specified with **−m** options.

/usr/share/groff/1.24.2/tmac/troffrc−end
: is an initialization macro file loaded after all macro packages specified with **−m** options.

*/usr/share/groff/1.24.2/tmac/*name.*tmac*
: are macro files distributed with *groff*.

*/usr/share/groff/1.24.2/font/dev*name*/DESC*
: describes the output device *name*.

*/usr/share/groff/1.24.2/font/dev*name/F
: describes the font *F* of device *name.*

troffrc and *troffrc−end* are sought neither in the current nor the home directory by default for security reasons, even if the **−U** option is specified. Use the **−M** command-line option or the *GROFF_TMAC_PATH* environment variable to add these directories to the search path if necessary.

Authors

The GNU version of *troff* was originally written by James Clark; he also wrote the original version of this document, which was updated by Werner Lemberg, Bernd Warken, and G. Branden Robinson.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

groff(1)
: offers an overview of the GNU *roff* system and describes its front end executable.

groff(7)
: details the *groff* language, including a short but complete reference of all predefined requests, registers, and escape sequences.

groff_char(7)
: explains the syntax of *groff* special character escape sequences, and lists all special characters predefined by the language.

groff_diff(7)
: enumerates the differences between AT&T device-independent *troff* and *groff*.

groff_font(5)
: covers the format of *groff* device and font description files.

groff_out(5)
: describes the format of *troff*'s output.

groff_tmac(5)
: includes information about macro files that ship with *groff*.

roff(7)
: supplies background on *roff* systems in general, including pointers to further related documentation.

Name

xtotroff – convert X font metrics into *groff* font metrics

Synopsis

xtotroff [**–d** *destination-directory*] [**–r** *resolution*] [**–s** *type-size*] *font-map*

xtotroff ––help

xtotroff –v
xtotroff ––version

Description

xtotroff uses *font-map* to create *groff* (1) font description files from X11 fonts. Each line in *font-map* pairs a *groff* font name with an X font name as an X Logical Font Description (XLFD) pattern. The pair's members are separated by spaces and/or tabs. For example, an input *font-map* file consisting of the line

```
TB -adobe-times-bold-r-normal--*-*-*-*-p-*-iso8859-1
```

maps the XLFD on the right to the *groff* font name **TB**, conventionally "Times bold".

xtotroff opens a connection to the running X server to query its font catalog, and aborts if it cannot. If necessary, the wildcards in the XLFD patterns are populated with the arguments to the **–r** and **–s** options. If a font name is still ambiguous, *xtotroff* aborts. For each successful mapping, *xtotroff* creates a *groff* font description file in the current working directory (or that specified by the **–d** option) named for each *groff* font, and reports the mapping to the standard output stream.

Options

––help displays a usage message, while **–v** and **––version** show version information; all exit afterward.

–d *destination-directory*
: Write font descriptions to *destination-directory* rather than the current working directory.

–r *resolution*
: Set the resolution for all font patterns in *font-map*. The value is used for both the horizontal and vertical motion quanta. If not specified, a resolution of 75dpi is assumed.

–s *type-size*
: Set the type size in points for all font patterns in *font-map*. If not specified, a size of 10 points is assumed.

Exit status

xtotroff exits with status **0** on successful operation, status **2** if the program cannot interpret its command-line arguments, and status **1** if it encounters an error during operation.

Files

/usr/share/groff/1.24.2/font/FontMap–X11
: is the font mapping file used to produce the pre-generated font description files, supplied with *groff*, of X11 core fonts corresponding to the 13 base Type 1 fonts for PostScript level 1.

Bugs

The only supported font encodings are "iso8859–1" and "adobe–fontspecific".

See also

"X Logical Font Description Conventions", by Jim Flowers and Stephen Gildea.

X(7), *groff* (1), *gxditview*(1), *troff* (1), *groff_font*(5)

Name

groff_font – GNU *roff* device and font description files

Description

The *groff* font and output device description formats are slight extensions of those used by AT&T device-independent *troff*. In distinction to the AT&T implementation, *groff* lacks a binary format; all files are text files. (Plan 9 *troff* has also abandoned the binary format.) The device and font description files for a device *name* are stored in a *dev*name directory. The device description file is called *DESC*, and, for each font supported by the device, a font description file is called *f*, where *f* is usually an abbreviation of a font's name and/or style. For example, the **ps** (PostScript) device has *groff* font description files for Times roman (**TR**) and Zapf Chancery Medium italic (**ZCMI**), among many others, while the **utf8** device (for terminals) has font descriptions for the roman, italic, bold, and bold-italic styles (**R**, **I**, **B**, and **BI**, respectively).

Device and font description files are read by the formatter, *troff*, and by output drivers. The programs typically delegate these files' processing to an internal library, *libgroff*, ensuring their consistent interpretation.

DESC file format

The *DESC* file contains a series of directives; each begins a line. Their order is not important, with two exceptions: (1) the **res** directive must precede any **papersize** directive; and (2) the **charset** directive must come last (if at all). If a directive name is repeated, later entries in the file override previous ones (except that the paper dimensions are computed based on the **res** directive last seen when **papersize** is encountered). Spaces and/or tabs separate words and are ignored at line boundaries. Comments start with the "**#**" character and extend to the end of a line. Empty lines are ignored.

family *fam*
: The default font family is *fam*.

fonts *n F1 ... Fn*
: Fonts *F1*, ..., *Fn* are mounted at font positions $m+1$, ..., $m+n$ where m is the number of **styles** (see below). This directive may extend over more than one line. A font name of **0** causes no font to be mounted at the corresponding position.

hor *n*
: The horizontal motion quantum is *n* basic units. Horizontal measurements round to multiples of *n*.

image_generator *program*
: Use *program* to generate PNG images from PostScript input. Under GNU/Linux, this is usually *gs*(1), but under other systems (notably Cygwin) it might be set to another name. The *grohtml*(1) driver uses this directive.

paperlength *n*
: The vertical dimension of the output medium is *n* basic units (deprecated: use **papersize** instead).

papersize *format-or-dimension-pair-or-file-name ...*
: The dimensions of the output medium are as according to the argument, which is either a standard paper format, a pair of dimensions, or the name of a plain text file containing either of the foregoing. Recognized paper formats are the ISO and DIN formats **A0**–**A7**, **B0**–**B7**, **C0**–**C7**, and **D0**–**D7**; the U.S. formats **letter**, **legal**, **tabloid**, **ledger**, **statement**, and **executive**; and the envelope formats **com10**, **monarch**, and **DL**. Matching is performed without regard for lettercase.

  Alternatively, the argument can be a custom paper format *length***,***width* (with no spaces before or after the comma). Both *length* and *width* must have a unit appended; valid units are "**i**" for inches, "**c**" for centimeters, "**p**" for points, and "**P**" for picas. Example: "**12c,235p**". An argument that starts with a digit is always treated as a custom paper format.

  Finally, the argument can be a file name (e.g., */etc/papersize*); if the file can be opened, the first line is read and a match attempted against each other form. No comment syntax is supported.

  More than one argument can be specified; each is scanned in turn and the first valid paper specification used.

paperwidth *n*
The horizontal dimension of the output medium is *n* basic units (deprecated: use **papersize** instead).

pass_filenames
Direct *troff* to emit the name of the source file being processed. This is achieved with the intermediate output command "**x F**", which *grohtml* interprets.

postpro *program*
Use *program* as the postprocessor.

prepro *program*
Use *program* as a preprocessor. The **html** and **xhtml** output devices use this directive.

print *program*
Use *program* as the print spooler. If omitted, *groff*'s **–l** and **–L** options are ignored.

res *n* The device resolution is *n* basic units per inch.

sizes *s1 . . . sn* **0**
The device has fonts at *s1*, . . ., *sn* scaled points (see below). The list of sizes must be terminated by a **0**. Each *si* can also be a range of sizes *m–n*. The list can extend over more than one line.

sizescale *n*
A typographical point is subdivided into *n* scaled points. The default is **1**.

styles *S1 . . . Sm*
The first *m* font mounting positions are associated with styles *S1*, . . ., *Sm*.

tcommand
The postprocessor can handle the **t** and **u** intermediate output commands.

unicode
The output device supports the complete Unicode repertoire. This directive is useful only for devices which produce character entities instead of glyphs.

If **unicode** is present, no **charset** section is required in the font description files since the Unicode handling built into *groff* is used. However, if there are entries in a font description file's **charset** section, they either override the default mappings for those particular characters or add new mappings (normally for composite characters).

The **utf8**, **html**, and **xhtml** output devices use this directive.

unitwidth *n*
Arbitrary basis with respect to which font metrics are proportionally scaled when rendering glyphs at a type size of one point.

unscaled_charwidths
Make the font handling module always return unscaled glyph widths. The *grohtml* driver uses this directive.

use_charnames_in_special
troff should encode special characters in arguments to device extension commands. The *grohtml* driver uses this directive.

vert *n* The vertical motion quantum is *n* basic units. Vertical measurements round to multiples of *n*.

charset
This directive and the rest of the file are ignored. It is recognized for compatibility with other *troff* implementations. In GNU *troff*, character set repertoire is described on a per-font basis.

troff recognizes but ignores the directives **spare1**, **spare2**, and **biggestfont**.

The **res**, **unitwidth**, **fonts**, and **sizes** lines are mandatory. Directives not listed above are ignored by *troff* but may be used by postprocessors to obtain further information about the device.

Font description file format

On typesetting output devices, each font is typically available at multiple sizes. While paper measurements in the device description file are in absolute units, measurements applicable to fonts must be proportional to the type size. The font's *unit width* establishes a numerical basis that permits all of its metrics to be expressed as integers if rendered at one point. When the formatter configures a type size, it scales the metrics linearly relative to that basis. The unit width has no inherent relationship to the device resolution, and the same division procedure applies to all font metrics. Observe that whatever unit might one select for the unit width, the division operation implied by scaling cancels it out, leaving a dimensionless quantity.

For instance, *groff*'s **lbp** device uses a **unitwidth** directive with an argument of 800. Its Times roman font **TR** has a **spacewidth** of 833; this is also the width of its comma, period, centered period, and mathematical asterisk, while its "**M**" has a width of 2,963. Thus, an "**M**" on the **lbp** device is 2,963 ÷ 800 times the unit width, or approximately 3.7. At a type size of 10 points, a Times roman "**M**" is therefore 37 units wide.

A font description file has two sections. The first is a sequence of directives, and is parsed similarly to the *DESC* file described above. Except for the directive names that begin the second section, their ordering is immaterial. Later directives of the same name override earlier ones, spaces and tabs are handled in the same way, and the same comment syntax is supported. Empty lines are ignored throughout.

name *F*
: The name of the font is *F*. " **DESC**" is an invalid font name. Simple integers are valid, but their use is discouraged. (*groff* requests and escape sequences interpret non-negative integers as mounting positions instead. Further, a font named "**0**" cannot be automatically mounted by the **fonts** directive of a *DESC* file.)

spacewidth *n*
: The width of an unadjusted inter-word space is *n*, relative to the device's unit width.

The directives above must appear in the first section; those below are optional.

slant *n*
: The font's glyphs have a slant of *n* degrees; a positive *n* slants in the direction of text flow.

ligatures *lig1* ... *lign* [**0**]
: Glyphs *lig1*, ..., *lign* are ligatures; possible ligatures are **ff**, **fi**, **fl**, **ffi**, and **ffl**. For compatibility with other *troff* implementations, the list of ligatures may be terminated with a **0**. The list of ligatures must not extend over more than one line.

special
: The font is *special*: when the document attempts to format a glyph that is not present in the formatter's currently selected font, the glyph is sought in any mounted fonts that bear this property. Often, such fonts are *unstyled*, having no heavy (bold) or slanted (italic or oblique) variants.

Other directives in this section are ignored by GNU *troff*, but may be used by postprocessors to obtain further information about the font.

The second section contains one to three subsections, which can appear in any order, and any of which starts the second section. Each starts with a directive on a line by itself. A **charset** subsection is mandatory unless the associated *DESC* file contains the **unicode** directive. Another subsection, **kernpairs**, is optional.

The directive **charset** starts the character set subsection. (On typesetters, this directive is misnamed since it starts a list of glyphs, not characters.) It precedes a series of glyph descriptions, one per line. Each such glyph description comprises a set of fields separated by spaces or tabs and organized as follows.

name metrics type index [*entity-name*] [**--** *comment*]

name identifies the glyph: a printable character *c* corresponds to the *troff* ordinary character *c*, and a multi-character sequence not beginning with ****, corresponds to the GNU *troff* special character escape sequence "**\[***name***]**". A name consisting of three minus signs, "**---**", indicates that the glyph is unnamed: such glyphs can be accessed only by the **\N** escape sequence in *troff*. A special character named " **---**" can still be defined using **.char** and similar requests. The *name* "**\-**" defines the minus sign glyph. Finally, *name* can be the

horizontal motion escape sequences, **\|** and **\^** ("thin" and "hair" spaces, respectively), in which case only the width metric described below is applied; a font can thus customize the widths of these spaces.

The form of the *metrics* field is as follows (on one line; it may be broken here for readability).

width[**,**[*height*[**,**[*depth*[**,**[*italic-correction*[**,**[*left-italic-correction*[**,**[*subscript-correction*]]]]]]]]]

Spaces, tabs, and newlines are prohibited between these *subfields,* which are expressed as decimal integers. The unit of measure is that established by the **unitwidth** directive and scaled to the type size. Unspecified subfields default to 0. Since there is no associated binary format, these values are not required to fit into the C language data type *char* as they are in AT&T device-independent *troff*.

The *width* subfield gives the width of the glyph. The *height* subfield gives the height of the glyph (upward is positive); if a glyph does not extend above the baseline, give it a zero height, not a negative height. The *depth* subfield gives the depth of the glyph—that is, the distance below the baseline to which the glyph extends (downward is positive); if a glyph does not extend below the baseline, give it a zero depth, not a negative depth. Italic corrections apply when upright and slanted (italic or oblique) styles are typeset adjacently. The *italic-correction* is the amount of space to add after a slanted glyph to be followed immediately by an upright glyph. The *left-italic-correction* is the amount of space to add before a slanted glyph to be preceded immediately by an upright glyph. The *subscript-correction* is the amount of space to add after a slanted glyph to be followed by a subscript; it should be less than the italic correction.

For fonts used with typesetters, the *type* field gives a featural description of the glyph: it is a bit mask recording whether the glyph is an ascender, descender, both, or neither. When a **\w** escape sequence is interpolated, these values are bitwise or-ed together for each glyph and stored in the **ct** register. In font descriptions for terminals, all glyphs might have a type of zero, regardless of their appearance.

0 means the glyph lies entirely between the baseline and a horizontal line at the "x-height" of the font, as with "a", "c", and "x";

1 means the glyph descends below the baseline, like "p";

2 means the glyph ascends above the font's x-height, like "A" or "b"); and

3 means the glyph is both an ascender and a descender—this is true of parentheses in some fonts.

The *index* field is an integer that uniquely identifies a glyph within the font; any integer is accepted as input, (that is, any integer parsable by the C standard library's *strtol*(3) function) but no practical font employs all possible values. An *index* is limited to the range of the system's C language data type *int*. In a *troff* document, use the indexed character escape sequence **\N** to specify a glyph by index.

The *entity-name* field defines an identifier for the glyph that the postprocessor uses to print the *troff* glyph *name*. This field is optional; it was introduced so that the *grohtml* output driver could encode its character set. For example, the glyph **\[Po]** is represented by "**£**" in HTML 4.0. For efficiency, these data are now compiled directly into *grohtml*. *grops* uses the field to build sub-encoding arrays for PostScript fonts containing more than 256 glyphs.

Anything on the line after the *entity-name* field or "**--**" is ignored. When *afmtodit* generates font description files for *gropdf*(1) and *grops*(1), it writes the UTF-16 code for the character to the comment field.

A line in the **charset** section can also have the form

name **"**

identifying *name* as another name for the glyph mentioned in the preceding line. Such aliases can be chained.

A **charset-range** subsection works like the **charset** directive except that the glyph descriptions use a *name* of the form **u***AAAA***..u***FFFF*, where *AAAA* and *FFFF* are hexadecimal digit sequences; the specified metrics then apply identically to all glyphs in the designated range.

The directive **kernpairs** starts a list of kerning adjustments to be made to adjacent glyph pairs from this font. It contains a sequence of lines formatted as follows.

g1 g2 n

The foregoing means that when glyph *g1* is typeset immediately before *g2*, the space between them should be

increased by *n*. The unit of measure is that established by the **unitwidth** directive and scaled to the type size. Most kerning pairs should have a negative value for *n*.

Files

*/usr/share/groff/1.24.2/font/dev*name*/DESC*
: describes the output device *name*.

*/usr/share/groff/1.24.2/font/dev*name/F
: describes the font known as *F* on device *name*.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

"Troff User's Manual" by Joseph F. Ossanna, 1976 (revised by Brian W. Kernighan, 1992), AT&T Bell Laboratories Computing Science Technical Report No. 54, widely called simply "CSTR #54", documents the language, device and font description file formats, and device-independent page description language referred to collectively in *groff* documentation as "AT&T *troff*".

"A Typesetter-independent TROFF" by Brian W. Kernighan, 1982, AT&T Bell Laboratories Computing Science Technical Report No. 97, provides additional insights into the device and font description file formats and device-independent page description language.

groff(1), subsection "Utilities", lists programs available for describing fonts in a variety of formats such that *groff* output drivers can use them.

troff(1) documents the default device and font description file search path.

groff_out(5), *addftinfo*(1)

Name

groff_out – GNU *roff* device-independent page description language

Description

The fundamental operation of the GNU *troff* formatter, *troff*(1), is the translation of the *groff*(7) input language into a series of instructions concerned primarily with placing glyphs or geometric objects at specific positions on a rectangular page. In the following discussion, the term *command* refers to this device-independent output language, never to the language intended for use by document authors. Device-independent output commands comprise several categories: glyph output; font, color, and text size selection; motion of the drawing position; page advancement; drawing of geometric primitives; and device extension commands, a catch-all for other operations. The last includes directives to start and stop output, identify the intended output device, and embed URL hyperlinks in supported output formats.

Background

As *groff*(1) is a wrapper program around GNU *troff* and automatically calls an output driver, users seldom encounter this format under normal circumstances. *groff* offers the option **–Z** to inhibit postprocessing such that GNU *troff*'s output is sent to the standard output stream just as it is when running GNU *troff* directly.

The purpose of device-independent output is to facilitate the development of postprocessors by providing a common programming interface for all devices. It is a distinct, and much simpler, language from that of the formatter, *troff*. The device-independent output can be thought of as a "page description language".

In the following discussion, the term *troff output* describes what is output by GNU *troff*, while *page description* denotes the language accepted by the parser that interprets this output for the output drivers. This parser handles whitespace more flexibly than AT&T *troff*'s implementation, recognizes a GNU extension to the language, and supports a legacy compressed encoding of a subset of commands for compatibility; otherwise, the formats are the same. (The parser for device-independent output can be found in the *groff* sources at *src/libs/libdriver/input.cpp*.)

When Brian Kernighan designed AT&T *troff*'s device-independent page description language circa 1980, he had to balance readability and maintainability against severe constraints on file size and transmission speed to the output device. A decade later, when James Clark wrote *groff*, these constraints were no longer as tight.

Syntax

roff's page description language is a sequence of *tokens:* single-letter commands or their arguments. Some commands accept a subcommand as a first argument, followed by one or more further arguments.

AT&T device-independent *troff* used whitespace minimally when producing output. GNU *troff*, in contrast, attempts to make its output more human-readable. The whitespace characters—tab, space, and newline—are always meaningful. They are never used to represent spacing in the document; that is done with horizontal (**h**, **H**) and vertical (**v**, **V**) positioning commands. Any sequence of space and/or tab characters is equivalent to a single space, separating commands from arguments and arguments from each other. Space is required only where omitting it would cause ambiguity. A line break separates commands. The comment character is a pound/hash sign (**#**), and marks the remainder of the line as a comment. A line comprising only whitespace after comment removal does nothing but separate input tokens.

For example, the relative horizontal motion command (**h**) and the command to write one glyph (**c**), each take a single argument; the former a signed integer, and the latter a printable ISO 646/"ASCII" character. A series of such commands could validly occur without spaces on an input line, but GNU *troff* follows each with a newline.

Some commands have a more complex syntax; the GNU *troff* extension command for writing glyph sequences (**t**) accepts a variable number of arguments. Those that draw geometric objects (**D**) or control the device (**x**) furthermore recognize subcommand arguments. Such commands thus must end with a newline. In GNU *troff,* the device extension (sub)command "**x X**" uniquely supports a line continuation syntax; a single input line contains any other.

Argument units

Some commands accept integer arguments that represent measurements, but the scaling units of the formatter's language are never used. Most commands assume a scaling unit of "**u**" (basic units), and others use "**s**" (scaled points). These are defined by the parameters specified in the device's *DESC* file; see *groff_font*(5) and, for more on scaling units, *groff*(7) and *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual. Color-related commands use dimensionless integers.

Note that single characters can have the eighth bit set, as can the names of fonts and special characters (this is, glyphs). The names of glyphs and fonts can be of arbitrary length. A glyph that is to be printed will always be in the current font.

A string argument is always terminated by the next whitespace character (space, tab, or newline); an embedded **#** character is regarded as part of the argument, not as the beginning of a comment command. An integer argument is already terminated by the next non-digit character, which then is regarded as the first character of the next argument or command.

Output structure

Device-independent *troff* output is organized into three parts: a header, a body, and a trailer.

The task of the header is to set general device parameters. GNU *troff* guarantees that its header consists of the following three lines:

x T *device*
x res *n h v*
x init

with the parameters *n*, *h*, and *v* set as outlined in subsection "Device Control Commands" below. The parser for the device-independent page description language format is able to interpret additional whitespace and comments as well even in the header.

The body contains the document's visible content. Once an output driver interprets "**x init**", it prepares to handle commands in general. Processing terminates when a "**x stop**" command is encountered; the last line of any GNU *troff* page description output always contains such a command.

Semantically, the body is page-oriented. The **p** command starts a new page. Positioning, writing, and drawing commands are performed within a page, so they cannot occur before the first **p** command. The output driver reckons absolute positioning (by the **H** and **V** commands) with respect to the current page's origin at the top left corner, and all other positioning relative to the drawing position on the page.

The trailer advances the drawing position to the bottom of the page and informs the device that the document (or "job") has ended.

Command reference

This section describes all page description output commands, both from AT&T *troff* as well as extension commands issued by GNU *troff*.

Comment command

#*anything*⟨line-break⟩

Apply comment annotation. Ignore any characters from the **#** character up to the next newline. Each comment can be preceded by arbitrary syntactical space, and every command can be terminated by a comment.

Simple commands

The commands in this subsection have a command code consisting of a single character, taking a fixed number of arguments. Most of them are commands for positioning and text writing. These commands are smart about whitespace. Optionally, *syntactical space* can be inserted before, after, and between the command letter and its arguments. All of these commands are stackable, i.e., they can be preceded by other simple commands or followed by arbitrary other commands on the same line. A separating *syntactical space* is necessary only when two integer arguments would clash or if the preceding argument ends with a string argument.

C *id*⟨white-space⟩
: Typeset the glyph of the special character *id*. Trailing syntactical space is necessary to allow special character names of arbitrary length. The drawing position is not advanced.

c *c*
: Typeset the glyph of the ordinary character *c*. The drawing position is not advanced.

f *n*
: Select the font mounted at position *n*. *n* cannot be negative.

H *n*
: Horizontally move the drawing position to *n* basic units from the left edge of the page. *n* cannot be negative.

h *n*
: Move the drawing position right *n* basic units. AT&T *troff* allowed negative *n;* GNU *troff* does not produce such values, but *groff*'s output driver library handles them.

m *scheme* [*component* . . .]
: Select the stroke color using the *component*s in the color space *scheme*. Each *component* is an integer between 0 and 65536. The quantity of components and their meanings vary with each *scheme*. This command is a *groff* extension.

  mc *cyan magenta yellow*
  : Use the CMY color scheme with components cyan, magenta, and yellow.

  md
  : Use the default color (no components; black in most cases).

  mg *gray*
  : Use a grayscale color scheme with a component ranging between 0 (black) and 65536 (white).

  mk *cyan magenta yellow black*
  : Use the CMYK color scheme with components cyan, magenta, yellow, and black.

  mr *red green blue*
  : Use the RGB color scheme with components red, green, and blue.

N *n*
: Typeset the glyph with index *n* in the current font. *n* is normally a non-negative integer. The drawing position is not advanced. The **html** and **xhtml** devices use this command with negative *n* to produce unbreakable space; the absolute value of *n* is taken and interpreted in basic units.

n *b a*
: Indicate a break. No action is performed; the command is present to make the output more easily parsed. The integers *b* and *a* describe the vertical space amounts before and after the break, respectively. GNU *troff* issues this command but *groff*'s output driver library ignores it. See **v** and **V**.

p *n*
: Begin a new page, setting its number to *n*. Each page is independent, even from those using the same number. The vertical drawing position is set to 0. All positioning, writing, and drawing commands are interpreted in the context of a page, so a **p** command must precede them.

s *n*
: Set type size to *n* scaled points (unit **z** in GNU *troff*). AT&T *troff* used unscaled points (**p**) instead; see section "Compatibility" below.

t *xyz* . . .⟨white-space⟩
t *xyz* . . . *dummy-arg*⟨white-space⟩
: Typeset word *xyz*; that is, set a sequence of ordinary glyphs named *x*, *y*, *z*, . . . , terminated by a space or newline; an optional second integer argument is ignored (this allows the formatter to generate an even number of arguments). Each glyph is set at the current drawing position, and the position is then advanced horizontally by the glyph's width. A glyph's width is read from its metrics in the font description file, scaled to the current type size, and rounded to a multiple of the horizontal motion quantum. Use the **C** command to emplace glyphs of special characters. The **t** command is a *groff* extension and is output only for devices whose *DESC* file contains the **tcommand** directive; see *groff_font*(5).

u *n xyz* . . .

u *xyz* . . . *dummy-arg*⟨white-space⟩
: Typeset word *xyz* with track kerning. As **t**, but after placing each glyph, the drawing position is further advanced horizontally by *n* basic units. The **u** command is a *groff* extension and is output only for devices whose *DESC* file contains the **tcommand** directive; see *groff_font*(5).

V *n*
: Vertically move the drawing position to *n* basic units from the top edge of the page. *n* cannot be negative.

v *n*
: Move the drawing position down *n* basic units. AT&T *troff* allowed negative *n;* GNU *troff* does not produce such values, but *groff*'s output driver library handles them.

w
: Indicate an inter-word space. No action is performed; the command is present to make the output more easily parsed. Only inter-word spaces on an output line (be they breakable or not) are thus described; those resulting from horizontal motion escape sequences are not. GNU *troff* issues this command but *groff*'s output driver library ignores it. See **h** and **H**.

Graphics commands

Each graphics or drawing command in the page description language starts with the letter "**D**", followed by one or two characters that specify a subcommand; this is followed by a fixed or variable number of integer arguments that are separated by a single space character. A "**D**", command may not be followed by another command on the same line (apart from a comment), so each "**D**" command is terminated by a syntactical line break.

GNU *troff* output follows AT&T *troff*'s output conventions (no space between command and subcommand, all arguments are preceded by a single space character), but *groff*'s parser allows optional space between the command letters and makes the space before the first argument optional. As usual, each space can be any sequence of tab and space characters.

Some graphics commands can take a variable number of arguments. In this case, they are integers representing a size measured in basic units **u**. The *h* arguments stand for horizontal distances where positive means right, negative left. The *v* arguments stand for vertical distances where positive means down, negative up. All these distances are offsets relative to the current location.

Each graphics command directly corresponds to a *troff* **\D** escape sequence. See subsection "Drawing commands" of *groff*(7).

Unless indicated otherwise, each graphics command directly corresponds to a similar *groff* **\D** escape sequence; see *groff*(7).

Unknown **D** commands are assumed to be device-specific. Its arguments are parsed as strings; the whole information is then sent to the postprocessor.

In the following command reference, the syntax element ⟨*line-break*⟩ means a *syntactical line break* as defined in subsection "Separation" above.

D~ $h_1\ v_1\ h_2\ v_2 \ldots h_n\ v_n$ ⟨line-break⟩
: Draw B-spline from current position to offset (h_1, v_1), then to offset (h_2, v_2) if given, etc., up to (h_n, v_n). This command takes a variable number of argument pairs; the current position is moved to the terminal point of the drawn curve.

Da $h_1\ v_1\ h_2\ v_2$ ⟨line-break⟩
: Draw arc from current position to $(h_1, v_1) + (h_2, v_2)$ with center at (h_1, v_1); then move the current position to the final point of the arc.

DC *d* ⟨line-break⟩
DC *d dummy-arg* ⟨line-break⟩
: Draw a solid circle using the current fill color with diameter *d* (integer in basic units **u**) with leftmost point at the current position; then move the current position to the rightmost point of the circle. An optional second integer argument is ignored (this allows the formatter to generate an even number of

arguments). This command is a GNU extension.

Dc *d* ⟨line-break⟩
: Draw circle line with diameter *d* (integer in basic units **u**) with leftmost point at the current position; then move the current position to the rightmost point of the circle.

DE *h v* ⟨line-break⟩
: Draw a solid ellipse in the current fill color with a horizontal diameter of *h* and a vertical diameter of *v* (both integers in basic units **u**) with the leftmost point at the current position; then move to the rightmost point of the ellipse. This command is a GNU extension.

De *h v* ⟨line-break⟩
: Draw an outlined ellipse with a horizontal diameter of *h* and a vertical diameter of *v* (both integers in basic units **u**) with the leftmost point at current position; then move to the rightmost point of the ellipse.

DF *color-scheme* [*component* . . .] ⟨line-break⟩
: Set fill color for solid drawing objects using different color schemes; the analogous command for setting the color of text, line graphics, and the outline of graphic objects is **m**. The color components are specified as integer arguments between 0 and 65536. The number of color components and their meaning vary for the different color schemes. These commands are generated by GNU *troff*'s escape sequences "**\D'F** . . .**'**" **\M** (with no other corresponding graphics commands). No position changing. This command is a GNU extension.

  DFc *cyan magenta yellow* ⟨line-break⟩
  : Set fill color for solid drawing objects using the CMY color scheme, having the 3 color components cyan, magenta, and yellow.

  DFd ⟨line-break⟩
  : Set fill color for solid drawing objects to the default fill color value (black in most cases). No component arguments.

  DFg *gray* ⟨line-break⟩
  : Set fill color for solid drawing objects to the shade of gray given by the argument, an integer between 0 (black) and 65536 (white).

  DFk *cyan magenta yellow black* ⟨line-break⟩
  : Set fill color for solid drawing objects using the CMYK color scheme, having the 4 color components cyan, magenta, yellow, and black.

  DFr *red green blue* ⟨line-break⟩
  : Set fill color for solid drawing objects using the RGB color scheme, having the 3 color components red, green, and blue.

Df *n* ⟨line-break⟩
: The argument *n* must be an integer in the range −32767 to 32767.

  $0 \le n \le 1000$
  : Set the color for filling solid drawing objects to a shade of gray, where 0 corresponds to solid white, 1000 (the default) to solid black, and values in between to intermediate shades of gray; this command is superseded by "**DFg**".

  $n < 0$ or $n > 1000$
  : Set the filling color to the color that is currently being used for the text and the outline, see command **m**. For example, the command sequence

```
mg 0 0 65536
Df -1
```

    sets all colors to blue.

  This command is a GNU extension.

Dl *h v* ⟨line-break⟩
: Draw line from current position to offset (*h*, *v*) (integers in basic units **u**); then set current position to the end of the drawn line.

Dp $h_1\ v_1\ h_2\ v_2 \ldots h_n\ v_n$ ⟨line-break⟩
: Draw a polygon line from current position to offset (h_1, v_1), from there to offset (h_2, v_2), etc., up to offset (h_n, v_n), and from there back to the starting position. For historical reasons, the position is changed by adding the sum of all arguments with odd index to the current horizontal position and the even ones to the vertical position. Although this doesn't make sense it is kept for compatibility. This command is a *groff* extension.

DP $h_1\ v_1\ h_2\ v_2 \ldots h_n\ v_n$ ⟨line-break⟩
: The same macro as the corresponding **Dp** command with the same arguments, but draws a solid polygon in the current fill color rather than an outlined polygon. The position is changed in the same way as with **Dp**. This command is a GNU extension.

Dt *n* ⟨line-break⟩
: Set the current line thickness to *n* (an integer in basic units **u**) if $n > 0$; if $n = 0$ select the smallest available line thickness; otherwise, the line thickness is made proportional to the type size, which is the default. For historical reasons, the horizontal position is changed by adding the argument to the current horizontal position, while the vertical position is not changed. Although this doesn't make sense, it is kept for compatibility. This command is a GNU extension.

Device control commands

Each device control command starts with the letter "**x**", followed by a space character (optional or arbitrary space or tab in GNU *troff*) and a subcommand letter or word; each argument (if any) must be preceded by a syntactical space. All "**x**", commands are terminated by a syntactical line break; no device control command can be followed by another command on the same line (except a comment).

The subcommand is basically a single letter, but to increase readability, it can be written as a word, i.e., an arbitrary sequence of characters terminated by the next tab, space, or newline character. All characters of the subcommand word but the first are simply ignored. For example, GNU *troff* outputs the initialization command "**x i**" as "**x init**" and the resolution command "**x r**" as "**x res**".

In the following, the syntax element ⟨*line-break*⟩ means a *syntactical line break* as defined in subsection "Separation" above.

xF *name* ⟨line-break⟩
: (*Filename* control command)
Use *name* as the intended name for the current file in error reports. This is useful for remembering the original file name when *groff* uses its internal piping mechanism. The input file is not changed by this command. This command is a GNU extension.

xf *n s* ⟨line-break⟩
: (*font* control command)
Mount font position *n* (a non-negative integer) with font named *s* (a text word); see *groff_font*(5).

xH *n* ⟨line-break⟩
: (*Height* control command)
Set character height to *n* (a positive integer in scaled points **z**). *Classical troff* used the unit points (**p**) instead; see section "Compatibility" below.

xi ⟨line-break⟩
: (*init* control command)
Initialize device. This is the third command of the header.

xp ⟨line-break⟩
: (*pause* control command)
Parsed but ignored. The classical documentation reads *pause device, can be restarted*.

xr *n h v* ⟨line-break⟩
(*resolution* control command)
Resolution is *n*, while *h* is the minimal horizontal motion, and *v* the minimal vertical motion possible with this device; all arguments are positive integers in basic units **u** per inch. This is the second command of the header.

xS *n* ⟨line-break⟩
(*Slant* control command)
Set slant to *n* degrees (an integer in basic units **u**).

xs ⟨line-break⟩
(*stop* control command)
Terminates the processing of the current file; issued as the last command of device-independent *troff* output.

xt ⟨line-break⟩
(*trailer* control command)
Generate trailer information, if any. In **groff**, this is currently ignored.

xT *xxx* ⟨line-break⟩
(*Typesetter* control command)
Set the name of the output driver to *xxx*, a sequence of non-whitespace characters terminated by whitespace. The possible names correspond to those of *groff*'s **−T** option. This is the first command of the header.

xu *n* ⟨line-break⟩
(*underline* control command)
Configure underlining of spaces. If *n* is 1, start underlining of spaces; if *n* is 0, stop underlining of spaces. This is needed for the **cu** request in **nroff** mode and is ignored otherwise. This command is a GNU extension.

xX *anything* ⟨line-break⟩
(*X-escape* control command)
Send string *anything* uninterpreted to the device. If the line following this command starts with a **+** character this line is interpreted as a continuation line in the following sense. The **+** is ignored, but a newline character is sent instead to the device, the rest of the line is sent uninterpreted. The same applies to all following lines until the first character of a line is not a **+** character. This command is generated by the *groff* escape sequence **\X**. Line continuation is a GNU extension.

Legacy compressed encoding

AT&T *troff* primarily emitted glyphs by writing two digits (a motion) followed by a single character corresponding to a glyph. This syntax is less a command itself than a compressed encoding of the **c** and **h** commands.

ddc Move right *dd* (exactly two decimal digits) basic units **u**, then print glyph with single-letter name *c*.

In *groff*, arbitrary *syntactical space* around and within this command is allowed to be added. Only when a preceding command on the same line ends with an argument of variable length a separating space is obligatory. In *classical troff*, large clusters of these and other commands were used, mostly without spaces; this made such output almost unreadable.

For modern high-resolution devices, this command is impractical because the widths of the glyphs have a greater magnitude in basic units than two decimal digits can represent. In *groff*, it is used only for output to the **X75**, **X75−12**, **X100**, and **X100−12** devices. For others, the commands **t** and **u** provide greater functionality and superior troubleshooting capacity.

Compatibility

The page description language of AT&T *troff* was first documented in "A Typesetter-independent TROFF", by Brian Kernighan, and by 1992 the AT&T *troff* manual was updated to incorporate a description of it.

groff's page description language is compatible with this specification except in the following aspects.

- AT&T device-independent *troff*'s quasi-device independence is not yet implemented.
- The printing hardware of the early 1980s differed from today's. *groff*'s output device names also differ from those of AT&T *troff* For example, the PostScript device in AT&T *troff,* **post** (implemented by the driver command *dpost*), has a resolution of only 720 units per inch, suitable for printers of decades past. *groff*'s **ps** device has a resolution of 72000 units per inch. In principle, by implementing a rescaling mechanism, *groff* could come to emulate AT&T's **post** device.
- While the B-spline command **D~** is reliably interpreted by *groff*'s page description language parser, some output drivers don't implement drawing routines for it.
- In GNU *troff,* the argument to the commands **s** and **x H** uses an implicit unit of scaled points **z** whereas AT&T *troff* uses spacing points **p**. This isn't an incompatibility, but a compatible extension, for both units coincide for any device without a **sizescale** directive in its *DESC* file, including all postprocessors from AT&T and *groff*'s text (*nroff*-mode) devices. *groff* devices that use **sizescale** either do not exist for AT&T *troff* have a different name, or seem to have a different resolution. So conflicts are very unlikely.
- The drawing position after the commands "**Dp**", "**DP**", and "**Dt**", are processed is illogical. Since old versions of GNU *troff* had this wart, we've retained it for compatibility, but may change it in the future. Wrap these drawing commands with the **\Z** escape sequence to both overcome the illogical positioning and keep your input working consistently regardless of the wart's presence in the implementation.

The differences between *groff* and *classical troff* are documented in *groff_diff*(7).

Files

*/usr/share/groff/1.24.2/font/dev*name*/DESC*
: describes the output device *name*.

Authors

James Clark wrote an early version of this document, which described only the differences between AT&T device-independent *troff*'s page description language and that of GNU *troff*. It has since been expanded and revised by Bernd Warken and G. Branden Robinson.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

"Troff User's Manual" by Joseph F. Ossanna, 1976 (revised by Brian W. Kernighan, 1992), AT&T Bell Laboratories Computing Science Technical Report No. 54, widely called simply "CSTR #54", documents the language, device and font description file formats, and device-independent page description language referred to collectively in *groff* documentation as "AT&T *troff*".

"A Typesetter-independent TROFF" by Brian W. Kernighan, 1982, AT&T Bell Laboratories Computing Science Technical Report No. 97, (CSTR #97), provides additional insights into the device and font description file formats and device-independent page description language.

groff(1)
: documents the **-Z** option and contains pointers to further *groff* documentation.

groff(7)
: describes the *groff* language, including its escape sequences and system of units.

groff_font(5)
: details the scaling parameters of *DESC* (device description) files.

troff(1)
: generates the language documented here.

roff(7)
: presents historical aspects and the general structure of *roff* systems.

groff_diff(7)
: enumerates differences between the output of AT&T *troff* and that of GNU *troff*.

gxditview(1)
: is a viewer for device-independent *troff* output.

Roff.js
: is a viewer for device-independent *troff* output written in JavaScript.

grodvi(1), *grohtml*(1), *grolbp*(1), *grolj4*(1), *gropdf*(1), *grops*(1), and *grotty*(1) are *groff* postprocessors.

Name

groff_tmac – macro files in the GNU *roff* typesetting system

Description

Definitions of macros, strings, and registers for use in a *roff*(7) document can be collected into *macro files*, *roff* input files designed to produce no output themselves but instead ease the preparation of other *roff* documents. There is no syntactical difference between a macro file and any other *roff* document; only its purpose distinguishes it. When a macro file is installed at a standard location, named according to a certain convention, and suitable for use by a general audience, it is termed a *macro package*. The "tmac" name originated in early Unix culture as an abbreviation of "*troff* macros".

Macro packages can be loaded by supplying the **–m** option to *troff*(1) or a *groff* front end. A macro file's name must have the form name.*tmac* (or *tmac.*name) and be placed in a "*tmac* directory" to be loadable with the "**–m** *name*" option. Section "Environment" of *troff*(1) lists these directories. Alternatively, a *groff* document requiring a macro file can load it with the **mso** ("macro source") request.

Macro files are named for their most noteworthy application, but a macro file need not define any macros. It can restrict itself to defining registers and strings or invoking other *groff* requests. It can even be empty.

Encode macro files in ISO 646:1991 IRV (US-ASCII) or ISO Latin-1 (8859-1). To prepare for a future *groff* release supporting UTF-8 input, restrict files to ISO 646 codes. *soelim*(1) by design does not interpret **mso** requests, and the encodings used by documents employing a macro file can vary.

Macro packages

Some macro packages ("major" or "full-service") assume responsibility for page layout and other critical functions; others ("supplemental" or "auxiliary") do not. GNU *roff* provides most major macro packages found in AT&T and BSD Unix systems, an additional full-service package, and many supplemental packages. Multiple full-service macro packages cannot be used by the same document. Auxiliary packages can, in general, be freely combined, though attention to their use of the *groff* language name spaces for identifiers (particularly registers, macros, strings, and diversions) should be paid. Name space management challenged AT&T *troff* users; GNU *troff*'s support for arbitrarily long identifiers affords few excuses for name collisions, apart from attempts at compatibility with the demands of historical documents.

Man pages

Two full-service macro packages are specialized for formatting Unix reference manuals; they do not support features like footnotes or multiple columnation.

an constructs man pages in a format introduced by Seventh Edition Unix (1979). Its macro interface is small, and the package widely used; see *groff_man*(7).

doc constructs man pages in a format introduced by 4.3BSD-Reno (1990). It provides many more features than *an*, but is also larger, more complex, and not as widely adopted; see *groff_mdoc*(7).

Because readers of man pages often do not know in advance which macros are used to format a given document, a wrapper is available.

andoc recognizes a document's use of *an* or *doc* and loads the corresponding macro package. Multiple man pages, in either format, can be handled; *andoc* reloads each macro package as necessary.

General full-service packages

The following packages each support composition of documents of any kind, from single-page memos to lengthy monographs. They are similar in functionality; select one that suits your taste.

me originates in 2BSD (1978); see *groff_me*(7).

mm originates in Programmer's Workbench (PWB) Unix 1.0 (1977); see *groff_mm*(7).

mom was contributed to *groff* in 2002, and freely exercises its many extended features. See *groff_mom*(7).

ms originates in Sixth Edition Unix (1975); see *groff_ms*(7).

Localization packages

For Western languages, an auxiliary package for localization sets the hyphenation mode and loads hyphenation patterns and exceptions. Localization files can also adjust the date format and provide translations of strings used by some of the full-service macro packages; alter the input encoding (see the next section); and change the amount of supplemental inter-sentence space. For Eastern languages, the localization file defines character classes and sets flags on them. By default, *troffrc* loads the localization file for English.

trans loads localized strings used by various macro packages after their localized forms have been prepared by a localization macro file.

groff provides the following localization files.

cs Czech; localizes *man*, *me*, *mm*, *mom*, and *ms*. Sets the input encoding to Latin-2 by loading *latin2.tmac*.

de
den German; localizes *man*, *me*, *mm*, *mom*, and *ms*. Sets the input encoding to Latin-1 by loading *latin1.tmac*.

de.tmac selects hyphenation patterns for traditional orthography, and *den.tmac* does the same for the new orthography ("Rechtschreibreform").

en English. Sets the input encoding to Latin-1 by loading *latin1.tmac*.

es Spanish; localizes *man*, *me*, *mm*, *mom*, and *ms*. Sets the input encoding to Latin-9 by loading *latin9.tmac*.

fr French; localizes *man*, *me*, *mm*, *mom*, and *ms*. Sets the input encoding to Latin-9 by loading *latin9.tmac*.

it Italian; localizes *man*, *me*, *mm*, *mom*, and *ms*. Sets the input encoding to Latin-1 by loading *latin1.tmac*.

ja Japanese.

pl Polish; localizes *man*, *me*, *mm*, *mom*, and *ms*. Sets the input encoding to Latin-2 by loading *latin2.tmac*.

ru Russian; localizes *man*, *me*, *mm*, *mom*, and *ms*. Sets the input encoding to KOI8-R by loading *koi8-r.tmac*.

sv Swedish; localizes *man*, *me*, *mm*, *mom*, and *ms*. Sets the input encoding to Latin-1 by loading *latin1.tmac*. Some of the localization of the *mm* package is handled separately; see *groff_mmse*(7).

zh Chinese.

Input encodings

Localization influences automatic hyphenation in two distinct but related respects. A macro file specific to a character coding identifies which character codes correspond to letters expected in the language's hyphenation pattern files and sets up case equivalences for those letters. A language's macro file determines which of these letters are equivalent to other letters for hyphenation purposes.

For example, in English, the letter "ñ" occurs in loan words. The *latin1.tmac* and *latin9.tmac* macro files define a hyphenation code for "ñ" and make "Ñ" equivalent to it. The English localization file *en.tmac* furthermore makes "ñ" equivalent to "n". In Spanish (*es.tmac*), however, "ñ" and "n" are *not* equivalent. The language localization file loads an appropriate encoding localization file; a document need not do so directly.

latin1
latin2
latin5
latin9 support the ISO Latin-1, Latin-2, Latin-5, and Latin-9 encodings (8859-1, 8859-2, 8859-9, and 8859-15, respectively).

koi8−r supports the KOI8-R encoding. KOI8-R code points in the range 0x80–0x9F are not valid input to GNU *troff*; see section "Identifiers" in *groff*(7). This should be no impediment to practical documents, as these KOI8-R code points do not encode letters, but box-drawing symbols and characters that are better obtained via special character escape sequences; see *groff_char*(7).

General auxiliary packages

The macro packages in this section are not intended for stand-alone use, but can add functionality to any other macro package or to plain ("raw") *groff* documents.

62bit provides macros for addition, multiplication, and division of 62-bit integers (allowing safe multiplication of signed 31-bit integers, for example).

hdtbl allows the generation of tables using a syntax similar to the HTML table model. This Heidelberger table macro package is not a preprocessor, which can be useful if the contents of table entries are determined by macro calls or string interpolations. Compare to *tbl*(1). It works only with the **ps** and **pdf** output devices. See *groff_hdtbl*(7).

papersize enables the paper format to be set on the command line with the "**−d paper=***fmt*" option to *troff*. Valid *fmt*s are the ISO and DIN formats " **A0**–**A6**", "**B0**–**B6**", "**C0**–**C6**", and "**D0**–**D6**"; the U.S. formats "**letter**", "**legal**", "**tabloid**", "**ledger**", "**statement**", and "**executive**"; and the envelope formats "**com10**", "**monarch**", and "**DL**". All formats, even those for envelopes, are in portrait orientation: the longer measurement is vertical. Appending "l" (ell) to any of these denotes landscape orientation instead. This macro file assumes one-inch horizontal margins, and sets registers recognized by the *groff* *man*, *mdoc*, *mm*, *mom*, and *ms* packages to configure them accordingly. If you want different margins, you will need to use those packages' facilities, or *troff* **ll** and/or **po** requests, to adjust them. An output device typically requires command-line options **−p** and **−l** to override the paper dimensions and orientation, respectively, defined in its *DESC* file; see subsection "Paper format" of *groff*(1). This macro file is normally loaded at startup by the *troffrc* file when formatting for a typesetter (but not a terminal).

pdfpic provides a single macro, **PDFPIC**, that operates in two modes. If it is not used with *gropdf*, the given *file* must be a PDF; **PDFPIC** then relies on the external program *pdftops*(1) to convert the PDF to an encapsulated PostScript (EPS) file and calls the **PSPIC** macro with which it shares an interface. If output is to a PDF document, **PDFPIC** uses the "**pdf: pdfpic**" device extension command (see *gropdf*(1)); the given *file* can then be a PDF or any graphic file format supported by *gropdf*.

Since **PDFPIC** needs to discover the width and height of the image (to check if sufficient room exists to place it on the page), it has dependencies on external programs as shown below.

	pdfinfo(1)	*file*(1)	*identify*(1)
.pdf	✓	✓	✓
.jpg	✗	✓	✓
.jp2	✗	✗	✓
other	✗	✗	✓

To include image formats such as PNG, PAM, and GIF, *gropdf* relies upon PerlMagick modules to embed the graphic. They are not needed for the types listed in the table above.

If the required programs are not available, *file* is treated as a PDF; failure is likely if it is not one.

pic supplies definitions of the macros **PS**, **PE**, **PF**, and **PY**, used with the *pic*(1) preprocessor. They center each picture. Use it if your document does not use a full-service macro package, or that package does not supply working *pic* macro definitions. Except for *man* and *mdoc*, those provided with *groff* already do so (exception: *mm* employs the name **PF** for a different purpose).

pspic provides a macro, **PSPIC**, that includes a PostScript graphic in a document. The **ps**, **dvi**, **html**, and **xhtml** output devices support such inclusions; for all other drivers, the image is replaced with a rectangular border of the same size. *pspic.tmac* is loaded at startup by the *troffrc* file.

Its syntax is as follows.

.PSPIC [**−L**|**−R**|**−C**|**−I** *n*] *file* [*width* [*height*]]

file is the name of the PostScript file; *width* and *height* give the desired width and height of the image. If neither a *width* nor a *height* argument is specified, the image's natural width (as given in the file's bounding box) or the current line length is used as the width, whatever is smaller. The *width* and *height* arguments may have scaling units attached; the default scaling unit is **i**. **PSPIC** scales the graphic uniformly in the horizontal and vertical directions so that it is no more than *width* wide and *height* high. Option **−C** centers the graphic horizontally; this is the default. **−L** and **−R** left- and right-align the graphic, respectively. **−I** indents the graphic by *n* (with a default scaling unit of **m**).

To use **PSPIC** within a diversion, we recommend extending it with the following code, assuring that the diversion's width completely covers the image's width.

```
.am PSPIC
.  vpt 0
\h'(\\n[ps-offset]u + \\n[ps-deswid]u)'
.  sp -1
.  vpt 1
..
```

Failure to load **PSPIC**'s image argument is not an error. (The **psbb** request does issue an error diagnostic.) To make such a failure fatal, append to the **pspic*error−hook** macro.

```
.am pspic*error-hook
.  ab
..
```

ptx provides a macro, **xx**, to format permuted index entries as produced by the GNU *ptx*(1) program. If your formatting needs differ, copy the macro into your document and adapt it.

rfc1345 defines special character escape sequences named for the glyph mnemonics specified in RFC 1345 and the digraph table of the Vim text editor. See *groff_rfc1345*(7).

sboxes offers an interface to the "**pdf: background**" device extension command supported by *gropdf*(1). Using this package, *groff ms* documents can draw colored rectangles beneath any output.

.BOXSTART SHADED *color* **OUTLINED** *color* **INDENT** *size* **WEIGHT** *size*

begins a box, where the argument after **SHADED** gives the fill color and that after **OUTLINED** the border color. Omit the former to get a borderless filled box and the latter for a border with no fill. The specified **WEIGHT** is used if the box is **OUTLINED**.

INDENT precedes a value that leaves a gap between the border and the contents inside the box.

Each *color* must be a defined *groff* color name, and each *size* a valid *groff* numeric expression. The keyword/value pairs can be specified in any order.

Boxes can be stacked, so you can start a box within another box; usually the later boxes would be smaller than the containing box, but this is not enforced. When using **BOXSTART**, the left position is the current indent minus the **INDENT** in the command, and the right position is the left position (calculated above) plus the current line length and twice the indent.

.BOXSTOP
takes no parameters. It closes the most recently started box at the current vertical position after adding its **INDENT** spacing.

Your *groff* documents can conditionally exercise the *sboxes* macros. The register **GSBOX** is defined if the package is loaded, and interpolates a true value if the **pdf** output device is in use.

sboxes furthermore hooks into the *groff_ms*(7) package to receive notifications when footnotes are growing, so that it can close boxes on a page before footnotes are printed. When that condition obtains, *sboxes* will close open boxes two points above the footnote separator and re-open them on the next page. (This amount probably will not match the box's **INDENT**.)

See "Using PDF boxes with *groff* and the *ms* macros" for a demonstration.

trace aids the debugging of *groff* documents by tracing macro calls. See *groff_trace*(7).

www defines macros corresponding to HTML elements. See *groff_www*(7).

Naming

AT&T *nroff* and *troff* were implemented before the conventions of the modern C *getopt*(3) call evolved, and used a naming scheme for macro packages that looks oddly terse to modern eyes. The formatter's **−m** option was the main means of loading a macro package, and its argument had to follow immediately without intervening space. This looked like a long option name preceded by a single minus—a sensation in the computer stone age. Macro packages therefore came to be known by names that started with the letter "m", which was omitted from the name of the macro file as stored on disk. For example, the manuscript macro package was stored as *tmac.s* and loaded with the option **−ms**. It has since become conventional in operating systems to use a suffixed file name extension to suggest a file type or format, thus we see *roff* documents with names ending in *.man*, *.me*, and so on.

groff commands permit space between an option and its argument. The syntax "**groff −m s**" makes the macro file name more clear but may surprise users familiar with the original convention, unaware that the package's "real" name was "s" all along. For such packages of long pedigree, *groff* accommodates different users' expectations by supplying wrapper macro files that load the desired file with **mso** requests. Thus, all of "**groff −m s**", "**groff −m ms**", "**groff −ms**", and "**groff −mms**" serve to load the manuscript macros.

Inclusion

The traditional method of employing a macro package is to specify the "**−m** *package*" option to the formatter, which then reads *package*'s macro file prior to any input. Historically, *package* was sought in a file named *tmac.*package (that is, with a "**tmac.**" prefix). GNU *troff* searches for package.*tmac* in the macro path; if not found, it looks for *tmac.*package instead, and vice versa.

Alternatively, one could include a macro file with the request "**so** *file-name*"; the argument is resolved as *fopen*(3) would, from the current working directory of the formatter. This approach was inadequate to locate macro packages, since systems stored them in varying locations. GNU *troff* offers an improved feature in the similar request "**mso** *package-file-name*", which searches the macro path for *package-file-name*. Because its argument is a file name, its "**.tmac**" component must be included for the file to be found.

If a sourced file requires preprocessing, for example if it includes *tbl* tables or *eqn* equations, the preprocessor *soelim*(1) must be used. This can be achieved with a pipeline or by specifying the **−s** option to *groff*(1). *man*(1) librarian programs typically run *soelim* automatically. (As a rule, macro packages themselves do not require preprocessing.)

Writing macros

A *roff*(7) document is a text file that is enriched by predefined formatting constructs, such as requests, escape sequences, strings, numeric registers, and macros from a macro package. *roff*(7) describes these elements.

To give a document a personal style, it is most useful to extend the existing elements by defining some macros for repeating tasks; the best place for this is near the beginning of the document or in a separate file.

Macros without arguments are just like strings. But the full power of macros occurs when arguments are passed with a macro call. Within the macro definition, the arguments are available as the escape sequences

\$1, . . ., **\$9**, **\$[. . .]**, **\$***, and **\$@**, the name under which the macro was called is in **\$0**, and the number of arguments is in register **\n[.$]**; see *groff* (7).

Drafting macros

One approach temporarily disables escape sequences by bracketing a macro definition with **eo** and **ec** requests.

```
.eo
.ds midpart was called with the following
.de print_args
\f[I]\$0\f[] \*[midpart] \n[.$] arguments: \$*
..
.ec
```

The above procedure has limitations; it is unsuitable for a macro that requires certain interpolations at the time it is defined, or for indirect definitions of identifiers. See section "Copy mode" of *groff* (7). In such cases, you might define and test the macro with the escape character doubled before escape sequences that are interpreted even in copy mode, then bracket it with **eo** and **ec** requests, un-double the escape characters, then test again.

Tips for macro definitions

- Use only control lines in macro definitions; that is, start every input line with a control character. *groff*'s **nop** request makes use of text lines unnecessary.

```
.de Text
.  if (\\n[.$] == 0) \
.    return
.  nop \&\\$*\&
..
```

- Write a comment macro that works in both draft and non-draft modes; since the escape character is disabled in draft mode, trouble might occur when comment escape sequences are used.

```
.de c
..
.c This is my comment.
```

- Comment lengthy macro definitions.
- Use empty requests, and indentation after control characters, to clarify a macro's structure.

Authors

This document was written by Werner Lemberg and G. Branden Robinson.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

groff (1) is an overview of the *groff* system.

groff_man(7), *groff_mdoc*(7), *groff_me*(7), *groff_mm*(7), *groff_mom*(7), *groff_ms*(7), *groff_rfc1345*(7), *groff_trace*(7), and *groff_www*(7) are *groff* macro packages.

groff (7)	summarizes the language recognized by GNU *troff*.
troff (1)	documents the default macro file search path.

Name

groff – GNU *roff* language reference

Description

groff is short for GNU *roff*, a free reimplementation of the AT&T device-independent *troff* typesetting system. See *roff* (7) for a survey of and background on *roff* systems.

This document is intended as a reference. The primary *groff* manual, *Groff: The GNU Implementation of troff*, by Trent A. Fisher and Werner Lemberg, is a better resource for learners, containing many examples and much discussion. It is written in Texinfo; you can browse it interactively with "info groff". Additional formats, including plain text, HTML, TEX DVI, and PDF, may be available in */usr/share/doc/groff*.

We apply the term "groff" to the language documented here, the GNU implementation of the overall system, the project that develops that system, and the command of that name. In the first sense, *groff* is an extended dialect of the *roff* language, for which many similar implementations exist.

GNU *troff*, installed on this system as *troff* (1), is the *formatter:* a program that reads device and font descriptions (*groff_font*(5)), interprets the *groff* language expressed in input text, and translates it into a device-independent page description language (*groff_out*(5)) that is usually then post-processed by an output driver to produce PostScript, PDF, HTML, DVI, or terminal output. We speak of "the formatter" when describing behavior that is generally true of *troff* and *nroff* programs.

Input format

Organize input to GNU *troff* into lines separated by the Unix newline character (U+000A), using the character encoding it recognizes: ISO Latin-1 (8859-1). We recommend use of ISO 646:1991 IRV (US-ASCII) or (equivalently) the Basic Latin subset of ISO 10646 (Unicode); see *groff_char*(7).

Some control characters (from the sets "C0 Controls" and "C1 Controls" as Unicode describes them) are invalid as input characters. GNU *troff* discards them upon reading. (It also emits a warning in category "input"; see section "Warnings" of *troff* (1).) It processes a character sequence "foo", followed by an invalid character and then "bar", as "foobar".

Invalid input characters comprise **0x00**, **0x0B**, **0x0D–0x1F**, and **0x80–0x9F**. (Historically, control characters like ASCII STX, ETX, and BEL (Control+B, Control+C, and Control+G) respectively) have been observed in *roff* documents, particularly in macro packages employing them as delimiters with the output comparison operator to try to avoid collisions with the content of arbitrary user-supplied parameters (see subsection "Conditional expressions" below). We discourage this expedient; in GNU *troff* it is unnecessary (outside of compatibility mode) because the program parses delimited arguments at a different input level than their surrounding context. See section "Miscellaneous" of *groff_diff* (7).) GNU *troff* uses some of these code points for internal purposes, making non-trivial the extension of the program to accept UTF-8 or other encodings that use characters from these ranges.

Syntax characters

Several input characters are syntactically significant to *groff*. The most important of these distinguish *control lines,* which instruct the formatter, from *text lines* that are formatted as output.

. A dot at the beginning of an input line marks it as a control line. It can also follow the **el** and **nop** requests, and the condition in "**if**", **ie**, and "**while**" requests. The control character invokes requests and calls macros by the name that follows it. The **cc** request can change the control character.

' The neutral apostrophe is the *no-break control character,* recognized where the control character is. It suppresses the (first) break implied by the **bp**, **ce**, **cf**, **fi**, **fl**, "**in**", **nf**, **rj**, **sp**, **ti**, and **trf** requests. The requested operation takes effect at the next break. It makes **br** and **brp** nilpotent. The no-break control character can be changed with the **c2** request. When formatted, "'" may be typeset as a typographical quotation mark; use the **\[aq]** special character escape sequence to format a neutral apostrophe glyph.

" The neutral double quote can be used to enclose arguments to macros and strings, and is required if those arguments contain space or tab characters. The requests "**as**", **as1**, **cf**, **char**, **ds**, **ds1**, **fchar**, **fschar**, **mso**, **msoquiet**, **nx**, "**open**", **opena**, **pi**, **schar**, "**so**", **soquiet**, **sy**, and **trf** strip a leading neutral double quote from their final arguments to allow embedding of leading spaces. All such arguments are "strings" in a

general sense, representing character sequences, file names, operating system commands, or parameters to an output device extension command. To include a double quote inside a quoted argument, use the **\[dq]** special character escape sequence (which also serves to typeset the glyph in text).

\ A backslash introduces an escape sequence. The escape character can be changed with the **ec** request; **eo** disables escape sequence recognition. Use the **\[rs]** special character escape sequence to format a backslash glyph, and **\e** to typeset the glyph of the current escape character.

(An opening parenthesis is special only in certain escape sequences; when recognized, it introduces an argument of exactly two characters. *groff* offers the more flexible square bracket syntax.

[An opening bracket is special only in certain escape sequences; when recognized, it introduces an argument (list) of any length, not including a closing bracket.

] A closing bracket is special only when an escape sequence using an opening bracket as an argument delimiter is being interpreted. It ends the argument (list).

Additionally, the Control+A character (U+0001) in text is interpreted as a *leader* (see below).

Horizontal whitespace characters are significant to *groff,* but trailing spaces on text lines are ignored.

space On control lines and within bracketed escape sequences, spaces separate arguments. On text lines, they separate words. Multiple adjacent space characters in text cause *groff* to attempt end-of-sentence detection on the preceding word (and trailing punctuation). The amount of space between words and sentences is controlled by the **ss** request. When filling is enabled (the default), a line may be broken at a space. When adjustment is enabled (the default), inter-word spaces are expanded until the output line reaches the configured length. An adjustable but non-breaking space is available with **\~**. To get a space of fixed width, use one of the escape sequences ‘ \ ’ (the escape character followed by a space), **\0**, **\|**, **\^**, or **\h**; see section “Escape sequences” below.

newline On text lines, a newline formats an inter-word space and, if filling is enabled, triggers end-of-sentence recognition on the preceding text. See section “Line continuation” below.

tab A tab character on a text line causes the drawing position to advance to the next defined tab stop.

Sentences

Configure the sets of characters that potentially end sentences or are transparent to sentence endings with the **cflags** request. Use the **ss** request to change—or eliminate—supplemental inter-sentence space.

Tabs and leaders

The formatter interprets input horizontal tab characters (“tabs”) and Control+A characters (“leaders”) into movements to the next tab stop. Tabs simply move to the next tab stop; leaders place enough periods to fill the space. Tab stops are by default located every half inch measured from the drawing position corresponding to the beginning of the input line; see section “Page geometry” of *roff* (7). Tabs and leaders do not cause breaks and therefore do not interrupt filling. Tab stops can be configured with the **ta** request, and tab and leader glyphs with the **tc** and **lc** requests, respectively.

Line continuation

When filling is enabled, input and output line breaks generally do not correspond. The *roff* language therefore distinguishes input and output line continuation.

A backslash \ immediately followed by a newline, sometimes discussed as *newline*, suppresses the effects of that newline on the input. The next input line thus retains the classification of its predecessor as a control or text line. *newline* is useful for managing line lengths in the input during document maintenance; you can break an input line at a space, or in the middle of a word, request invocation, macro call, or escape sequence. Input line continuation is invisible to the formatter, with two exceptions: the | operator recognizes the new input line, and the input line counter register **.c** increments.

The **\c** escape sequence continues an *output* line. Nothing on the input line after it is formatted. In contrast to *newline*, a line after **\c** is treated as a new input line, so a control character is recognized at its beginning. The visual results depend on whether filling is enabled. An intervening control line that causes a break overrides

\c, flushing out the pending output line in the usual way. The register **.int** interpolates a positive value only if the pending output line has been continued with **\c**; this datum is associated with the environment.

Colors

groff supports color output with a variety of color spaces and up to 16 bits per channel. Some devices, particularly terminals, may be more limited. When color support is enabled, two colors are current at any given time: the *stroke color,* with which glyphs, rules (lines), and geometric objects like circles and polygons are drawn, and the *fill color,* which can be used to paint the interior of a closed geometric figure. The **color**, **defcolor**, **gcolor**, and **fcolor** requests; **\m** and **\M** escape sequences; and **.color**, **.m**, and **.M** registers exercise color support.

Each output device has a color named "**default**", which cannot be redefined. A device's default stroke and fill colors are not necessarily the same. For the **dvi**, **html**, **pdf**, **ps**, and **xhtml** output devices, *troff* automatically loads a macro file defining many color names at startup. By the same mechanism, the devices supported by *grotty*(1) recognize the eight standard ISO 6429/ECMA-48 color names. (These are known vulgarly as "ANSI" colors, after its X3.64 standard, now withdrawn.)

Measurements

Express numeric parameters that specify measurements as integers or decimal fractions with an optional scaling unit suffixed. A *scaling unit* is a letter that immediately follows the magnitude of a measurement. Digits after the decimal point are optional.

The formatter scales measurements by the specified scaling unit, storing them internally (with any fractional part discarded) in basic units. The device resolution can therefore be obtained by storing a value of "**1i**" to a register, then reading the register.

u Basic unit; it is at least as small as any other unit.
i Inch; defined as 2.54 centimeters.
c Centimeter.
p Point; a typesetter's unit used for measuring type size. There are 72 points to an inch.
P Pica; another typesetter's unit. There are 6 picas to an inch and 12 points to a pica.
z Typographical point; like **p**, but used only with type sizes, to overcome a limitation of AT&T *troff*.
s Scaled point.
f Multiplication by 65,536; scales decimal fractions in the interval [0, 1] to 16-bit unsigned integers.

The magnitudes of other scaling units depend on the text formatting parameters in effect.

m Em; an em is equal to the current type size in points.
n En; on typesetters, an en is one-half em, but on terminals an en equals an em.
v Vee; distance between text baselines.
M Hundredth of an em.

Motion quanta

The basic unit **u** is not necessarily an output device's smallest addressable length; **u** can be smaller to avoid integer rounding errors. The minimum distances that a device can work with in the horizontal and vertical directions are termed its *motion quanta,* stored in the **.H** and **.V** registers, respectively. The formatter rounds measurements to applicable motion quanta. Half-quantum fractions round toward zero.

Default units

A general-purpose register (one created or updated with the **nr** request; see section "Registers" below) is implicitly dimensionless, or reckoned in basic units if interpreted in a measurement context. But it is convenient for many requests and escape sequences to infer a scaling unit for an argument if none is specified. An explicit scaling unit (not after a closing parenthesis) can override an undesirable default. Effectively, the default unit is suffixed to the expression if a scaling unit is not already present. GNU *troff*'s use of integer arithmetic in numeric expressions should also be kept in mind.

Numeric expressions

When evaluated, a *numeric expression* interpolates an integer. GNU *troff* recognizes the following operators.

	+	addition
	−	subtraction
	*	multiplication
	/	truncating division
	%	modulus
unary	+	assertion, motion, incrementation
unary	−	negation, motion, decrementation
	;	scaling
	>?	maximum
	<?	minimum
	<	less than
	>	greater than
	<=	less than or equal
	>=	greater than or equal
	=	equal
	==	equal
	&	logical conjunction (“and”)
	:	logical disjunction (“or”)
	!	logical complementation (“not”)
	( )	precedence
	|	boundary-relative measurement

troff provides a set of mathematical and logical operators familiar to programmers—as well as some unusual ones—but supports only integer arithmetic. (Provision is made for interpreting and reporting decimal fractions in certain cases.) The internal data type used for computing results depends on the host machine but is at least a 32-bit signed integer, which suffices to represent magnitudes within a range of ±2 billion. (If that’s not enough, see *groff_tmac*(5) for the *62bit.tmac* macro package.) Arithmetic saturates. (If overflow would occur, GNU *troff* emits a warning in category “**range**”. See section “Warnings” of *troff* (1).)

Arithmetic infix operators perform a function on the numeric expressions to their left and right; they are **+** (addition), **−** (subtraction), ***** (multiplication), **/** (truncating division), and **%** (modulus). *Truncating division* rounds to the integer nearer to zero, no matter how large the fractional portion. Division and modulus by zero are errors and abort evaluation of a numeric expression.

Arithmetic unary operators operate on the numeric expression to their right; they are **−** (negation) and **+** (assertion—for completeness; it does nothing). The unary minus must often be used with parentheses to avoid confusion with the decrementation operator, discussed below.

The sign of the modulus of operands of mixed signs is determined by the sign of the first. Division and modulus operators satisfy the following property: given a dividend *a* and a divisor *b*, a quotient *q* formed by “**(a / b)**” and a remainder *r* by “**(a % b)**”, then $qb + r = a$.

GNU *troff*’s scaling operator, used with parentheses as (*c***;***e*), evaluates a numeric expression *e* using *c* as the default scaling unit. If *c* is omitted, scaling units are ignored in the evaluation of *e*. GNU *troff* also provides a pair of operators to compute the extremum of two operands: **>?** (maximum) and **<?** (minimum).

Comparison operators comprise **<** (less than), **>** (greater than), **<=** (less than or equal), **>=** (greater than or equal), and **=** (equal, with a synonym **==**). When evaluating a comparison, the formatter replaces it with “**0**” if it is false and “**1**” if true. In the *roff* language, positive values are true, others false.

Operate on truth values with the logical operators **&** (logical conjunction or “and”) and **:** (logical disjunction or “or”). They evaluate as comparison operators do. A logical complementation (“not”) operator, **!,** works only within “**if**”, “**ie**”, and “**while**” requests. Furthermore, the formatter recognizes **!** only at the beginning of a numeric expression not contained by another numeric expression. In other words, **!** must be the “outermost”

operator. Its presence elsewhere causes the expression to evaluate false. (GNU *troff* emits a warning in category "**number**". See section "Warnings" of *troff* (1).) This unfortunate limitation maintains compatibility with AT&T *troff*. Test a numeric expression for falsity within a complex expression by comparing it to a false value.

The *roff* language has no operator precedence: expressions are evaluated strictly from left to right, in contrast to schoolhouse arithmetic. Use parentheses **( )** to impose a desired precedence upon subexpressions.

For many requests and escape sequences that cause motion on the page, the unary operators **+** and **−** work differently when leading a numeric expression. They then indicate a motion relative to the drawing position: positive is down in vertical contexts, right in horizontal ones.

+ and **−** are also treated differently by the following requests and escape sequences: **bp**, **in**, **ll**, **pl**, **pn**, **po**, **ps**, **pvs**, **rt**, **ti**, **\H**, **\R**, and **\s**. Here, leading plus and minus signs serve as incrementation and decrementation operators, respectively. To negate an expression in these contexts, subtract it from zero or include the unary minus in parentheses with its argument.

A leading **|** operator indicates a motion relative not to the drawing position but to a boundary. For horizontal motions, the measurement specifies a distance relative to a drawing position corresponding to the beginning of the *input* line. By default, tab stops reckon movements in this way. Most escape sequences do not; **|** tells them to do so. For vertical motions, the **|** operator specifies a distance from the first text baseline on the page or in the current diversion, using the current vertical spacing.

The delimited escape sequence **\B** tests its argument for validity as a numeric expression.

A register interpolated as an operand in a numeric expression must have an Arabic format; luckily, this is the default.

Because spaces separate arguments to requests, spaces are not allowed in numeric expressions unless parentheses surround the (sub)expression containing them.

Identifiers

An *identifier* labels a GNU *troff* datum such as a register, name (macro, string, or diversion), typeface, color, special character or character class, hyphenation language code, environment, or stream. Valid identifiers consist of one or more ordinary characters. An *ordinary character* is any Unicode Basic Latin character that is not a space and not the escape character; recall section "Input format" above.

An identifier with a closing bracket ("]") in its name can't be accessed with bracket-form escape sequences that expect an identifier as a parameter. Similarly, the identifier "(" can't be interpolated *except* with bracket forms.

Beginning a macro, string, or diversion name with the character "[" or "]" forecloses use of the *refer* (1) preprocessor, which recognizes input lines starting with ".[" and ".]" as bibliographic reference delimiters.

The delimited escape sequence **\A** tests its argument for validity as an identifier.

The formatter's handling of undefined identifiers is context-dependent. There is no way to invoke an undefined request; such syntax is interpreted as a macro call instead. If the identifier is interpreted as a string, macro, or diversion name, the formatter defines it as empty and interpolates nothing. (GNU *troff* emits a warning in category "**mac**". See section "Warnings" of *troff* (1).) Similarly, if the identifier is interpreted as a register name, the formatter initializes it to zero and interpolates that value. GNU *troff* emits a warning in category "**reg**"; see section "Warnings" in *troff* (1), and subsection "Interpolating registers" and section "Strings" below. Attempting to use an undefined typeface, special character or character class, color, environment, hyphenation language code, or stream generally provokes an error diagnostic.

Identifiers for requests, macros, strings, and diversions share one name space; special characters and character classes another. No other object types do.

Control characters

The formatter recognizes a control character only at the beginning of an input line, or at the beginning of a branch of a control structure request; see section "Control structures" below.

A few requests cause a break implicitly; invoke them with the no-break control character to prevent the break. Break suppression is its sole behavioral distinction. Employing the no-break control character to invoke requests that don't cause breaks is harmless but poor style.

The control character "**.**" and the no-break control character "**'**" can be changed with the **cc** and **c2** requests, respectively. Within a macro definition, register **.br** indicates the control character used to call it.

Invoking requests

A control character is optionally followed by tabs and/or spaces and then an identifier naming a request or macro. The invocation of an unrecognized request is interpreted as a macro call. Defining a macro with the same name as a request replaces the request. Deleting a request name with the **rm** request makes it unavailable. The **als** request can alias requests, permitting them to be wrapped or non-destructively replaced. See section "Strings" below.

There is no inherent limit on argument length or quantity. Most requests take one or more arguments, and ignore any they do not expect. A request may be separated from its arguments by tabs or spaces, but only spaces can separate an argument from its successor. Only one between arguments is necessary; any excess is ignored. GNU *troff* does not allow tabs for argument separation. (Plan 9 *troff* does.)

Generally, a space *within* a request argument is not relevant, not meaningful, or is supported by bespoke provisions, as with the **tl** request's delimiters. Some requests, like **ds**, interpret the remainder of the control line as a single argument. See section "Strings" below.

Spaces and tabs immediately after a control character are ignored. Commonly, authors use them to indent the source of documents or macro files.

Calling macros

If a macro of the desired name does not exist when called, the formatter creates it and assigns it an empty definition. (GNU *troff* emits a warning in category "**mac**". See section "Warnings" in *troff* (1).) Calling an undefined macro *does* end a macro definition naming it as its end macro (see section "Writing macros" below).

To embed spaces *within* a macro argument, enclose the argument in neutral double quotes '**"**'. Horizontal motion escape sequences are sometimes a better choice for arguments to be formatted as text.

The foregoing raises the question of how to embed neutral double quotes or backslashes in macro arguments when *those* characters are desired as literals. In GNU *troff*, the special character escape sequence **\[rs]** produces a backslash and **\[dq]** a neutral double quote.

In GNU *troff*'s AT&T compatibility mode, these characters remain available as **\(rs** and **\(dq**, respectively. AT&T *troff* did not consistently define these special characters, but its descendants can be made to support them. See *groff_font*(5). If even that is not feasible, see the "Calling Macros" section of the *groff* Texinfo manual for the complex macro argument quoting rules of AT&T *troff*.

Using escape sequences

Whereas requests must occur on control lines, escape sequences can occur intermixed with text and may appear in arguments to requests, macros, and other escape sequences. An escape sequence is introduced by the escape character, a backslash \. The next character selects the escape's function.

Escape sequences vary in length. Some take an argument, and of those, some have different syntactical forms for a one-character, two-character, or arbitrary-length argument. Others accept *only* an arbitrary-length argument. In the former scheme, a one-character argument follows the function character immediately, an opening parenthesis "(" introduces a two-character argument (no closing parenthesis is used), and an argument of arbitrary length is enclosed in brackets "[]". In the latter scheme, the user selects a delimiter character. A few escape sequences are idiosyncratic, and support both of the foregoing conventions (**\s**), designate their own termination sequence (**\?**), consume input until the next newline (**\!**, **\"**, **\#**), or support an additional modifier character (**\s** again, and **\n**). In no case can an escape sequence parameter contain an unescaped newline.

If the character that follows the escape character does not identify a defined operation, the formatter ignores the escape character. (GNU *troff* emits a warning in category "**escape**". See section "Warnings" in *troff* (1).)

Escape sequence interpolation is of higher precedence than escape sequence argument interpretation. This rule affords flexibility in using escape sequences to construct parameters to other escape sequences.

The escape character can be interpolated (**\e**). Requests permit the escape mechanism to be deactivated (**eo**) and restored, or the escape character changed (**ec**), and to save and restore it (**ecs** and **ecr**).

Delimiters

Some escape sequences that require parameters use delimiters. The neutral apostrophe **'** is a popular choice and shown in this document. The neutral double quote **"** is also commonly seen. Punctuation characters are the best choice (and most portable to other *troff* s, except for those meaningful in numeric expressions; see below.

The following escape sequences are not themselves delimited, and thus are allowed as delimiters: *space*, **\%**, **\|**, **\^**, **\{**, **\}**, **\'**, **\`**, **\–**, **_**, **\!**, **\?**, **\)**, **\/**, **\,**, **\&**, **\:**, **\~**, **\0**, **\a**, **\c**, **\d**, **\e**, **\E**, **\p**, **\r**, **\t**, and **\u**. However, we discourage using them this way; they can make the input confusing to read. (The *eqn*(1) and *tbl*(1) preprocessors use parameterized but non-delimited special character escape sequences **\(** and **\[** to bracket portions of their output.) An invalid escape sequence is valid as a delimiter if the character after the escape character would be valid.

The escape sequences **\D**, **\h**, **\H**, **\l**, **\L**, **\N**, **\R**, **\s**, **\S**, **\v**, and **\x** prohibit delimiters that are meaningful in numeric expressions, because they accept numeric expressions as (or within) their arguments. For consistency, GNU *troff* prohibits the same delimiters in the argument to the **tl** request. The "**if**", **ie**, and "**while**" requests each interpret their first argument as a conditional expression; only characters that are not meaningful as operators in that context can be used as output comparison delimiters. The following inputs are therefore invalid as delimiters in GNU *troff* .

- the numerals 0–9 and the decimal point "**.**"
- the (single-character) operators **+–/*%<>=&:()|**
- any escape sequences other than **\%**, **\:**, **\{**, **\}**, **\'**, **\`**, **\–**, **_**, **\!**, **\/**, **\c**, **\e**, and **\p**

Delimiter syntax is flexible (and laborious to describe) primarily for historical reasons; the foregoing restrictions need be kept in mind mainly when using GNU *troff* in AT&T compatibility mode. Normally, GNU *troff* keeps track of the nesting depth of escape sequence interpolations, so the only characters you need to avoid using as delimiters are those that appear in the arguments you input, not those that result from interpolation. Typically, **'** works fine. See section "Implementation differences" in *groff_diff* (7).

Dummy characters

As discussed in *roff* (7), the first character on an input line is treated specially. Further, formatting a glyph has many consequences on formatter state (see section "Environments" below). Occasionally, we want to escape this context or embrace some of those consequences without actually rendering a glyph to the output. **\&** interpolates a dummy character, which is constitutive of output but invisible. Its presence alters the interpretation context of a subsequent input character, and enjoys several applications: preventing the insertion of extra space after an end-of-sentence character, preventing interpretation of a control character at the beginning of an input line, preventing kerning between two glyphs, and permitting the **tr** request to remap a character to "nothing". **\)** works as **\&** does, except that it does not cancel a pending end-of-sentence state.

Page control

Discretionary page breaks can prevent the unwanted separation of content. A new page number takes effect during page ejection; see subsection "The implicit page trap" below. The **bp** request breaks the page, incrementing the page number by one (or setting it per the supplied argument). The **ne** request forces a page break if insufficient vertical space is available (it asserts "needed" space). **sv** requires vertical space as **ne** does, but also *saves* it for later output by the **os** request.

The **nl** register interpolates the vertical drawing position as of the most recently typeset output line. It does not necessarily (and often does not) represent that of the pending output line, because the formatter does not determine the position of its baseline until it is output. Assigning a value to **nl** sets the vertical drawing

position in advance of further modifications to baseline positioning arising from alterations to type size, changes to vertical spacing, or application of extra pre- or post-vertical spacing.

When the formatter starts, the transition to the first page has not yet happened—**nl** is negative. If you plant a page location trap at vertical position "**0**" (idiomatically to format a header), you can assign a negative value to **nl** to spring that trap even if the page has already started (see subsection "Page location traps" below).

Control structures

groff has "if" and "while" control structures like other languages. However, the syntax for grouping multiple input lines in the branches or bodies of these structures is unusual.

They have a common form: the request name is (except for **el** "else") followed by a conditional expression *cond-expr*; the remainder of the line, *input*, is interpreted as if it were an input line. Any quantity of spaces between arguments to requests serves only to separate them; leading spaces in *input* are therefore not seen. *input* effectively *cannot* be omitted; if *cond-expr* is true and *input* is empty, the newline at the end of the control line is interpreted as a blank line (and therefore a blank text line).

It is frequently desirable for a control structure to govern more than one request, macro call, or text line, or combination of the foregoing. The opening and closing brace escape sequences **\{** and **\}** perform such grouping. Brace escape sequences outside of control structures have no meaning and produce no output.

\{ should appear (after optional spaces and tabs) immediately subsequent to the request's conditional expression. **\}** should appear on a line with other occurrences of itself as necessary to match **\{** sequences. It can be preceded by a control character, spaces, and tabs. Input after any quantity of **\}** sequences on the same line is processed only if all the preceding conditions to which they correspond are true. Furthermore, a **\}** closing the body of a "**while**" request must be the last such escape sequence on an input line.

GNU *troff* treats the body of a "**while**" request similarly to that of a **de** request (albeit one not read in copy mode), but stores it under an internal name and deletes it when the loop finishes. The operation of a macro containing a "**while**" request can slow significantly if its body is large. Each time GNU *troff* interpolates the macro, it parses and stores the "**while**" body again. An often better solution—and one that is more portable, since AT&T *troff* lacked the "**while**" request—is to instead write a recursive macro, which is parsed only once (unless you redefine it). To prevent infinite loops, GNU *troff* limits the default number of available recursion levels to 1,000 or somewhat less (because things other than macro calls can be on the input stack). You can disable this protective measure, or alter the limit, by setting the **slimit** register. See section "Debugging" below.

Conditional expressions

The "**if**", **ie**, and "**while**" requests test the truth values of numeric expressions. They also support several additional Boolean operators; the members of this expanded class are termed *conditional expressions*; their truth values are as shown below.

***cond-expr*...**	**...is true if...**
'*s1***'***s2***'**	*s1* produces the same formatted output as *s2*.
c *g*	a character *g* is defined.
d *m*	a string, macro, diversion, or request *m* is defined.
e	the current page number is even.
F *f*	a font named *f* is available.
m *c*	a color named *c* is defined.
n	the formatter is in *nroff* mode.
o	the current page number is odd.
r *n*	a register named *n* is defined.
S *s*	a font style named *s* is available.
t	the formatter is in *troff* mode.
v	n/a (historical artifact; always false).

If the first argument to an "**if**", **ie**, or "**while**" request begins with a non-alphanumeric character apart from "**!**" (see below) and is not a numeric expression, the formatter performs an *output comparison test.* Shown first in

the table above, the *output comparison operator* interpolates a true value if formatting its comparands *s1* and *s2* produces the same output commands. Other delimiters can be used in place of the neutral apostrophes; see section "Delimiters" above. *troff* formats *s1* and *s2* in separate scratch buffers; after comparison, it discards the resulting data. The resulting glyph properties, including font family, style, size, and slant, must match, but not necessarily the requests and/or escape sequences used to obtain them. Motions must match in orientation and magnitude to within the applicable horizontal or vertical motion quantum of the device, after rounding.

Surround the comparands with **\?** to avoid formatting them; this causes them to be compared character by character, as with string comparisons in other programming languages. Since GNU *troff* reads comparands protected with **\?** in copy mode, they need not be syntactically valid. The escape character is still lexically recognized, however, and consumes the next character.

The above operators can't be combined with most others, but a leading "**!**", not followed immediately by spaces or tabs, complements an expression. Spaces and tabs are optional immediately after the "**c**", "**d**", "**F**", "**m**", "**r**", and "**S**" operators, but right after "**!**", they end the predicate and the conditional evaluates true. (This bizarre behavior maintains compatibility with AT&T *troff*.)

Conditional operators do not create *roff* language objects as interpolations with **\n** and ***** escape sequences do.

Syntax reference conventions

In the following request and escape sequence specifications, most argument names were chosen to be descriptive. GNU *troff* reads arguments named *character-sequence, command, contents, file,* and *message* in copy mode (see section "Copy Mode" below) until the end of the input line. A *character-sequence* comprises one or more ordinary, special, or indexed characters; spaces; or escape sequences that interpolate only these. We name the remaining arguments for clarity; they are also *character-sequences.* A neutral double quote "**"**" can optionally prefix a *character-sequence;* the formatter discards one if present, permitting initial embedded spaces in the argument. *input* refers to arbitrary character sequences (up to a newline or delimiter) that GNU *troff* fully interprets, in contrast to copy mode.

A few denotations are more specific.

b is a numerical expression evaluated as a Boolean; positive values are true, others false.

c denotes a single character—ordinary, special, or indexed.

command is an instance of *contents* (see below) to be passed to the system as a command (potentially with arguments). GNU *troff* strips a leading neutral double quote, allowing embedded leading spaces.

contents is arbitrary input, excluding an unescaped newline, read in copy mode. GNU *troff* strips a leading neutral double quote, allowing embedded leading spaces.

div is a diversion identifier.

env is an environment identifier.

file is an instance of *contents* naming a file on the system. GNU *troff* strips a leading neutral double quote, allowing embedded leading spaces. GNU *troff* does not accept newlines (line feeds) in file names supplied as arguments to requests.

font is a typeface specified as a font identifier, an abstract style, or a mounting position.

ident is a valid *groff* identifier; its use often indicates that the operation creates an object of a type subsequently referred to as *mac, reg, str,* and so forth.

mac is a macro identifier.

message is an instance of *contents* to emit on the standard error stream. GNU *troff* strips a leading neutral double quote, allowing embedded leading spaces.

n is a numeric expression that evaluates to a non-negative integer.

±*N* is a numeric expression with a meaning dependent on its sign; see below.

name is a macro, string, or diversion identifier, or the name of a request.

npl is a numeric expression constituting a count of subsequent *productive* input lines; that is, those that directly produce formatted output. Text lines produce output, as do control lines containing requests like "**.tl //Page %//**" or escape sequences like "**\l'1i'**". Macro calls are not themselves productive, but their interpolations can be.

reg	is a register identifier.
str	is a string identifier.
stream	is an output stream identifier.

If a numeric expression presented as ±*N* starts with a '**+**' sign, an increment in the amount of of *N* is applied to the value applicable to the request or escape sequence. If it starts with a '**−**' sign, a decrement of magnitude *N* is applied instead. Without a sign, *N* replaces any existing value. A *roff* formatter always interprets a leading minus sign in *N* as a decrementation operator, not an algebraic sign. To assign a register a negative value or the negated value of another interpolation, you must force the formatter to interpret "**−**" as a negation or minus, rather than decrementation, operator: enclose the "**−**" with its operand in parentheses or subtract the expression of interest from zero. If a prior value does not exist—the register was undefined—an increment or decrement applies as if to 0.

Request short reference

Not all details of request behavior are outlined here. See the *groff* Texinfo manual or, for features new to GNU *troff*, *groff_diff* (7).

.ab Abort the formatter; exit with failure status.

.ab *terminal-message*
Abort the formatter; write *terminal-message* to the standard error stream and exit with failure status.

.ad Enable output line alignment and adjustment using the mode stored in **\n[.j]**.

.ad *c* Enable output line alignment and adjustment in mode *c* (*c*=**b**,**c**,**l**,**n**,**r**). Sets **\n[.j]**.

.af *reg c* Assign format *c* to register *reg,* where *c* is "**i**", "**I**", "**a**", "**A**", or a sequence of decimal digits whose quantity denotes the minimum width in digits to be used when the register is interpolated. "**i**" and "**a**" indicate Roman numerals and basic Latin alphabetics, respectively, in the lettercase specified. The default is **0**.

.aln *new-register existing-register*
Create alias (additional name) *new-register* of *existing-register*, causing the names to refer to the same stored object. If *existing-register* is undefined, the formatter ignores the request. GNU *troff* emits a warning in category "**reg**".

.als *new-name existing-name*
Create alias (additional name) *new-name* of request, string, macro, or diversion *existing-name*, causing the names to refer to the same stored object. If *existing-name* is undefined, the formatter ignores the request. GNU *troff* emits a warning in category "**mac**". If *new-name* already exists, its definition is lost unless already aliased.

.am *mac* Append to macro *mac* until encountering "**..**" at the start of a control line in the current conditional block.

.am *mac end-mac*
Append to macro *mac* until *end-mac* is called at the start of a control line in the current conditional block. *end-mac* can be a request.

.am1 *mac* As "**am**", with compatibility mode disabled when the appendment to macro *mac* is interpreted.

.am1 *mac end-mac*
As "**.am** *mac*", with compatibility mode disabled when the appendment to macro *mac* is interpreted.

.ami *str* Append to a macro indirectly—its name is in string *str*—until encountering " **..**".

.ami *str end-mac-str*
Append to a macro indirectly. As "**am**", but *str* and *end-mac-str* contain the names of the macro to be appended to, and that whose call ends the appendment, respectively.

.ami1 *str* As **ami**, with compatibility mode disabled when the appendment is interpreted.

.ami1 *str end-mac-str*
As **ami**, with compatibility mode disabled when the appendment is interpreted.

.as *ident* Create string *ident* with empty contents; no operation if *ident* already exists.

.as *str contents*
Append *contents* to string *str*.

.as1 *ident* As "**.as** *ident*".

.as1 *str contents*
As "**as**", with compatibility mode disabled when the appendment to string *str* is interpreted.

.asciify *div* Unformat ordinary characters, spaces, and some escape sequences in diversion *div*. When transforming a glyph node back into an input sequence that demands expression as a special character escape sequence, GNU *troff* uses the default escape character.

.backtrace Write the state of the input stack to the standard error stream. See the **−b** option of *groff* (1).

.bd *font* Stop emboldening font *font.*

.bd *font n* Embolden *font* by overstriking its glyphs offset by $n-1$ units. See register **.b**.

.bd *special-font font*
Stop emboldening *special-font* when *font* is selected. *special-font* must be a font name, not a mounting position.

.bd *special-font font n*
Embolden *special-font,* overstriking its glyphs offset by $n-1$ units when *font* is selected. See register **.b**.

.blm Unset blank line macro (trap). Restore default handling of blank lines.

.blm *mac* Set blank line macro (trap) to *mac*.

.box Stop directing output to current diversion; any pending output line is discarded.

.box *ident* Direct output to diversion *ident*, omitting a partially collected line.

.boxa Stop appending output to current diversion; any pending output line is discarded.

.boxa *div* Append output to diversion *div*, omitting a partially collected line.

.bp Break page and start a new one.

.bp $\pm N$ Break page, starting a new one numbered $\pm N$.

.br Break output line.

.brp Break and force adjustment of the output line per the current adjustment mode.

.break Break out of a "**while**" loop.

.c2 Reset no-break control character to "**'**".

.c2 *o* Recognize ordinary character *o* as no-break control character.

.cc Reset control character to '**.**'.

.cc *o* Recognize ordinary character *o* as the control character.

.ce Break, center the output of the next productive input line without filling, and break again.

.ce *npl* Break, center the output of the next *npl* productive input lines without filling, then break again. If $npl \leq 0$, stop centering.

.cf *file* Break and copy contents of *file* as "throughput" to GNU *troff* output (see *groff_out*(5)). Each line of *file* is output as if preceded by **\!**, but is not interpreted by the formatter. Unsafe request; disabled by default.

.cflags *n c. . .*
Assign properties encoded by non-negative integer *n* to each character or class *c*. Spaces need not separate *c* arguments.

.ch *mac* Unplant page location trap *mac*.

.ch *mac vertical-position*
Change page location trap *mac* planted by **wh** by moving its location to *vertical-position* (default scaling unit **v**).

.char *c* Define an ordinary, special, or indexed character *c* as empty.

.char *c contents*
Define an ordinary, special, or indexed character *c* as *contents.*

.chop *name*
Remove the last character from the macro, string, or diversion *name.*

.class *ident c . . .*
Define a (character) class *ident* comprising the characters or range expressions *c*, where each *c* is an ordinary, special, or indexed character; or a *range expression.* A class thus defined can

then be referred to in a **cflags** request in lieu of listing all the characters within it.

.close *stream*
Close *stream*, making it unavailable for " **write**" requests.

.color Enable output of color-related device-independent output commands. It is enabled by default.

.color *b* Enable or disable output of color-related device-independent output commands per Boolean expression *b*.

.composite *c*
Remove composite character mapping for character *c*.

.composite *c1 c2*
Map character *c1* to *c2* when *c1* is a combining component in a composite character.

.continue Skip the remainder of a "**while**" loop's body, immediately retesting its conditional expression.

.cp Enable AT&T *troff* compatibility mode. It is disabled by default.

.cp *b* Enable or disable AT&T *troff* compatibility mode per Boolean expression *b*.

.cs *f* Disable constant-width glyph spacing mode for font *f*.

.cs *f n* Enable constant-width glyph spacing mode for font *f* at *n*/36 ems.

.cs *f n p* Enable constant-width glyph spacing mode for *font* at *n*/36 ems, as if one em equals *p* scaled points.

.cu Continuously underline the output of the next productive input line.

.cu *npl* Continuously underline the output of the next *npl* productive input lines. If *npl*=0, stop continuously underlining.

.da Stop appending output to current diversion.

.da *div* Append output to diversion *div*.

.de *ident* Define macro *ident* until "**..**" occurs at the start of a control line in the current conditional block.

.de *ident end-mac*
Define macro *ident* until *end-mac* is called at the start of a control line in the current conditional block. *end-mac* can be a request.

.de1 *ident* As **de**, with compatibility mode disabled when *mac* is interpreted.

.de1 *ident end-mac*
As "**.de** *ident end-mac*", with compatibility mode disabled when *mac* is interpreted.

.defcolor *ident scheme color-component* ...
Define a color named *ident*. *scheme* identifies a color space and determines the number of required *color-component*s; it must be one of " **rgb**" (three components), "**cmy**" (three), "**cmyk**" (four), or "**gray**" (one). "**grey**" is accepted as a synonym of "**gray**". The color components can be encoded as a single hexadecimal value starting with **#** or **##**. The former indicates that each component is in the range 0–255 (0–FF), the latter the range 0–65,535 (0–FFFF). Alternatively, each color component can be specified as a decimal fraction in the range 0–1, interpreted using a default scaling unit of "**f**", which multiplies its value by 65,536 (but clamps it at 65,535).

.dei *str* Define macro indirectly. As **de**, but interpolate string *str* to obtain the macro's name.

.dei *str end-mac-str*
Define macro indirectly. As **de**, but *str* and *end-mac-str* contain the names of the macro to be defined, and that whose call ends the definition, respectively.

.dei1 *str* As **dei**, with compatibility mode disabled when the macro is interpreted.

.dei1 *str end-mac-str*
As **dei**, with compatibility mode disabled when the macro is interpreted.

.device *character-sequence*
Embed *character-sequence* into GNU *troff* output as parameters to an "x**X**" device extension command; see *groff_out*(5). The output driver or other postprocessor interprets *character-sequence* as it sees fit.

.devicem *name*
Write contents of macro or string *name* to *troff* output as the argument to a device extension command.

.di Stop directing output to current diversion.
.di *ident* Direct output to diversion *ident*.
.do *name argument* . . .
Interpret the string, request, diversion, or macro *name* (along with any further arguments) with compatibility mode disabled. Compatibility mode is restored (only if it was active) when the interpolation of *name* is interpreted.
.ds *ident* Create empty string named *ident*.
.ds *ident contents*
Create a string named *ident* containing *contents.*
.ds1 *ident*
.ds1 *ident contents*
As **ds**, with compatibility mode disabled when the string is interpreted.
.dt Clear diversion trap.
.dt *vertical-position mac*
Set the diversion trap to macro *mac* at *vertical-position* (default scaling unit **v**).
.ec Recognize **** as the escape character.
.ec *o* Recognize ordinary character *o* as the escape character.
.ecr Restore escape character saved with **ecs**.
.ecs Save the escape character.
.el Interpolate a newline if the conditional expression of the corresponding **ie** request was false.
.el *input* Interpret *input* as if it were an input line if the conditional expression of the corresponding **ie** request was false.
.em Unset end-of-input macro.
.em *mac* Call macro *mac* after the end of input.
.eo Disable the escape mechanism in interpretation mode.
.ev Pop environment stack, returning to previous one.
.ev *env* Push current environment onto stack and switch to *env*, creating it if necessary.
.evc *env* Copy environment *env* to the current one.
.ex Exit the formatter.
.fam Set default font family to previous value.
.fam *name* Set default font family to *name*.
.fc Disable field mechanism.
.fc *c* Set field delimiter to *c* and pad glyph to space.
.fc *c1 c2* Set field delimiter to *c1* and pad glyph to *c2*.
.fchar *c* Define fallback character *c* as empty.
.fchar *c contents*
Define fallback character *c* as *contents*. As **char**, but while that request hides a glyph with the same name in the selected font, **fchar** definitions are used only if the font lacks a glyph for *c*. GNU *troff* performs this test before searching special fonts.
.fcolor Restore previous fill color, or the default if there is none.
.fcolor *col* Select *col* as the fill color.
.fi Enable filling of output lines; a pending output line is broken. Sets **\n[.u]**.
.fl Flush any pending output line.
.fp *pos id* Mount font with font description file name *id* at non-negative position *pos*.
.fp *pos id font-description-file-name*
Mount font with *font-description-file-name* as name *id* at non-negative position *pos*.
.fschar *f c* Define fallback character *c* specific to font *f* as empty.
.fschar *f c contents*
Define fallback character *c* specific to font *f* as *contents*. As **char**, but GNU *troff* locates a character defined by **fschar** after any fonts named as arguments to the **fspecial** are searched and before those named as arguments to the "**special**" request.

.fspecial *f* Empty the list of fonts treated as special when *f* is selected.

.fspecial *f s* ...
Declare each font *s* as special only when font **f** is selected. GNU *troff* searches fonts specified as arguments to the "**special**" request after those given as arguments to the **fspecial** request.

.ft

.ft P Select the typeface *font*. If *font* is an integer, the formatter interprets it as a mounting position; the font mounted there is selected. If that position refers to an abstract style, GNU *troff* combines it with the default family (see **fam** above and **\F** below) to make a resolved font name. If *font* is "**DESC**", if the mounting position is not an abstract style and no font is mounted there, or the mounting position is negative, GNU *troff* ignores the request. (It also emits a warning in category "**font**", or "**range**", as appropriate. See section "Warnings" in *troff* (1).) Also see **\f .**

.ftr *f* Remove translation of font named *f*.

.ftr *f1 f2* Translate font name *f1* to *f2*.

.fzoom *font* **0**
Stop magnifying *font*.

.fzoom *font zoom*
Set magnification of mounted *font* to *zoom*, a multiplier of the current type size in thousandths (default: 1000).

.gcolor Restore previous stroke color, or the default if there is none.

.gcolor *col* Select *col* as the stroke color.

.hc Reset the hyphenation character to **\%** (the default).

.hc *c* Change the hyphenation character to *c*.

.hcode *dst1 src1* [*dst2 src2*] ...
Set the hyphenation code of character *dst1* to that of *src1*, and so on.

.hla Clear the environment's hyphenation language (disabling automatic hyphenation).

.hla *ident* Set the environment's hyphenation language to *ident*.

.hlm Set the consecutive automatically hyphenated line limit to **−1** (the default), meaning "no limit".

.hlm *n* Set the consecutive automatically hyphenated line limit to to *n*. A negative value means "no limit".

.hpf *file* Read hyphenation patterns from *file*.

.hpfa *file* Append hyphenation patterns from *file*.

.hpfcode *a b* [*c d*] ...
Caution: This request will be withdrawn in a future *groff* release. Use **hcode** instead.
Define mappings for character codes in hyphenation pattern files.

.hw *word* ...
Define each argument *word* (comprising ordinary, special, or indexed characters) as a *hyphenation exception word* such that each occurrence of a hyphen-minus "**−**" in *word* indicates a hyphenation point.

.hy Set automatic hyphenation mode to the value of the **.hydefault** register.

.hy 0 Disable automatic hyphenation; same as **.nh**.

.hy *mode* Set automatic hyphenation mode to *mode*; see section "Hyphenation" below.

.hydefault *mode*
Set hyphenation mode default to *mode*; see section "Hyphenation" below.

.hym Set the (right) hyphenation margin to **0** (the default).

.hym *length*
Set the (right) hyphenation margin to *length* (default scaling unit **m**).

.hys Set the hyphenation space to **0** (the default).

.hys *hyphenation-space*
Suppress automatic hyphenation in adjustment modes "**b**" or "**n**" if that adjustment can be achieved by adding no more than *hyphenation-space* to each inter-word space (default scaling unit **m**).

.ie *cond-expr*
: Interpolate a newline if *cond-expr* is true, otherwise skip to a corresponding **el** request.

.ie *cond-expr input*
: If *cond-expr* is true, interpret *input* as if it were an input line, otherwise skip to a corresponding **el** request.

.if *cond-expr*
: Interpolate a newline if *cond-expr* is true.

.if *cond-expr input*
: If *cond-expr* is true, then interpret *input* as if it were an input line.

.ig Ignore input (except for side effects of **\R** on auto-incrementing registers) until "**..**" occurs at the start of a control line in the current conditional block.

.ig *end-mac*
: Ignore input (except for side effects of **\R** on auto-incrementing registers) until *end-mac* is called at the start of a control line in the current conditional block. *end-mac* can be a request.

.in Set indentation amount to previous value.

.in ±*N* Set indentation to ±*N* (default scaling unit **m**).

.it Cancel any pending input line trap.

.it *npl mac* Set (or replace) an input line trap in the environment, calling *mac* after the next *npl* productive input lines have been read. Lines interrupted with the **\c** escape sequence are counted separately.

.itc Cancel any pending input line trap.

.itc *npl mac*
: As "**it**", but lines interrupted with the **\c** escape sequence do not apply to the line count.

.kern Enable pairwise kerning.

.kern *b* Enable or disable pairwise kerning per Boolean expression *b*.

.lc Unset leader repetition character.

.lc *c* Set leader repetition character to *c* (default: "**.**").

.length *reg contents*
: Compute the number of characters in *contents* and store the count in the register *reg*.

.linetabs Activate line-tabs in the environment. It is disabled by default.

.linetabs *b* Activate or deactivate line-tabs in the environment per Boolean expression *b*.

.lf *input-line-number*
: Set the input line number the formatter uses when reporting diagnostics. The argument becomes the input line number of the *next* line the formatter reads.

.lf *input-line-number character-sequence*
: As **lf** with one argument, but also update the reported file name to *character-sequence*.

.lg Enable ligature mode **1**.

.lg *m* Set ligature mode to *m* (**0** = disable, **1** = enable, **2** = enable for two-letter ligatures only).

.ll Set line length to previous value. Does not affect a pending output line.

.ll ±*N* Set line length to ±*N* (default length 6.5**i**, default scaling unit **m**). Does not affect a pending output line.

.lsm Unset the leading space macro (trap). Restore default handling of lines with leading spaces.

.lsm *mac* Set the leading space macro (trap) to *mac*.

.ls Change to the previous value of additional intra-line skip.

.ls *n* Set additional intra-line skip value to *n*, i.e., *n*−1 blank lines are inserted after each text output line.

.lt Set length of title lines to previous value.

.lt ±*N* Set length of title lines (default length 6.5**i**, default scaling unit **m**).

.mc Cease writing margin character.

.mc *c* Begin writing margin character *c* to the right of each output line at a distance of 10**p**.

.mc *c d* Begin writing margin character *c* on each output line at distance *d* to the right of the right margin (default distance 10**p**, default scaling unit **m**).

.mk Mark vertical drawing position in an internal register; see **.rt**.

.mk *reg* Mark vertical drawing position in register *reg*.

.mso *file* As "**so**", except that GNU *troff* searches for the specified *file* in the same directories as macro files; see *GROFF_TMAC_PATH* in section "Environment" of *groff* (1) and **−m** in section "Options" of the same page.

.msoquiet *file*
As **mso**, but no warning is emitted if *file* does not exist.

.na Disable output line adjustment.

.ne Break page if distance to next page location trap is less than one vee.

.ne *d* Break page if distance to next page location trap is less than distance *d* (default scaling unit **v**).

.nf Disable filling of output lines; a pending output line is broken. Clears **\n[.u]**.

.nh Disable automatic hyphenation; same as "**.hy 0**".

.nm Deactivate output line numbering.

.nm ±*N*
.nm ±*N m*
.nm ±*N m s*
.nm ±*N m s i*
Activate output line numbering: number the next output line ±*N,* writing numbers every *m* lines (default 1), with *s* numeral widths (**\0**) between the line number and the output (default 1), and indenting the line number by *i* numeral widths (default 0).

.nn Suppress numbering of the next output line counted by **nm**.

.nn *n* Suppress numbering of the next *n* output lines counted by **nm**. If *n*=0, cancel suppression.

.nop Interpolate a newline.

.nop *input* Interpret *input* as if it were an input line.

.nr *reg* ±*N* Define or update register *reg* with value *N*.

.nr *reg* ±*N I*
Define or update register *reg* with value *N* and auto-increment *I*, which may be any integer.

.nroff Make the conditional expressions **n** true and **t** false.

.ns Enable *no-space mode*, ignoring **.sp** requests until a glyph or **\D** primitive is output. See **.rs**.

.nx Stop processing the input file and read the next, if any.

.nx *file* Stop processing the input file and read *file*.

.open *ident file*
Open *file* for writing and associate a stream named *ident* with it, making it available for "**write**" requests. Unsafe request; disabled by default.

.opena *ident file*
As "**open**", but if *file* already exists, appends to it instead of overwriting it. Unsafe request; disabled by default.

.os Output vertical space that was saved by the **sv** request.

.output *character-sequence*
Emit *character-sequence* "transparently" (directly) to *troff*'s output.

.pc Reset page number character to '**%**'.

.pc *c* Change the page number character used in titles to *c*.

.pchar *c* . . .
Report, to the standard error stream, information about each character (be it ordinary, special, or indexed) or character class *c*. A character defined by a request (**char**, **fchar**, **fschar**, or **schar**) reports its contents as a JSON-encoded string, but the output is not otherwise in JSON format.

.pcolor Report, to the standard error stream, each defined color name, its color space identifier, and channel value assignments. A device's default stroke and/or fill colors, "default", are not listed since they are immutable and their details unknown to the formatter.

.pcolor *col* . . .
Report, to the standard error stream, the name, color space identifier, and channel value assignments of each color *col*.

.pcomposite
: Report, to the standard error stream, the list of configured composite character mappings. See the "**composite**" request. The "from" code point is listed first, followed by its "to" mapping.

.pev
: Report the state of the current environment followed by that of all other environments to the standard error stream.

.pfp
: Report, to the standard error stream, the list of occupied font mounting positions. Occupied mounting positions are listed, one per line, in increasing order, followed by the typeface name; if the name corresponds to an abstract style, the entry ends there. Otherwise, the name of the font description file and the font's "internal name" datum, the meaning of which varies by output device, follow.

.pftr
: Report, to the standard error stream, the list of font translations.

.phw
: Report, to the standard error stream, the list of hyphenation exception words associated with the hyphenation language selected by the **hla** request. A "**–**" marks each hyphenation point. A word prefixed with "**–**" is not hyphenated at all. The report suffixes words to which automatic hyphenation applies (meaning those defined in a hyphenation pattern file rather than with the **hw** request) with a tab and asterisk "*****".

.pi *command*
: Pipe GNU *troff* output through *command* (which is read in copy mode) by passing it to *popen*(3). Multiple **pi** requests construct a multi-stage pipeline in the same order as the formatter encounters the requests. Unsafe request; disabled by default.

.pl
: Set page length to default 11**i**. The current page length is stored in register **.p**.

.pl ±*N*
: Change page length to ±*N* (default scaling unit **v**).

.pline
: Report, in JSON syntax to the standard error stream, the list of output nodes corresponding to the pending output line. In JSON, a pair of empty brackets "**[ ]**" represents an empty list. A *pending* output line has not yet undergone adjustment, and lacks a line number and margin character (all as applicable).

.pm
: Report, to the standard error stream, the names of all defined macros, strings, and diversions and their lengths in characters or nodes.

.pm *name* ...
: Report, to the standard error stream, the JSON-encoded name and contents of each macro, string, or diversion *name*.

.pn ±*N*
: Set next page number.

.pnr
: Report the names, values, and (as applicable) autoincrement amounts and assigned formats of all defined registers to the standard error stream.

.pnr *reg* ...
: Report the name and value and, if its type is numeric, the autoincrement amount and assigned format, of each register *reg* to the standard error stream.

.po
: Change to previous page offset. The current page offset is available in register **.o**.

.po ±*N*
: Alter page offset (default scaling unit **m**).

.ps
: Restore previous type size.

.ps ±*N*
: Set/increase/decrease the type size to/by *N* scaled points (a non-positive resulting type size is set to 1 u); also see **\s[**±*N***]**.

.psbb *postscript-file*
: Retrieve the bounding box of the PostScript image found in *postscript-file,* which must conform to Adobe's Document Structuring Conventions (DSC). See registers **llx**, **lly**, **urx**, **ury**.

.pso *command*
: As "**so**", except that input comes from the standard output stream of *command*, which is passed to *popen*(3). Unsafe request; disabled by default.

.pstream
: Report, in JSON syntax to the standard error stream, the list of open streams, including the name of each open stream, the name of the file backing it, and its mode (writing or appending).

.pwh
: Report names and positions of all page location traps to the standard error stream.

.pvs Change to previous post-vertical line spacing.

.pvs ±*N* Change post-vertical line spacing according to ±*N* (default scaling unit **p**).

.rchar *c*... Remove definition of each ordinary, special, or indexed character *c*, undoing the effect of a **char**, **fchar**, or **schar** request. Spaces need not separate *c* arguments. The character definition removed (if any) is the first encountered in the resolution process documented in section "Using Symbols" of *Groff: The GNU Implementation of troff*. Glyphs, which are defined by font description files, cannot be removed.

.rd Read insertion from standard input stream, ringing terminal bell as prompt.

.rd *terminal-message*
Read insertion from standard input stream, writing *terminal-message* to standard error stream as prompt.

.return Stop interpreting an interpolated macro, skipping the remainder of its definition.

.return *input*
As **return**, but perform the skip twice—once within the macro being interpreted and once in an enclosing macro.

.rfschar *f c*...
Remove each fallback special character *c* for font *f*. Spaces need not separate *c* arguments. See **fschar**.

.rj Break, right-align the output of the next productive input line without filling, then break again.

.rj *npl* Break, right-align the output of the next *npl* productive input lines without filling, then break again. If *npl* ≤ 0, stop right-aligning.

.rm *name* ...
Remove each request, macro, diversion, or string *name*.

.rn *old new*
Rename request, macro, diversion, or string *old* to *new*.

.rnn *reg1 reg2*
Rename register *reg1* to *reg2*.

.rr *reg* ... Remove each register *reg*.

.rs Restore spacing; disable no-space mode. See **.ns**.

.rt Return *(upward only)* to vertical position marked by **.mk** on the current page.

.rt *N* Return *(upward only)* to vertical position *N* (default scaling unit **v**).

.schar *c* Define global fallback character *c* as empty.

.schar *c contents*
Define global fallback character *c* as *contents*. As **char**, but GNU *troff* locates a character defined with **schar** after any fonts named as arguments to the "**special**" request and before any mounted special fonts.

.shc Reset the soft hyphen character to **\[hy]**.

.shc *c* Set the soft hyphen character to *c*.

.shift Shift macro or string parameters one place *n* places: argument *i* becomes argument *i* − *1*; argument 1 becomes unavailable.

.shift *n* Shift macro or string parameters *n* places: argument *i* becomes argument *i* − *n*; arguments 1 to *n* become unavailable.

.sizes *s1 s2* ... *sn* [**0**]
Set available type sizes to the list of values or ranges; each *s*i is interpreted in units of scaled points (**s**). GNU *troff* strips a leading neutral double quote from *s1*; it reads each *s*i in copy mode.

.so *file* Replace the request's control line with the contents of *file*, "sourcing" it. GNU *troff* searches for *file* in any directories specified by **−I** command-line options, followed by the current working directory. If *file* does not exist, the formatter ignores the request. (GNU *troff* emits a warning in category "**file**".)

.soquiet *file*
As "**so**", but no warning is emitted if *file* does not exist.

.sp Break and move the next text baseline down by one vee, or until springing a page location trap. If invoked with the no-break control character, **sp** moves the text baseline applicable to the entire pending output line.

.sp *vertical-distance*
As **sp**, but move by *vertical-distance* instead of **1v**. Inside a diversion, the formatter ignores any argument. A negative *vertical-distance* cannot reduce the position of the text baseline below zero. Applying the boundary-relative measurement operator | operator to *vertical-distance,* as in "|*N*", moves to a position relative to the page top for positive *N*, and the bottom if *N* is negative.

.special Empty the list of special fonts designated by this request when given arguments.

.special *s* ...
Declare each font *s* as special, irrespective of its description file, populating a list that GNU *troff* searches, in order, to find the glyph demanded. GNU *troff* searches fonts specified as arguments to the "**special**" request after those given as arguments to the **fspecial** request.

.spreadwarn
Toggle the spread warning on and off (the default) without changing its value.

.spreadwarn *N*
Emit a **break** warning if the additional space inserted for each space between words in an adjusted output line is greater than or equal to *N*. A negative *N* is treated as 0. The default scaling unit is **m**. At startup, **spreadwarn** is inactive and *N* is 3 m**.**

.ss *n* Set minimum inter-word space and supplemental inter-sentence space sizes to *n* 12ths of the selected font's **spacewidth** parameter (default: 12).

.ss *n m* As "**.ss** *n*", but set supplemental inter-sentence space size to *m* 12ths of the selected font's **spacewidth** parameter.

.stringdown *str*
Replace each byte in the string named *str* with its lowercase version.

.stringup *str*
Replace each byte in the string named *str* with its uppercase version.

.sty *pos style*
Associate abstract *style* with non-negative font position *pos*.

.substring *str start* [*end*]
Replace the string named *str* with its substring bounded by the indices *start* and *end*, inclusive. Negative indices count backward from the end of the string.

.sv As **ne**, but save 1 v for output with **os** request.

.sv *d* As **ne**, but save distance *d* for later output with **os** request (default scaling unit **v**).

.sy *command*
Execute *command* (which is read in copy mode) in the operating environment by passing it to *system*(3). See register **systat**. Unsafe request; disabled by default.

.ta Clear tab stops.

.ta *n1 n2* ... *n*n **T** *r1 r2* ... *r*n
Set tabs at positions *n1*, *n2*, ..., *n*n, then set tabs at *n*n+*m*×*r*n+*r1* through *n*n+*m*×*r*n+*r*n, where *m* increments from 0, 1, 2, ... to the output line length. Each *n* argument can be prefixed with a "**+**" to place the tab stop *ni* at a distance relative to the previous, *n*(*i*−1). Each argument *ni* or *ri* can be suffixed with a letter to align text within the tab column bounded by tab stops *i* and *i*+1; "**L**" for left-aligned (the default), "**C**" for centered, and "**R**" for right-aligned.

.tag Reserved for internal use.

.taga Reserved for internal use.

.tc Unset tab repetition character.

.tc *c* Set tab repetition character to *c* (default: none).

.ti ±*N* Temporarily indent next output line (default scaling unit **m**).

.tkf *font s1 n1 s2 n2*
Enable track kerning for *font*.

.tl '*left*'*center*'*right*'
: Format three-part title.

.tm
: Write a newline to the standard error stream.

.tm *terminal-message*
: Send *terminal-message*, followed by a newline, to the standard error stream. The formatter reads the argument to the end of the input line in copy mode, but does *not* remove a leading double quote; contrast **.tm1**.

.tm1
: As **tm**.

.tm1 *message*
: As **tm**, but removes a leading neutral double quote '"' from *message*, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode.

.tmc
: No operation.

.tmc *message*
: As **tm1**, but does not append a newline.

.tr *abcd...*
: Translate ordinary or special characters *a* to *b*, *c* to *d*, and so on prior to output.

.trf *file*
: Break and copy contents of *file* as "throughput" to GNU *troff* output (see *groff_out*(5)). Unlike **cf**, characters invalid as input to GNU *troff* are discarded. If *file*'s contents do not end with a newline, **trf** appends one.

.trin *abcd...*
: As **tr**, except that **asciify** ignores the translation when a diversion is interpolated.

.trnt *abcd...*
: As **tr**, except that translations are suppressed in the argument to **\!**.

.troff
: Make the conditional expressions **t** true and **n** false.

.uf *font*
: Set underline font used by **ul** to *font*.

.ul
: Underline (italicize in *troff* mode) the output of the next productive input line.

.ul *npl*
: Underline (italicize in *troff* mode) the output of the next *npl* productive input line. If *npl*=0, stop underlining.

.unformat *div*
: Unformat space characters and tabs in diversion *div*, preserving font information.

.vpt
: Enable vertical position traps. They are enabled by default.

.vpt *b*
: Enable or disable vertical position traps per Boolean expression *b*.

.vs
: Change to previous vertical spacing.

.vs ±*N*
: Set vertical spacing to ±*N* (default scaling unit **p**).

.warn
: Enable all warning categories.

.warn 0
: Disable all warning categories.

.warn *n*
: Enable warnings in categories whose codes sum to *n*; see *troff*(1).

.warnscale *scaling-unit*
: Select scaling unit used in certain warnings (one of **u**, **i**, **c**, **p**, or **P**; default: **i**). Ignored on *nroff*-mode output devices, for which these diagnostics report the vertical page location in lines, and the horizontal page location in ens.

.wh *vertical-position*
: Remove visible page location trap at *vertical-position* (default scaling unit **v**).

.wh *vertical-position mac*
: Plant macro *mac* as page location trap at *vertical-position* (default scaling unit **v**), removing any visible trap already there.

.while *cond-expr*
: Interpolate newlines unless and until *cond-expr* evaluates false.

.while *cond-expr input*
: Repeatedly execute *input* unless and until *cond-expr* evaluates false.

.write *stream character-sequence*
: Write *character-sequence*, a sequence of ordinary characters, spaces, or tabs, to *stream*, which must previously have been the subject of an "**open**" (or **opena**) request, followed by a newline. GNU *troff* flushes the stream after writing to it.

.writec *stream character-sequence*
: As "**write**", but does not write a newline to the output.

.writem *stream name*
: Write the contents of the macro or string *name* to *stream*, which must previously have been the subject of an "**open**" (or **opena**) request. GNU *troff* reads the contents of *name* in copy mode.

Escape sequence short reference

The escape sequences **\"**, **\#**, **\$**, *****, **\?**, **\a**, **\e**, **\n**, **\t**, **\g**, **\V**, and *****newline* are interpreted even in copy mode.

The escape sequences **\f**, **\F**, **\H**, **\m**, **\M**, **\R**, **\s**, and **\S** are not tokenized when GNU *troff* reads its input. **\R** updates only the formatter's register dictionary, and does not contribute (directly) to output. The others alter the environment (see section "Environments" below). See the "GNU troff Internals" section of the *groff* Texinfo manual.

\" Comment; read everything up to the next newline in copy mode and discard it.

\# Whole-line comment; read everything up to and including the next newline in copy mode and discard it.

*****s Interpolate string with one-character name s.

*****(*st* Interpolate string with two-character name st.

*****[*string*]
: Interpolate string with name *string* (of arbitrary length).

*****[*string arg* ...]
: Interpolate string with name *string* (of arbitrary length), taking *arg* ... as arguments.

\$0 Interpolate name by which currently executing macro was invoked.

\$$n$ Interpolate macro or string parameter numbered n ($1 \le n \le 9$).

\$(*nn* Interpolate macro or string parameter numbered nn ($01 \le nn \le 99$).

\$[*nnn*]
: Interpolate macro or string parameter numbered nnn ($nnn \ge 1$).

\$* Interpolate catenation of all macro or string parameters, separated by spaces.

\$@ Interpolate catenation of all macro or string parameters, with each surrounded by double quotes and separated by spaces.

\$^ Interpolate catenation of all macro or string parameters as if they were arguments to the **ds** request.

\' is a synonym for **\[aa]**, the acute accent special character.

\` is a synonym for **\[ga]**, the grave accent special character.

\– is a synonym for **\[–]**, the minus sign special character.

_ is a synonym for **\[ul]**, the underrule special character.

\% Mark a hyphenation point within a word. At the beginning of a word, prevent it from being otherwise hyphenated.

\!*character-sequence*
: *Transparently* embed *character-sequence*, up to and including the end of the line, into the current diversion, requests, macro calls, and escape sequences when reading them into a diversion. Doing so prevents them from taking effect until the diverted text is actually output.

\?*character-sequence***\?**
: Transparently embed *character-sequence*, up to its own next occurrence on the input line, into a diversion, or unformatted as an output comparand in a conditional expression. Ignored in the top-level diversion.

*****space Move right one inter-word space.

\~ Insert an unbreakable, adjustable space.

\0 Move right by the width of a numeral in the current font.

\| Move one-sixth em to the right on typesetters.

\^ Move one-twelfth em to the right on typesetters.

\& Interpolate a dummy character.

\) Interpolate a dummy character that is transparent to end-of-sentence recognition.

\/ Apply italic correction. Use between an immediately adjacent oblique glyph on the left and an upright glyph on the right.

\, Apply left italic correction. Use between an immediately adjacent upright glyph on the left and an oblique glyph on the right.

\: Non-printing break point (similar to **\%**, but never produces a hyphen glyph).

*****newline*
Continue current input line on the next.

\{ Begin conditional input.

\} End conditional input.

\(*sc* Interpolate glyph of special character with two-character identifier *sc*.

\[*spchar***]**
Interpolate glyph of special character with identifier *spchar* (of arbitrary length).

\[*base-char comp* . . .**]**
Interpolate composite glyph constructed from *base-char* and each component *comp*.

\[char*nnn***]**
Interpolate glyph of eight-bit encoded character *nnn*, where $0 \le nnn \le 255$.

\[u*nnnn*[*n*[*n*]]**]**
Interpolate glyph of Unicode character with code point *nnnn*[*n*[*n*]] in uppercase hexadecimal.

\[u*base-char*[*_combining-component*]. . .**]**
Interpolate composite glyph from Unicode character *base-char* and *combining-components*.

\a Interpolate a leader in copy mode.

\A'*input***'**
Interpolate 1 if *input* is a valid identifier, and 0 otherwise. Because GNU *troff* ignores any input character with an invalid code when reading it, invalid identifiers are empty or contain spaces, tabs, newlines, or escape sequences that interpolate something other than a sequence of ordinary characters.

\b'*string***'**
Build bracket: pile a sequence of glyphs corresponding to each character in *string* vertically, and center it vertically on the output line.

\B'*input***'**
Interpolate 1 if *input* is a valid numeric expression, and 0 otherwise.

\c Continue output line at next input line.

\C'*glyph***'**
As **\[***glyph***]**, but compatible with other *troff* implementations.

\d Move downward ½ em on typesetters.

\D'*drawing-command***'**
See subsection "Drawing commands" below.

\e Interpolate the escape character.

\E As **\e**, but not interpreted in copy mode.

\fP Select previous font mounting position (abstract style or font); same as "**.ft**" or "**.ft P**".

\f*F* Select font mounting position, abstract style, or font with one-character name or one-digit position *F*. *F* cannot be **P**.

\f(*ft* Select font mounting position, abstract style, or font with two-character name or two-digit position *ft*.

\f[*font***]**
Select font mounting position, abstract style, or font with arbitrarily long name or position *font*. *font* cannot be **P**.

\f[] Select previous font mounting position (abstract style or font).

\F*f* Set default font family to that with one-character name *f*.

\F(*fm* Set default font family to that with two-character name *fm*.

\F[*fam***]**
Set default font family to that with arbitrarily long name *fam*.

\F[] Set default font family to previous value.

\g*r* Interpolate format of register with one-character name *r*.

\g(*rg* Interpolate format of register with two-character name *rg*.

\g[*reg***]** Interpolate format of register with arbitrarily long name *reg*.

\h'*N***'** Horizontally move the drawing position by *N* ems (or specified units); | may be used. Positive motion is rightward.

\H'*N***'** Set (increment, decrement) the height of the current font, but not its width. If *N* is zero, the formatter uses the font's inherent height for its type size default scaling unit **z**).

\k*r* Store the horizontal drawing position, relative to that corresponding to the start of the input line (ignoring page offset and indentation), in register name *r*.

\k(*rg* Store the horizontal drawing position, relative to that corresponding to the start of the input line (ignoring page offset and indentation), in register name *rg*.

\k[*reg***]** Store the horizontal drawing position, relative to that corresponding to the start of the input line (ignoring page offset and indentation), in register name *reg*.

\l'*N*[*c*]**'**
Draw horizontal line of length *N* with character *c* (default: **\[ru]**; default scaling unit **m**).

\L'*N*[*c*]**'**
Draw vertical line of length *N* with character *c* (default: **\[br]**; default scaling unit **v**).

\m*c* Select stroke color with one-character name *c*.

\m(*cl* Select stroke color with two-character name *cl*.

\m[*color***]**
Select stroke color with arbitrarily long name *color*.

\m[] Restore previous stroke color.

\M*c* Select fill color with one-character name *c*.

\M(*cl* Select fill color with two-character name *cl*.

\M[*color***]**
Select fill color with arbitrarily long name *color*.

\M[] Restore previous fill color.

\n*r* Interpolate value of register with one-character name *r*.

\n(*rg* Interpolate value of register with two-character name *rg*.

\n[*reg***]** Interpolate value of register with arbitrarily long name *reg*.

\N'*n***'** Format indexed character numbered *n* in the current font.

\o'*abc*. . .**'**
Overstrike centered glyphs of characters *a*, *b*, *c*, and so on.

\O0 At the outermost suppression level, disable emission of glyphs and geometric objects to the output driver.

\O1 At the outermost suppression level, enable emission of glyphs and geometric objects to the output driver.

\O2 At the outermost suppression level, enable glyph and geometric primitive emission to the output driver and write to the standard error stream the page number, four bounding box registers enclosing glyphs written since the previous **\O** escape sequence, the page offset, line length, image file name (if any), horizontal and vertical device motion quanta, and input file name.

\O3 Begin a nested suppression level.

\O4 End a nested suppression level.

\O[5*P file***]**
At the outermost suppression level, write the name *file* to the standard error stream at position *P*, which must be one of **l**, **r**, **c**, or **i**.

\p Break output line at next word boundary; adjust if applicable.

\r Move "in reverse" (upward) 1 em.

\R'*name* ±*N***'**
Set, increment, or decrement register *name* by *N*.

\s±*N* Set/increase/decrease the type size to/by *N* scaled points. *N* must be a single digit. If *n* is an unsigned "0", restore the previous size. (In compatibility mode only, a non-zero *N* must be in the range 4–39.) Otherwise, as **ps** request.

\s(±*N*

\s±**(***N* Set/increase/decrease the type size to/by *N* scaled points; *N* is a two-digit number ≥1. If *n* is an unsigned "00", restore the previous size. Otherwise, as **ps** request.

\s[±*N***]**

\s±**[***N***]**

\s'±*N***'**

\s±**'***N***'** Set/increase/decrease the type size to/by *N* scaled points. If *n* is an unsigned "0" (with any number of leading zeroes), restore the previous size. Otherwise, as **ps** request.

\S'*N***'** Slant the glyphs of the currently selected font by @var{N} degrees. Positive values slant in the direction of text flow. Only integer values are possible.

\t Interpolate a tab in copy mode.

\u Move upward ½ em on typesetters.

\v'*N***'** Vertically move the drawing position by *N* vees (or specified units); | may be used. Positive motion is downward.

\V*e* Interpolate value of system environment variable with one-character name *e* as returned by *getenv*(3).

\V(*ev* Interpolate value of system environment variable with two-character name *ev* as returned by *getenv*(3).

\V[*env***]**

Interpolate value of system environment variable with arbitrarily long name *env* as returned by *getenv*(3).

\w'*input***'**

Interpolate the width of *input*, as interpreted, in basic units. This escape sequence allows several properties of formatted output to be measured without writing it out. The formatter processes *input* in a dummy environment: this means that font and type size changes, for example, may occur within it without affecting subsequent output.

\x'*N***'** Increase vertical spacing of pending output line by *N* vees (or specified units; negative before, positive after).

\X'*character-sequence***'**

Write *character-sequence* to *troff* output as a device extension command. The *groff* special character repertoire is unknown to output drivers outside of glyphs named in a device's fonts, and even then they may not possess complete coverage of the names documented in *groff_char*(7). Further, escape sequences that produce horizontal or vertical motions, hyphenation breaks, or that are dummy characters may appear in strings or be converted to nodes, particularly in diversions. These are not representable when interpolated directly into device-independent output, as might be done when writing out tag names for PDF bookmarks, which can appear in a viewer's navigation pane.

So that documents or macro packages do not have to laboriously "sanitize" strings destined for interpolation in device extension commands, this escape sequence performs certain transformations on its argument. For these transformations, character translations and definitions are ignored.

GNU *troff* converts several ordinary characters that typeset as non-basic Latin code points to code points outside that range so that they are used consistently whether they are formatted as glyphs or used in a device control command argument. These ordinary characters are "**'**", "**-**", "**^**", "**`**", and "**~**"; others are written as-is.

Special characters that typeset as Unicode basic Latin characters are translated to basic Latin characters accordingly. So that any Unicode code point can be represented in device extension commands, for example in an author's name in document metadata or as a usefully named bookmark or hyperlink anchor, GNU *troff* maps other special characters to Unicode special character notation. Special characters without a Unicode representation, and escape sequences that do not interpolate a sequence of ordinary and/or special characters, produce warnings in category "**char**".

\Y*n* Write contents of macro or string *n* to *troff* output as a device extension command.
\Y(*nm* Write contents of macro or string *nm* to *troff* output as a device extension command.
\Y[*name***]**
Write contents of macro or string *name* to *troff* output as a device extension command.
\z*c* Format character *c* with zero width—without advancing the drawing position.
\Z'*input***'**
Save the drawing position, format *input*, then restore it. GNU *troff* ignores tabs and leaders in *input* with an error diagnostic.

Drawing commands

Drawing commands direct the output device to render geometrical objects rather than glyphs. Specific devices may support only a subset, or may feature additional ones; consult the man page for the output driver in use. Terminals in particular implement almost none.

Rendering starts at the drawing position; when finished, the drawing position is left at the rightmost point of the object, even for closed figures, except where noted. GNU *troff* draws stroked (outlined) objects with the stroke color, and shades filled ones with the fill color. See section "Colors" above. Coordinates *h* and *v* are horizontal and vertical motions relative to the drawing position or previous point in the command. The default scaling unit for horizontal measurements (and diameters of circles) is **m**; for vertical ones, **v**.

Circles, ellipses, and polygons can be drawn stroked or filled. These are independent properties; if you want a filled, stroked figure, you must draw the same figure twice using each drawing command. A filled figure is always smaller than an outlined one because the former is drawn only within its defined area, whereas strokes have a line thickness (set with **\D't'**).

\D'~ *h1 v1 . . . hn vn***'**
Draw B-spline to each point in sequence, leaving drawing position at (*hn*, *vn*).
\D'a *hc vc h v***'**
Draw circular arc centered at (*hc*, *vc*) counterclockwise from the drawing position to a point (*h*, *v*) relative to the center. (*hc*, *vc*) is adjusted to the point nearest the perpendicular bisector of the arc's chord.
\D'c *d***'**
Draw circle of diameter *d* with its leftmost point at the drawing position.
\D'C *d***'**
As **\D'C'**, but the circle is filled.
\D'e *h v***'**
Draw ellipse of width *h* and height *v* with its leftmost point at the drawing position.
\D'E *h v***'**
As **\D'e'**, but the ellipse is filled.
\D'l *h v***'**
Draw line from the drawing position to (*h*, *v*).
\D'p *h1 v1 . . . hn vn***'**
Draw polygon with vertices at the drawing position and each point in sequence. GNU *troff* closes the polygon by drawing a line from (*hn*, *vn*) back to the initial drawing position. Afterward, the drawing position is left at (*hn*, *vn*).
\D'P *h1 v1 . . . hn vn***'**
As **\D'p'**, but the polygon is filled.
\D't *n***'**
Set stroke thickness of geometric objects to to *n* basic units. A zero *n* selects the minimum supported thickness. A negative *n* selects a thickness proportional to the type size; this is the default.

Strings

groff supports strings primarily for user convenience. Conventionally, if one would define a macro only to interpolate a small amount of text, without invoking requests or calling any other macros, one defines a string instead. Only one string is predefined by the language.

*[**.T**] Contains the name of the output device (for example, "**utf8**" or "**pdf**").

The **ds** request creates a string with a specified name and contents. If the identifier named by **ds** already exists as an alias, the target of the alias is redefined. If **ds** is invoked with only one argument, the named string becomes empty. The formatter removes a leading neutral double quote '"' from *contents* to permit the embedding of leading spaces. It interprets any other '"' literally, but the wise author uses the special character escape sequence **\[dq]** instead if the string might be interpolated as part of a macro argument; see section "Calling macros" above.

The ***** escape sequence dereferences a string's name, interpolating its contents. If the name does not exist, the formatter defines it as empty, interpolates nothing, and emits a warning in category "**mac**". See section "Warnings" in *troff* (1). The bracketed interpolation form accepts arguments that are handled as macro arguments are; see section "Calling macros" above. In contrast to macro calls, an argument containing a closing bracket **]** must be enclosed in double quotes. When defining strings, argument interpolations must be escaped if they are to reference parameters from the calling context; see section "Parameters" below. Any non-initial neutral double quote **"** is interpreted literally, but it is wise to use the special character escape sequence **\[dq]** instead if the string might be interpolated as part of a macro argument; see section "Calling macros" above. Strings are not limited to a single input line of text. *****_newline_ works just as it does elsewhere. The resulting string is stored *without* the newlines. When filling is disabled, care is required to avoid overrunning the line length when interpolating strings. Conversely, when filling is enabled, it is not necessary to append **\c** to a string interpolation to prevent a break afterward, as might be required in a macro argument. Nor does a string require use of the GNU *troff* **chop** request to excise a trailing newline as is often done with diversions. It is not possible to embed a newline in a string that will be interpreted as such when the string is interpolated. To achieve that effect, use ***** to interpolate a macro instead; see section "Punning names" below.

The "**as**" request is similar to **ds** but appends to a string instead of redefining it. If "**as**" is invoked with only one argument, no operation is performed (beyond dereferencing the string).

Because strings are similar to macros, they too can be defined to suppress AT&T *troff* compatibility mode enablement when interpolated; see section "Compatibility mode" below. The **ds1** request defines a string that suspends compatibility mode when the string is later interpolated. **as1** is likewise similar to "**as**", with compatibility mode suspended when the appended portion of the string is later interpolated.

Caution: These requests treat the remainder of the input line as their second argument, including any spaces, up to a newline or comment escape sequence. Ending string definitions (and appendments) with a comment, even an empty one, prevents unwanted space from creeping into them during source document maintenance. This rule and suggestion also applies to the second argument of the "**length**" request, to requests that define characters (**char**, **fchar**, **fschar**, and **schar**), and to the file name or command arguments of the **cf**, **hpf**, **hpfa**, **mso**, **msoquiet**, **nx**, "**open**", **opena**, **pi**, **pso**, "**so**", **soquiet**, and **trf** requests.

Several requests exist to perform rudimentary string operations. Strings can be queried (**length**) and modified (**chop**, **substring**, **stringup**, **stringdown**), and their names can be manipulated through renaming, removal, and aliasing (**rn**, **rm**, **als**).

Redefinitions and appendments "write through" request, macro, string, and diversion names. To replace an aliased object with a separately defined one, you must use the **rm** request on its name first.

Registers

In the *roff* language, numbers and measurements can be stored in *registers*. Many built-in registers exist, supplying anything from components of the date to details of formatting parameters; some of these are read-only. You can also define your own. See section "Identifiers" above regarding the construction of valid names for registers.

Each register (except read-only ones) can be assigned a *format*, causing its value to interpolate with leading zeroes, in Roman numerals, or alphabetically. Some read-only registers are string-valued, meaning that they interpolate text and lack a format.

Define registers and update their values with the **nr** request or the **\R** escape sequence.

User-defined registers can also be incremented or decremented by a configured amount at the time they are

interpolated. The value of the increment is specified with a third argument to the **nr** request, and a special interpolation syntax, **\n±**, alters and then retrieves the register's value. Together, these features are called *auto-increment*. (A negative auto-increment can be considered an "auto-decrement".)

Many predefined registers are available. The following presentation uses register interpolation syntax **\n[***name***]** to refer to a register *name* to clearly distinguish it from a string or request *name*; the symbols **\n[]** are *not* part of the register name. The register name space is separate from that used for requests, macros, strings, and diversions.

Read-only registers

Predefined registers whose identifiers start with a dot are read-only. Many are Boolean-valued.

Caution: Built-in registers are subject to removal like others; once removed, they can be recreated only as normal writable registers and will not otherwise reflect the configuration of the formatter.

A register name is often associated with a request of the same name (without the dot); exceptions are noted.

\n[**.$**]	Count of arguments passed to currently interpolated macro or string.
\n[**.a**]	Amount of extra post-vertical line space; see **\x**.
\n[**.A**]	Approximate output is being formatted (Boolean-valued); see *troff* **−a** option.
\n[**.b**]	Font emboldening offset; see **bd**.
\n[**.br**]	The normal control character was used to call the macro being interpreted (Boolean-valued).
\n[**.c**]	Input line number; see **lf** and register "**c.**".
\n[**.C**]	AT&T *troff* compatibility mode is enabled (Boolean-valued); see **cp**. Always false when processing "**do**"; see register **.cp**.
\n[**.cdp**]	Depth of last glyph formatted in the environment; positive if glyph extends below the baseline.
\n[**.ce**]	Count of input lines remaining to be centered in the environment.
\n[**.cht**]	Height of last glyph formatted in the environment; positive if glyph extends above the baseline.
\n[**.color**]	Color output is enabled (Boolean-valued).
\n[**.cp**]	Within "**do**", the saved value of compatibility mode; see register **.C**.
\n[**.csk**]	Skew of the last glyph formatted in the environment; see register **skw**.
\n[**.d**]	Vertical drawing position in diversion.
\n[**.ev**]	Name of environment (string-valued).
\n[**.f**]	Mounting position of selected font; see **ft** and **\f**.
\n[**.F**]	Name of input file (string-valued); see **lf**.
\n[**.fam**]	Name of the environment's default font family (string-valued).
\n[**.fn**]	Resolved name of font selected in the environment (string-valued); see **ft** and **\f**.
\n[**.fp**]	Next non-zero free font mounting position index.
\n[**.g**]	Always true in GNU *troff* (Boolean-valued).
\n[**.h**]	Text baseline high-water mark on page or in diversion.
\n[**.H**]	Horizontal motion quantum of output device in basic units.
\n[**.height**]	Height of the environment's selected font; see **\H**.
\n[**.hla**]	Hyphenation language in environment (string-valued).
\n[**.hlc**]	Count of immediately preceding consecutive hyphenated lines in environment.
\n[**.hlm**]	Maximum quantity of consecutive hyphenated lines allowed in environment.
\n[**.hy**]	Automatic hyphenation mode in environment.
\n[**.hydefault**]	Hyphenation mode default in environment.
\n[**.hym**]	Hyphenation margin in environment.
\n[**.hys**]	Hyphenation space adjustment threshold in environment.
\n[**.i**]	Environment's indentation amount; see "**in**".
\n[**.in**]	Indentation amount applicable to the output line pending in the environment; see **ti**.

\n[**.int**] The text most recently formatted in the environment was "interrupted" or continued with **\c** (Boolean-valued).
\n[**.it**] Count of input lines remaining in the environment's pending input trap.
\n[**.itc**] The environment's pending input trap honors output line continuation with **\c** (Boolean-valued).
\n[**.itm**] Name of the macro associated with the pending input trap (string-valued).
\n[**.j**] The environment's adjustment mode encoded as an integer; see **ad** and **na**. Do not interpret or perform arithmetic on its value.
\n[**.k**] The environment's horizontal drawing position relative to indentation.
\n[**.kern**] Pairwise kerning is enabled (Boolean-valued).
\n[**.l**] Line length; see **ll**.
\n[**.L**] Line spacing; see **ls**.
\n[**.lg**] Ligature mode.
\n[**.linetabs**] Line-tabs mode is enabled in the environment (Boolean-valued).
\n[**.ll**] Line length applicable to the environment's pending output line.
\n[**.lt**] Environment's title line length.
\n[**.m**] Stroke color (string-valued); see **gcolor** and **\m**. The default stroke color is named "**default**".
\n[**.M**] Fill color (string-valued); see **fcolor** and **\M**. The default fill color is named "**default**".
\n[**.n**] Length of formatted output on previous output line.
\n[**.ne**] Amount of vertical space required by last **ne** that caused a trap to be sprung; also see register **.trunc**.
\n[**.nm**] Output line numbering is enabled in the environment (Boolean-valued).
\n[**.nn**] Count of lines remaining in the environment for which numbering is suppressed while output line numbering is enabled.
\n[**.ns**] No-space mode is enabled (Boolean-valued).
\n[**.o**] Page offset; see **po**.
\n[**.O**] Output suppression nesting level; see **\O**.
\n[**.p**] Page length; see **pl**.
\n[**.P**] The page is selected for output (Boolean-valued); see *troff* **−o** option.
\n[**.pe**] Page ejection is in progress (Boolean-valued).
\n[**.pn**] Number of the next page.
\n[**.ps**] The environment's type size in scaled points.
\n[**.psr**] The environment's most recently requested type size in scaled points; see **ps** and **\s**.
\n[**.pvs**] The environment's post-vertical line spacing.
\n[**.R**] Maximum representable integer in GNU *troff*.
\n[**.rj**] Count of input lines remaining to be right-aligned in the environment.
\n[**.s**] The environment's type size in points as a decimal fraction (string-valued); see **ps** and **\s**.
\n[**.S**] Reserved. Plan 9 *troff* uses this name for roughly the same purpose as GNU *troff*'s \n[**.tabs**]
\n[**.slant**] Slant in degrees of the environment's selected font; see **\S**.
\n[**.sr**] The environment's most recently requested type size in points as a decimal fraction (string-valued); see **ps** and **\s**.
\n[**.ss**] Size of the environment's minimum inter-word space in twelfths of the space width of the selected font.
\n[**.sss**] Size of the environment's supplemental inter-sentence space in twelfths of the space width of the selected font.
\n[**.sty**] The environment's selected abstract style (string-valued); see **ft** and **\f**.
\n[**.t**] Distance to next vertical position trap; see **wh** and **ch**. If no such traps exist between the drawing position and the bottom of the page, *troff* interpolates the distance to the page bottom. Within a diversion, in the absence of a diversion trap, this distance is the maximum possible vertical position supported by the output device.

\n[**.T**] An output device was explicitly selected (Boolean-valued); see *troff* **−T** option.
\n[**.tabs**] The environment's tab stop settings (if any) suitable for use as argument to **ta** (string-valued).
\n[**.trap**] Name of the next vertical position trap after the vertical drawing position. (string-valued); see **wh**, **ch**, and **dt**.
\n[**.trunc**] Amount of vertical space truncated by the most recently sprung vertical position trap, or, if the trap was sprung by an **ne**, minus the amount of vertical motion produced by **ne**; also see register **.ne**.
\n[**.u**] Filling is enabled in the environment (Boolean-valued); see **fi** and **nf**.
\n[**.U**] Unsafe mode is enabled (Boolean-valued); see *troff* **−U** option.
\n[**.v**] The environment's vertical line spacing; see **vs**.
\n[**.V**] Vertical motion quantum of the output device in basic units.
\n[**.vpt**] Vertical position traps are enabled (Boolean-valued).
\n[**.w**] Width of last glyph formatted in the environment.
\n[**.warn**] Sum of the numeric codes of enabled warning categories.
\n[**.x**] Major version number of the running *troff* formatter.
\n[**.y**] Minor version number of the running *troff* formatter.
\n[**.Y**] Revision number of the running *troff* formatter.
\n[**.z**] Name of diversion (string-valued). Empty if output is directed to the top-level diversion.
\n[**.zoom**] Magnification of the environment's selected font (in thousandths; zero if no magnification); see **fzoom**.

Writable predefined registers

Several registers are predefined but also modifiable; some are updated upon interpretation of certain requests or escape sequences. Date- and time-related registers are set to the local time as determined by *localtime*(3) when the formatter launches. This initialization can be overridden by *SOURCE_DATE_EPOCH* and *TZ*; see section "Environment" of *groff* (1).

\n[**$$**] Process ID of *troff*.
\n[**%**] Page number.
\n[**c.**] Input line number.
\n[**ct**] Union of character types of each glyph rendered into dummy environment by **\w**.
\n[**dl**] Width of last closed diversion.
\n[**dn**] Height of last closed diversion.
\n[**dw**] Day of the week (1–7; 1 is Sunday).
\n[**dy**] Day of the month (1–31).
\n[**hours**] Count of hours elapsed since midnight (0–23).
\n[**hp**] Horizontal drawing position relative to that at the start of the input line.
\n[**llx**] Lower-left *x* coordinate (in PostScript units) of PostScript image; see **psbb**.
\n[**lly**] Lower-left *y* coordinate (in PostScript units) of PostScript image; see **psbb**.
\n[**ln**] Output line number; see **nm**.
\n[**lsn**] Count of leading spaces on input line.
\n[**lss**] Amount of horizontal space corresponding to leading spaces on input line.
\n[**minutes**] Count of minutes elapsed in the hour (0–59).
\n[**mo**] Month of the year (1–12).
\n[**nl**] Vertical drawing position.
\n[**opmaxx**]
\n[**opmaxy**]
\n[**opminx**]
\n[**opminy**] These four registers mark the top left- and bottom right-hand corners of a rectangle encompassing all formatted output on the page. They are reset to −1 by **\O0** or **\O1**.
\n[**rsb**] As register **sb**, adding maximum glyph height to measurement.

\n[**rst**]	As register **st**, adding maximum glyph depth to measurement.
\n[**sb**]	Maximum displacement of text baseline below its original position after rendering into dummy environment by **\w**.
\n[**seconds**]	Count of seconds elapsed in the minute (0–60).
\n[**skw**]	Skew of last glyph rendered into dummy environment by **\w**.
\n[**slimit**]	The maximum depth of *troff*'s internal input stack. If ≤0, there is no limit: recursion can continue until available memory is exhausted. The default is 1,000.
\n[**ssc**]	Subscript correction of last glyph rendered into dummy environment by **\w**.
\n[**st**]	Maximum displacement of text baseline above its original position after rendering into dummy environment by **\w**.
\n[**systat**]	Return value of *system*(3); see **sy**.
\n[**urx**]	Upper-right *x* coordinate (in PostScript units) of PostScript image; see **psbb**.
\n[**ury**]	Upper-right *y* coordinate (in PostScript units) of PostScript image; see **psbb**.
\n[**year**]	Gregorian year.
\n[**yr**]	Gregorian year minus 1900.

Using fonts

In digital typography, a *font* is a collection of characters in a specific typeface that a device can render as glyphs at a desired size. (Terminals and some typesetters have fonts that render at only one or two sizes. As examples, take the *groff* **lj4** device's Lineprinter, and **lbp**'s Courier and Elite faces.) A *roff* formatter can change typefaces at any point in the text. The basic faces are a set of *styles* combining upright and slanted shapes with normal and heavy stroke weights: "**R**", "**I**", "**B**", and "**BI**"—these stand for *roman, bold, italic,* and *bold-italic*. For linguistic text, GNU *troff* groups typefaces into *families* containing each of these styles. (Font designers prepare families such that the styles share esthetic properties.) A *text font* is thus often a family combined with a style, but it need not be: consider the **ps** and **pdf** devices' **ZCMI** (Zapf Chancery Medium italic)—often, no other style of Zapf Chancery Medium is provided. On typesetters, at least one *special font* is available, comprising *unstyled* glyphs for mathematical operators and other purposes.

Like the AT&T *troff* formatter, GNU *troff* does not itself load or manipulate a digital font file; instead it works with a *font description file* that characterizes it, including its glyph repertoire and the *metrics* (dimensions) of each glyph. This information permits the formatter to accurately place glyphs with respect to each other. Before using a font description, the formatter associates it with a *mounting position,* a place in an ordered list of available typefaces. So that a document need not be strongly coupled to a specific font family, in GNU *troff* an output device can associate a style in the abstract sense with a mounting position. Thus the default family can be combined with a style dynamically, producing a *resolved font name.* A user-specified font name that combines family and style, or refers to a font that is not a member of a family, is already "resolved".

Fonts often have trademarked names, and even Free Software fonts can require renaming upon modification. *groff* maintains a convention that a device's serif font family is given the name **T** ("Times"), its sans-serif family **H** ("Helvetica"), and its monospaced family **C** ("Courier"). Historical inertia has driven *groff*'s font identifiers to short uppercase abbreviations of font names, as with **TR**, **TB**, **TI**, **TBI**, and a special font **S**.

The default family used with abstract styles can be changed at any time; initially, it is **T**. Typically, abstract styles are arranged in the first four mounting positions in the order shown above. The default mounting position, and therefore style, is always **1** (**R**). By issuing appropriate formatter instructions, you can override these defaults before your document writes its first glyph.

Terminals cannot change font families and lack special fonts. They support style changes by overstriking, or by altering ISO 6429/ECMA-48 *graphic renditions* (character cell attributes).

The **ft** request and **\f** escape sequence select a typeface by name, abstract style, or mounting position. The **fam** request and **\F** escape sequence set the default font family. The **ftr** request translates one font name to another; **fzoom** magnifies or reduces the typeface corresponding to a resolved one. **sty** and **fp** associate abstract styles and font names with mounting positions.

Characters and glyphs

A glyph is a graphical representation of a character. Whereas a *character* is an abstraction of semantic information, a *glyph* is an intelligible mark visible on screen or paper. A character has many possible representation forms; for example, the character "A" can be written in an upright or slanted typeface, producing distinct glyphs. Sometimes, a sequence of characters map to a single glyph:@: this is a *ligature*—the most common is 'fi'.

Space characters never become glyphs in GNU *troff*. If not discarded (as when trailing text lines), horizontal motions represent them in the output.

The formatter supports three kinds of character. An ordinary character (see section "Identifiers" above) is the most commonly used, has no special syntax, and typically represents itself. (Depending on the breadth of the output device's glyph repertoire, the characters **'**, **−**, **^**, **`**, and **~** can be exceptions to this rule. **"** and **** are not exceptions, but because they are syntactically meaningful to the formatter, access to their glyphs may require use of special characters (or changing or disabling the escape character). See *groff_char*(7). Interpolate a *special character* with the "**\[***xxx***]** or "**\C'***xxx***'**" escape sequence syntax, where *xxx* is an identifier. An *indexed character* bypasses most character-to-glyph resolution logic, uses the "**\N'***i***'**syntax, and selects a glyph from the currently selected font by its integer-valued position *i* in the output device's representation of that font. (A device's fonts do not necessarily arrange their glyphs according to a standard character encoding.)

User-defined characters are similar to string definitions (see section "Strings" above) and permit extension of or substitution within the character repertoire. Any ordinary, special, or indexed character can be user-defined. The **char**, **fchar**, **schar**, and **fschar** requests create user-defined characters employed at various stages of the character-to-glyph resolution process.

In a *troff* system, a font description file lists all of the glyphs a particular font provides. If the user requests a glyph not available in the currently selected font, the formatter looks it up an ordered list of *special fonts.* By default, the "**ps**" (PostScript) and "**pdf**" output devices support the two special fonts "**SS**" (slanted symbol) and "**S**" (symbol); and these devices' *DESC* files arrange them such that the formatter searches the former before the latter. Other output devices use different names for special fonts. Fonts mounted with the **fonts** keyword in the *DESC* file are globally available. GNU *troff*'s **special** and **fspecial** requests alter the list of fonts treated as special on a general basis, or only when a certain font is currently selected, respectively.

GNU *troff* employs the following procedure to resolve an input character into a glyph. User-defined characters make this resolution process recursive. The first step that succeeds ends the resolution procedure for the character being formatted, which may not be the last in the sequence interpolated by a user-defined character.

- Interpolate the definition of any character defined by the **char** request and apply this procedure to each character in its definition.
- Check the current font for a glyph corresponding to the character.
- Interpolate the definition of any user-defined character matching defined by the **fchar** request and apply this procedure to each character in its definition.
- Check whether the current font has a font-specific list of special fonts; if so, check the each font therein, in the order determined by the last applicable **fspecial** request, for a glyph corresponding to the character.
- Interpolate the definition of any character defined by the **fschar** request for the currently selected font, and apply this procedure to each character in its definition.
- Check each font in the list configured by the most recently issued **special** request for a glyph corresponding to the character.
- Interpolate the definition of any character defined by the **sschar** request and apply this procedure to each character in its definition.
- Finally, iterate through the list of mounted fonts by position. For each mounted font, if that font bears the **special** directive (see *groff_font*(5)), check it for a glyph corresponding to the character. This stage of the resolution process can sometimes lead to surprising results since the **fonts** directive

in the *DESC* file often contains empty positions that are filled by a macro file or document employing the **fp** request after the formatter initializes.

Special fonts

Special fonts are those that the formatter searches, in mounting position order, when it cannot find a requested glyph in the selected font. Typically, they are declared as such in their description files (see *groff_font*(5)), and contain unstyled glyphs. The "Symbol" and "Zapf Dingbats" fonts of the PostScript and PDF standards are examples. Ordinarily, only typesetters have special fonts.

GNU *troff*'s **special** and **fspecial** requests permit a document to supplement the set of fonts the device configures for glyph search without having to use the **fp** request to manipulate the list of mounting positions, which can be tedious—by default, GNU *troff* mounts 40 fonts at startup when using the **ps** device.

Hyphenation

When filling, *groff* hyphenates words as needed at user-specified and automatically determined hyphenation points. Explicitly hyphenated words such as "mother-in-law" are always eligible for breaking after each of their hyphens. The hyphenation character **\%** and non-printing break point **\:** escape sequences may be used to control the hyphenation and breaking of individual words. The **hw** request sets user-defined hyphenation points for specified words at any subsequent occurrence. Otherwise, *groff* determines hyphenation points automatically by default.

Several requests influence automatic hyphenation. Because conventions vary, a variety of hyphenation modes is available to the **hy** request; these determine whether hyphenation will apply to a word prior to breaking a line at the end of a page (more or less; see below for details), and at which positions within that word automatically determined hyphenation points are permissible. The localization macro files loaded by *troffrc* configure a default hyphenation mode appropriate to the language.

0 disables hyphenation.

1 enables hyphenation except after the first and before the last character of a word.

The remaining values "imply" **1**; that is, they enable hyphenation under the same conditions as "**.hy 1**", and then apply or lift restrictions relative to that basis.

2 disables hyphenation of the last word on a page or column, even for explicitly hyphenated words. (Hyphenation is prevented if the next page location trap is closer to the vertical drawing position than the next text baseline would be. See section "Traps" below.)

4 disables hyphenation before the last two characters of a word.

8 disables hyphenation after the first two characters of a word.

16 enables hyphenation before the last character of a word.

32 enables hyphenation after the first character of a word.

Apart from value 2, restrictions imposed by the hyphenation mode are *not* respected for words whose hyphenations have been specified with the hyphenation character ("**\%**" by default) or the **.hw** request.

Nonzero values are additive. For example, mode 12 causes *groff* to hyphenate neither the last two nor the first two characters of a word. Some values cannot be used together because they contradict; for instance, values 4 and 16, and values 8 and 32. As noted, it is superfluous to add 1 to any non-zero even mode.

The places within a word that are eligible for hyphenation are determined by language-specific data (**hla**, **hpf**, and **hpfa**) and lettercase relationships (**hcode** and **hpfcode**). Furthermore, hyphenation of a word might be suppressed due to a limit on consecutive hyphenated lines (**hlm**), a minimum line length threshold (**hym**), or because the line can instead be adjusted with additional inter-word space (**hys**).

Localization

GNU *troff* ties the set of hyphenation patterns to the hyphenation language code selected by the **hla** request. The **hpf** request is usually invoked by a localization file loaded by the *troffrc* file. *groff* provides localization files for several languages; See subsection "Localization packages" of *groff_tmac*(5). For Western languages, the localization file sets the default hyphenation mode and loads hyphenation patterns and exceptions. By

default, *troffrc* loads the localization file for English.

Writing macros

The **.de** request defines a macro named for its argument. If that name already exists as an alias, the target of the alias is redefined; see section "Strings" above. *troff* enters "copy mode" (see below), storing subsequent input lines as the definition. If the optional second argument is not specified, the definition ends with the control line "**..**" (two dots). Tabs and spaces are permitted between the dots. Alternatively, a second argument to **.de** names a macro whose call (or request whose invocation) syntax ends the definition; this *end macro* is then interpreted normally. Spaces or tabs are permitted after the first control character in the line containing this ending token, but a tab immediately after the token prevents its recognition as the end of a macro definition. Macro definitions can be nested if they use distinct end macros or if their ending tokens are sufficiently escaped. An end macro need not be defined until it is called. This fact enables a nested macro definition to begin inside one macro and end inside another.

Variants of **.de** disable compatibility mode and/or indirect the names of the macros specified for definition or termination: these are **.de1**, **.dei**, and **.dei1**. Append to macro definitions with **.am**, **.am1**, **.ami**, and **.ami1**. The **.als**, **.rm**, and **.rn** requests create an alias of, remove, and rename a macro, respectively. **.return** stops the execution of a macro immediately, returning to the enclosing context.

Parameters

Macro call and string interpolation parameters can be accessed using escape sequences starting with "**\$**". The **\n[.$]** read-only register stores the count of parameters available to a macro or string; change its value with the **.shift** request, which dequeues parameters from the current list. The **\$0** escape sequence interpolates the name by which a macro was called. Applying string interpolation to a macro does not change this name.

Copy mode

GNU *troff* processes certain requests in *copy mode:* it copies ordinary, special, and indexed characters as-is; interpolates the escape sequences **\n**, **\g**, **\$**, *****, **\V**, and **\?** normally; discards comments **\"** and **\#**; interpolates **\a**, **\e**, and **\t**, as the current leader, escape, or tab character with respectively; represents *****newline*, **\&**, **_**, **\|**, **\^**, **\{**, **\}**, **\`**, **\'**, **\–**, **\!**, **\c**, **\%**, *****space*, **\E**, **\)**, **\~**, and **\:** in an encoded form, and copies other escape sequences as-is. The term "copy mode" reflects its most visible application in requests that populate macros and strings, but other requests also use it when interpreting arguments that can't meaningfully represent typesetting operations. For example, a font selection escape sequence has no meaning in a hyphenation pattern file name (**hpf**) or a diagnostic message written to the terminal (**tm**). The complement of copy mode—a *roff* formatter's behavior when not defining or appending to a macro, string, or diversion—where all macros are interpolated, requests invoked, and valid escape sequences processed immediately upon recognition, can be termed *interpretation mode*.

The escape character (**** by default) when used before itself *quotes* an escape character for later interpretation in an enclosing context. Escape character quotation enables you to control whether the formatter interprets a given **\n**, **\g**, **\$**, *****, **\V**, or **\?** escape sequence at the time the macro containing it is defined, or later when the macro is called.

You can think of **** as a "delayed" backslash; it is the escape character followed by a backslash from which the escape character has removed its special meaning. Consequently, **** is not best considered an escape sequence, but a quoted escape character. In any escape sequence *X* that *troff* does not recognize, the formatter discards the escape character and outputs *X*. An unrecognized escape sequence causes a warning in category "**escape**", with two exceptions, **** being one. The other is **\.**, which quotes the control character. It is used to permit nested macro definitions to end without a named macro call to conclude them. Without a syntax for quoting the control character, this would not be possible. Outside of copy mode, *roff* documents should not use the **** or **\.** character sequences; they serve only to obfuscate the input. Use **\e** to represent the escape character, **\[rs]** to obtain a backslash glyph, and **\&** before **.** and **'** where *troff* expects them as control characters if you mean to use them literally.

Macro definitions can be nested to arbitrary depth. Given the input character sequence "\\" in a macro or string definition, the formatter interprets each escape character in multiple contexts; once, when populating the macro or string, where the first "\" serves its quotation function—thus only one " \" is stored in the definition.

(Verify this fact with the **pm** request.) The formatter interprets the second "\" as an escape character (assuming the escape character hasn't been changed in the meantime) each time it interpolates the macro or string definition. This fact leads to exponential growth in the quantity of escape characters required to quote and thereby delay interpolation of **\n**, **\g**, **\$**, *****, **\V**, and **\?** at each nesting level. An alternative is to use **\E**, which represents an escape character that is not interpreted in copy mode. Because **\.** is not a true escape sequence, we can't use **\E** to keep "**..**" from ending a macro definition prematurely. If the multiplicity of backslashes complicates maintenance, use end macros.

Traps

Traps are locations in the output, or conditions on the input that, when reached or fulfilled, call a specified macro. A *vertical position trap* calls a macro when the formatter's vertical drawing position reaches or passes, in the downward direction, a certain location on the output page or in a diversion. Its applications include setting page headers and footers, body text in multiple columns, and footnotes. These traps can occur at a given location either on the page (**wh**, **ch**) or in the current diversion (**dt**)—together, these are known as vertical position traps, which can be disabled and reënabled (**vpt**).

A diversion is not formatted in the context of a page, so it lacks page location traps; instead it can have a *diversion trap.* There can exist at most one such vertical position trap per diversion.

Other kinds of trap can be planted at a blank line (**blm**); at a line with leading space characters (**lsm**); after a certain number of productive input lines (**it**, **itc**); or at the end of input (**em**).

Setting a trap is also called *planting* one. It is said that a trap is *sprung* if its condition is fulfilled.

The formatter passes no arguments to macros called by traps.

Registers associated with trap management include vertical position trap enablement status (**\n[.vpt]**), distance to the next trap (**\n[.t]**), and the name of that trap (**\n[.trap]**); the count of lines remaining in the pending input trap (**\n[.it]**), the name of the macro associated with it (**\n[.itm]**), and whether that input trap honors the **\c** output line continuation escape sequence (**\n[.itc]**); amount of needed (**.ne**-requested) space that caused the most recent vertical position trap to be sprung (**\n[.ne]**), amount of needed space truncated from the amount requested (**\n[.trunc]**); page ejection status (**\n[.pe]**); and leading space count (**\n[lsn]**) with its corresponding amount of motion (**\n[lss]**).

Page location traps

A *page location trap* is a vertical position trap that applies to the page; that is, to the top-level diversion. Many can be present; manage them with the **wh** and **ch** requests. Non-negative page locations given to these requests set the trap relative to the top of the page; negative values set the trap relative to the bottom of the page. It is not possible to plant a trap less than one basic unit from the page bottom: a location of "−0" is interpreted as "0", the top of the page. An existing *visible* trap (see below) at the same location is removed; this is **wh**'s sole function if its second argument is missing.

A trap is sprung only if it is *visible,* meaning that its location is reachable on the page and it is not hidden by another trap at the same location already planted there. (A trap planted at "20i" or "−30i" will not be sprung on a page of length "11i".)

A trap above the top or at or below the bottom of the page can be made visible by either moving it into the page area or increasing the page length so that the trap is on the page. A negative trap value always uses the *current* page length; the formatter does not convert it to an absolute vertical position. We can use the **pwh** request to dump page location traps to the standard error stream; see section "Debugging" below. GNU *troff* reports their positions in basic units, and includes empty slots in the list, where a trap had been planted but subsequently (re)moved, because they can affect the visibility of subsequently planted traps.

The implicit page trap

An *implicit page trap* always exists in the top-level diversion (see below); its purpose is to eject the current page and start the next one. It works like a trap in some ways but not others. It has no name, so it cannot be moved or deleted with **wh** or **ch** requests. You cannot hide it by placing another trap at its location, and can move it only by redefining the page length with **pl**. Its operation is suppressed when vertical page traps are disabled with GNU *troff's* **vpt** request.

Diversions

In *roff* systems it is possible to format text as if for output, but instead of writing it immediately, one can *divert* the formatted text into a named storage area. It is retrieved later by specifying its name after a control character. The formatter uses the same name space for such *diversions* as for strings and macros; see section "Identifiers" above. Such text is sometimes said to be "stored in a macro", but this coinage obscures the important distinction between macros and strings on one hand and diversions on the other; the former store *unformatted* input text, and the latter capture *formatted* output. Diversions also do not interpret arguments. Applications of diversions include footnotes, tables of contents, indices, and "keeps" (preventing a page break from occurring at an inconvenient place by forcing a set of output lines to be set as a group). For orthogonality it is said that GNU *troff* populates the *top-level diversion* if no diversion is active (that is, formatted output is being "diverted" directly to the output device). The top-level diversion has no name.

Dereferencing an undefined diversion creates an empty one of that name. (GNU *troff* also emits a warning in category "mac"; see section "Warnings" of *troff* (1).) A diversion does not exist for the purpose of testing with the **d** conditional expression operator until its initial definition ends; see subsection "Conditional expressions" above.

The **di** request creates a diversion, including any partially collected line. **da** appends to a diversion, creating one if it does not already exist. If the diversion's name already exists as an alias, the target of the alias is replaced or appended to; see section "Strings" above. The pending output line is diverted as well. Switching to another environment (with the **ev** request) before invoking **di** or **da** avoids including any pending output line in the diversion. (See section "Environments[" below.)

Invoking **di** or **da** without an argument stops diverting output to the diversion named by the most recent corresponding request. Invoking **di** or **da** without an argument when no diversion is being populated does nothing. (GNU *troff* emits a warning in category "di"; see section "Warnings" of *troff* (1).)

box and **boxa** work similarly, but ignore partially collected lines. Call any of these macros again without an argument to end the diversion.

Diversion requests can be nested. The registers **.d** and **.z** report information about the current diversion, and **dn** and **dl** about the most recently closed one. **.h** is meaningful in diversions, including the top-level one.

After output to a (named) diversion stops, the formatter stores its vertical and horizontal sizes, to the writable registers **dn** and **dl**, respectively. Only the lines just processed are counted: for the computation of **dn** and **dl**, the requests **da** and **boxa** are handled as if **di** and **box** had been used, respectively—lines that have been already stored in the diversion (box) are not taken into account.

The **\!** and **\?** escape sequences and **output** request escape from a diversion, the first two to the enclosing level and the last to the top level. This facility is termed *transparent embedding*.

The **asciify** and "**unformat**" requests reprocess diversions.

Punning names

Macros, strings, and diversions share a name space; see section "Identifiers" above. Internally, the same mechanism is used to store them. You can thus call a macro with string interpolation syntax and vice versa. Interpolating a string does not hide existing macro arguments. Place the sequence **** at the end of a line in a macro definition or, within a macro definition, immediately after the interpolation of a macro as a string, to suppress the effect of a newline.

Environments

Environments store most of the parameters that control text processing. A default environment named "**0**" (zero) exists when exists when the formatter starts up; formatting-related requests and escape sequences

modify its properties.

You can create new environments and switch among them. Only one is current at any given time. Active environments are managed using a *stack,* a data structure supporting "push" and "pop" operations. The current environment is at the top of the stack. The same environment name can be pushed onto the stack multiple times, possibly interleaved with others. Popping the environment stack does not destroy the current environment; it remains accessible by name and can be made current again by pushing it at any time. Environments cannot be renamed or deleted, and can only be modified when current. To inspect the environment stack, use the **pev** request; see section "Debugging" below.

Environments store the following information.

- a partially collected line, if any
- data about the most recently output glyph and line (registers **.cdp**, **.cht**, **.csk**, **.n**, **.w**)
- typeface parameters (size, family, style, height and slant, inter-word and inter-sentence space sizes)
- page parameters (line length, title length, vertical spacing, line spacing, indentation, line numbering, centering, right-alignment, underlining, hyphenation parameters)
- filling enablement; adjustment enablement and mode
- tab stops; tab, leader, escape, control, no-break control, hyphenation, and margin characters
- input line traps
- stroke and fill colors

The **ev** request (with an argument) pushes to and (without) pops from the environment stack, while **evc** copies a named environment's parameters to the current one.

Postprocessor access

Beyond the **cf** and **trf** requests, two escape sequences and two requests enable documents to pass information directly to a postprocessor. These are useful for exercising device-specific capabilities that the *groff* language does not abstract or generalize; examples include the embedding of hyperlinks and image files. Device-specific functions are documented in each output driver's man page, such as *gropdf*(1), *grops*(1), or *grotty*(1).

The "**device**" request and **\X** escape sequence embed their arguments into GNU *troff* output as parameters to an "**x X**" device extension command (see *groff_out*(5)). The interpretation of such parameters is determined by the output driver or other postprocessor.

GNU *troff* also permits the interpolation of macro or string contents as a device extension command. The **devicem** request and **\Y** escape sequence correspond to "**.device *[***name***]**" and "**\X'*[***name***]**", respectively. They differ from their counterparts in that GNU *troff* does not interpret the contents of the string or macro *name*; further, *name* may be a macro and thus contain newlines. (There is no way to embed a newline in the arguments to "**device**" or **\X**.) The inclusion of newlines requires an extension to the AT&T *troff* device-independent page description language, and their presence confuses drivers that do not know about it (see subsection "Device control commands" of *groff_out*(5)).

Use of device extension commands early in a document (before the first text is formatted) can interfere with processing of page location traps. If you experience problems when placing them early, precede the first with a dummy character escape sequence "**\&**"; recall section "Dummy Characters" above.

The **tag** and **taga** requests are reserved for internal use.

Underlining

In *RUNOFF* (see *roff*(7)), underlining, even of lengthy passages, was straightforward because only fixed-pitch printing devices were targeted. Typesetter output posed a greater challenge. There exists a *groff* request **.ul** (see above) that underlines subsequent source lines on terminals, but on typesetters, it selects an italic font style instead. The *ms* macro package (see *groff_ms*(7)) offers a macro **.UL**, but it too produces the desired effect only on typesetters, and has other limitations.

One could adapt *ms*'s approach to the construction of a macro as follows.

```
.de UNDERLINE
. ie n \\$1\f[I]\\$2\f[P]\\$3
. el \\$1\Z'\\$2'\v'.25m'\D'l \w'\\$2'u 0'\v'-.25m'\\$3
..
```

If *doclifter*(1) makes trouble, change the macro name **UNDERLINE** into some 2-letter word, like **Ul**. Moreover, change the form of the font selection escape sequence from **\f[P]** to **\fP**.

Underlining without macro definitions

If one does not want to use macro definitions, e.g., when *doclifter* gets lost, use the following.

```
.ds u1 before
.ds u2 in
.ds u3 after
.ie n \*[u1]\f[I]\*[u2]\f[P]\*[u3]
.el \*[u1]\Z'\*[u2]'\v'.25m'\D'l \w'\*[u2]'u 0'\v'-.25m'\*[u3]
```

When using *doclifter*, it might be necessary to change syntax forms such as **\[xy]** and ***[xy]** to those supported by AT&T *troff*: ***(xy** and **\(xy**, and so on.

Then these lines could look like

```
.ds u1 before
.ds u2 in
.ds u3 after
.ie n \*[u1]\fI\*(u2\fP\*(u3
.el \*(u1\Z'\*(u2'\v'.25m'\D'l \w'\*(u2'u 0'\v'-.25m'\*(u3
```

The result looks like

```
before in after
```

Underlining by overstriking with \(ul

The **\z** escape sequence writes a glyph without advancing the drawing position, enabling overstriking. Thus, **\z***c***\(ul** formats *c* with an underrule glyph on top of it. Video terminals implement the underrule by setting a character cell's underline attribute, so this technique works in both *nroff* and *troff* modes.

Long words may then look intimidating in the input; a clarifying approach might be to use the input line continuation escape sequence *****newline* to place each underlined character on its own input line. Thus,

```
.nf
\&\fB: ${\fIvar\fR\c
\zo\(ul\
\zp\(ul\c
\&\fIvalue\fB}
.fi
```

produces

```
: ${var op value}
```

as output.

Compatibility mode

groff_diff(7) documents differences between the *roff* language recognized by GNU *troff* and that of AT&T *troff*, as well as the device, font, and device-independent page description formats described by CSTR #54. GNU *troff* provides an AT&T *troff* compatibility mode. The **cp** request and registers **.C** and **.cp** set and test the enablement of this mode.

Debugging

Preprocessors use the **lf** request to preserve the identities of line numbers and names of input files. *groff* emits a variety of error diagnostics and supports several categories of warning; the output of these can be selectively suppressed with "**warn**" (and see the **−E**, **−w**, and **−W** options of *troff*(1)). A trace of the formatter's input processing stack can be emitted when errors or warnings occur by means of *troff*(1)'s **−b** option, or produced on demand with the **backtrace** request. **tm**, **tmc**, and **tm1** can be used to emit customized diagnostic messages or for instrumentation while troubleshooting. **ex** and **ab** cause early termination with successful and

error exit codes respectively, to halt further processing when continuing would be fruitless. Examine the state of the formatter with requests that write lists of defined names—macros, strings, and diversions—(**pm**); characters (**pchar**; experimental); colors (**pcolor**); composite character mappings (**pcomposite**); environments (**pev**); occupied font mounting positions (**pfp**); font translations (**pftr**); automatic hyphenation codes (**pchar**) and exceptions (**phw**); registers (**pnr**); open streams (**pstream**); and page location traps (**pwh**). Requests can also disclose to the standard error stream the internal properties and representations of characters and classes (**pchar**), macros (and strings and diversions) (**pm**), and the list of output nodes corresponding to the pending input line (**pline**).

Authors

This document was written by Trent A. Fisher, Werner Lemberg, and G. Branden Robinson. Section "Underlining" was primarily written by Bernd Warken.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

"Troff User's Manual" by Joseph F. Ossanna, 1976 (revised by Brian W. Kernighan, 1992), AT&T Bell Laboratories Computing Science Technical Report No. 54, widely called simply "CSTR #54", documents the language, device and font description file formats, and device-independent page description language referred to collectively in *groff* documentation as "AT&T *troff*".

"A Typesetter-independent TROFF" by Brian W. Kernighan, 1982, AT&T Bell Laboratories Computing Science Technical Report No. 97 (CSTR #97), provides additional insights into the device and font description file formats and device-independent page description language.

groff (1)
: is the preferred interface to the *groff* system; it manages the pipeline that carries a source document through preprocessors, the *troff* formatter, and an output driver to viewable or printable form. It also exhaustively lists the man pages provided with the GNU *roff* system.

groff_char(7)
: discusses character encoding issues and escape sequences that produce glyphs.

groff_diff (7)
: covers differences between the GNU *troff* formatter, its device and font description file formats, its device-independent page description language, and those of AT&T *troff*.

groff_font(5)
: describes the formats of the files that describe devices (*DESC*) and fonts.

groff_tmac(5)
: surveys macro packages provided with *groff*, describes how documents can take advantage of them, offers guidance on writing macro packages and using diversions, and includes historical information.

roff (7)
: presents a detailed history of *roff* systems and summarizes concepts common to them.

Name

groff_char – GNU *roff* special character and glyph repertoire

Description

The GNU *roff* typesetting system has a large glyph repertoire suitable for production of varied literary, professional, technical, and mathematical documents. *groff* works with *characters;* an output device renders *glyphs.* *groff*'s input character set is restricted to that defined by the ISO Latin-1 (ISO 8859-1) standard. For ease of document maintenance in UTF-8 environments, it is advisable to use only the Unicode basic Latin code points; these correspond to ISO 646:1991 IRV (US-ASCII), a subset of all of the foregoing which has only 94 visible, printable code points. In *groff,* these are termed *ordinary characters.* Often, many more are desired in output.

AT&T *troff* in the 1970s faced a similar problem: the available typesetter's glyph repertoire differed from that of the computers that controlled it. *troff*'s solution was a form of escape sequence known as a *special character* to access several dozen additional glyphs available in the fonts prepared for mounting in the phototypesetter. These glyphs were mapped onto a two-character name space for a degree of mnemonic convenience; for example, the escape sequence **\(aa** encoded an acute accent and **\(sc** a section sign.

groff has lifted historical *roff* limitations on special character name lengths, but recognizes and retains compatibility with the historical names. *groff* expands the lexicon of glyphs available by name and permits users to define their own special character escape sequences with the **char** request. Special character names are *groff* identifiers; see section "Identifiers" in *groff*(7). Our discussion uses the terms "glyph name" and "special character name" interchangeably; we assume no character translations or redefinitions.

This document lists all of the glyph names predefined by *groff*'s font description files and presents the systematic notation by which it enables access to arbitrary Unicode code points and construction of composite glyphs. Glyphs listed may be unavailable, or may vary in appearance, depending on the output device and font chosen when the page was formatted. This page was rendered for device **pdf** primarily using font **TR**.

A few escape sequences that are not *groff* special characters also produce glyphs; these exist for syntactical or historical reasons. **\'**, **\`**, **\–**, and **_** are translated on input to the special character escape sequences **\[aa]**, **\[ga]**, **\[–]**, and **\[ul]**, respectively. Others include ****, **\.** (backslash-dot), and **\e**; see *groff*(7). A small number of special characters represent glyphs that are not encoded in Unicode; examples include the baseline rule **\[ru]** and the Bell System logo **\[bs].**

In *groff*, test support for any character (ordinary or special) with the conditional expression operator "**c**".

```
.ie c \[bs] \{Welcome to the \[bs] Bell System;
did you get the Wehrmacht helmet or the Death Star?\}
.el No Bell System logo.
```

While *groff* accepts eight-bit encoded input, not all such code points are valid as input. Character codes 0, 11, 13–31, and 128–159 are invalid. (This is all C0 and C1 controls except for SOH through LF [Control+A to Control+J], and FF [Control+L].) Some of these code points are used by *groff* for internal purposes, which is one reason it does not support UTF-8 natively.

Fundamental character set

The ordinary characters catalogued above, plus the space, tab, newline, and leader (Control+A), form the fundamental character set for *groff* input; anything in the language, even over one million code points in Unicode, can be expressed using it. Code points in the range 33–126 comprise a common set of printable glyphs in all of the aforementioned ISO character encoding standards. It is this character set and (with some noteworthy exceptions) the corresponding glyph repertoire for which AT&T *troff* was implemented.

All of the following characters map to glyphs as you would expect.

```
! # $ % & ( ) * + , . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ ] _
a b c d e f g h i j k l m n o p q r s t u v w x y z { | }
```

The remaining ordinary characters surprise computing professionals and others intimately familiar with the ISO character encodings. The developers of AT&T *troff* chose mappings for them that would be useful for

typesetting technical literature in a broad range of scientific disciplines: Bell Labs used the system to prepare AT&T's patent filings with the U.S. government. Further, the prevailing character encoding standard in the 1970s, USAS X3.4-1968 (ASCII), deliberately supported semantic ambiguity at some code points, and outright substitution at several others, to suit the localization demands of various national standards bodies.

The table below presents the seven exceptional code points with their typical keycap engravings, their glyph mappings and semantics in *roff* systems, and the escape sequences producing the Unicode basic Latin character they replace. The first, the neutral double quote, is a partial exception because it does represent itself, but since the *roff* language also uses it to quote macro arguments, *groff* supports a special character escape sequence as an alternative form so that the glyph can be easily included in macro arguments without requiring the user to master the quoting rules that AT&T *troff* required in that context. (Some requests, like **ds**, also treat **"** non-literally at the beginning of an argument.) Furthermore, not all of the special character escape sequences are portable to AT&T *troff* and all of its descendants; these *groff* extensions are presented using its special character form **\[]**, whereas portable special character escape sequences are shown in the traditional **\(** form. **\–** and **\e** are portable to all known *troff*s. **\e** means "the glyph of the current escape character"; it therefore can produce unexpected output if the **ec** request is used. On devices with a limited glyph repertoire, glyphs in the "keycap" and "appearance" columns on the same row of the table may look identical; except for the neutral double quote, this will *not* be the case on more-capable devices. Review your document using as many different output devices as possible.

Keycap	Appearance and meaning	Special character and meaning
"	" neutral double quote	`\[dq]` neutral double quote
'	’ closing single quote	`\[aq]` neutral apostrophe
–	- hyphen	`\-` or `\[-]` minus sign/Unix dash
\	(escape character)	`\e` or `\[rs]` reverse solidus
^	ˆ modifier circumflex	`\[ha]` circumflex/caret/"hat"
`	‘ opening single quote	`\(ga` grave accent
~	˜ modifier tilde	`\[ti]` tilde

The hyphen-minus is a particularly unfortunate case of overloading. Its awkward name in ISO 8859 and later standards reflects the many distinguishable purposes to which it had already been put by the 1980s, including a hyphen, a minus sign, and (alone or in repetition) dashes of varying widths. For best results in *roff* systems, use the "**–**" character in input outside an escape sequence *only* to mean a hyphen, as in the phrase "long-term". For a minus sign in running text or a Unix file name or command-line option dash, use **\–** (or **\[–]** in *groff* if you find it helps the clarity of the source document). (Another minus sign, for use in mathematical expressions, is available as **\(mi**.) AT&T *troff* supported em-dashes as **\(em**, as does *groff*.

The special character escape sequence for the apostrophe as a neutral single quote is typically needed only in technical content; typing words like "can't" and "Anne's" in a natural way will render correctly, because in ordinary prose an apostrophe is typeset either as a closing single quotation mark or as a neutral single quote, depending on the capabilities of the output device. By contrast, special character escape sequences should be used for quotation marks unless portability to limited or historical *troff* implementations is necessary; on those systems, the input convention is to pair the grave accent with the apostrophe for single quotes, and to double both characters for double quotes. AT&T *troff* defined no special characters for quotation marks or the apostrophe. Repeated single quotes (''thus'') will be visually distinguishable from double quotes ("thus") on terminal devices, and perhaps on others (depending on the font selected and its kerning configuration).

AT&T *troff* input	recommended *groff* input
`A Winter's Tale`	`A Winter's Tale`
`` `U.K. outer quotes' ``	`\[oq]U.K. outer quotes\[cq]`
`` `U.K. ``inner'' quotes' ``	`\[oq]U.K. \[lq]inner\[rq] quotes\[cq]`
`` ``U.S. outer quotes'' ``	`\[lq]U.S. outer quotes\[rq]`
`` ``U.S. `inner' quotes'' ``	`\[lq]U.S. \[oq]inner\[cq] quotes\[rq]`

If you frequently require quotation marks in your document, see if the macro package you're using supplies strings or macros to facilitate quotation, or define them yourself (except in man pages).

Using Unicode basic Latin characters to compose boxes and lines is ill-advised. *roff* systems have dedicated features for drawing horizontal and vertical lines, the **\l**, **\L**, and **\D** escape sequences. Also see subsection "Rules and lines" below. Preprocessors like *tbl*(1) and *pic*(1) draw boxes and will produce the best possible output for the device, falling back to basic Latin glyphs only when necessary.

Eight-bit encodings and Latin-1 supplement

ISO 646 is a seven-bit code encoding 128 code points; eight-bit codes are twice the size. ISO Latin-1 (8859-1) allocated the additional space to what Unicode calls "C1 controls" (control characters) and the "Latin-1 supplement". The C1 controls are neither printable nor usable as GNU *troff* input.

GNU *troff* handles two Latin-1 supplement characters specially, and never produces them as output.

NBSP encodes a no-break space; it maps to **\~**, the adjustable non-breaking space escape sequence.

SHY encodes a soft hyphen; it maps to **\%**, the hyphenation control escape sequence.

The remaining characters in the Latin-1 supplement represent themselves. Although they can be specified directly with the keyboard on systems configured to use Latin-1 as the character encoding, it is more portable, both to other *roff* systems and to UTF-8 environments, to use their special character escape sequences, shown below. The glyph descriptions we use are non-standard in some cases, for brevity.

¡	\[r!]	inverted exclamation mark	Ñ	\[~N]	N tilde
¢	\[ct]	cent sign	Ò	\[`O]	O grave
£	\[Po]	pound sign	Ó	\['O]	O acute
¤	\[Cs]	currency sign	Ô	\[^O]	O circumflex
¥	\[Ye]	yen sign	Õ	\[~O]	O tilde
¦	\[bb]	broken bar	Ö	\[:O]	O dieresis
§	\[sc]	section sign	×	\[mu]	multiplication sign
¨	\[ad]	dieresis accent	Ø	\[/O]	O slash
©	\[co]	copyright sign	Ù	\[`U]	U grave
ª	\[Of]	feminine ordinal indicator	Ú	\['U]	U acute
«	\[Fo]	left double chevron	Û	\[^U]	U circumflex
¬	\[no]	logical not	Ü	\[:U]	U dieresis
®	\[rg]	registered sign	Ý	\['Y]	Y acute
¯	\[a-]	macron accent	Þ	\[TP]	uppercase thorn
°	\[de]	degree sign	ß	\[ss]	lowercase sharp s
±	\[+-]	plus-minus	à	\[`a]	a grave
²	\[S2]	superscript two	á	\['a]	a acute
³	\[S3]	superscript three	â	\[^a]	a circumflex
´	\[aa]	acute accent	ã	\[~a]	a tilde
µ	\[mc]	micro sign	ä	\[:a]	a dieresis
¶	\[ps]	pilcrow sign	å	\[oa]	a ring
·	\[pc]	centered period	æ	\[ae]	ae ligature
¸	\[ac]	cedilla accent	ç	\[,c]	c cedilla
¹	\[S1]	superscript one	è	\[`e]	e grave
º	\[Om]	masculine ordinal indicator	é	\['e]	e acute
»	\[Fc]	right double chevron	ê	\[^e]	e circumflex
¼	\[14]	one quarter symbol	ë	\[:e]	e dieresis
½	\[12]	one half symbol	ì	\[`i]	i grave
¾	\[34]	three quarters symbol	í	\['i]	e acute
¿	\[r?]	inverted question mark	î	\[^i]	i circumflex
À	\[`A]	A grave	ï	\[:i]	i dieresis
Á	\['A]	A acute	ð	\[Sd]	lowercase eth
Â	\[^A]	A circumflex	ñ	\[~n]	n tilde
Ã	\[~A]	A tilde	ò	\[`o]	o grave
Ä	\[:A]	A dieresis	ó	\['o]	o acute

Å	`\[oA]`	A ring	ô	`\[^o]`	o circumflex
Æ	`\[AE]`	AE ligature	õ	`\[~o]`	o tilde
Ç	`\[,C]`	C cedilla	ö	`\[:o]`	o dieresis
È	`\[`E]`	E grave	÷	`\[di]`	division sign
É	`\['E]`	E acute	ø	`\[/o]`	o slash
Ê	`\[^E]`	E circumflex	ù	`\[`u]`	u grave
Ë	`\[:E]`	E dieresis	ú	`\['u]`	u acute
Ì	`\[`I]`	I grave	û	`\[^u]`	u circumflex
Í	`\['I]`	I acute	ü	`\[:u]`	u dieresis
Î	`\[^I]`	I circumflex	ý	`\['y]`	y acute
Ï	`\[:I]`	I dieresis	þ	`\[Tp]`	lowercase thorn
Ð	`\[-D]`	uppercase eth	ÿ	`\[:y]`	y dieresis

Special character escape forms

Glyphs that lack a character code in the basic Latin repertoire to directly represent them are entered by one of several special character escape forms. Such glyphs can be simple or composite, and accessed either by name or numerically by code point. Code points and combining properties are determined by character encoding standards, whereas glyph names as used here originated in AT&T *troff* special character escape sequences. Predefined glyph names use only characters in the basic Latin repertoire.

\(*gl* is a special character escape sequence for the glyph with the two-character name *gl*. This is the original syntax form supported by AT&T *troff*. The acute accent, **\(aa**, is an example.

\C'*glyph-name***'**
is a special character escape sequence for *glyph-name*, which can be of arbitrary length. The delimiter, shown here as a neutral apostrophe, can be any character not occurring in *glyph-name*. This syntax form was introduced in later versions of AT&T device-independent *troff*. The foregoing acute accent example can be expressed as **\C'aa'**.

\[*glyph-name***]**
is a special character escape sequence for *glyph-name*, which can be of arbitrary length but must not contain a closing square bracket "]". (No glyph names predefined by *groff* employ "]".) The foregoing acute accent example can be expressed in in GNU *troff* as **\[aa]**.

\C'*c***'** and **\[***c***]** are not synonyms for the ordinary character "*c*", but request the special character named " *c*". For example, "**\[a]**" is not "a", but rather a special character with the internal glyph name (used in font description files and diagnostic messages) **\a**, which is typically undefined. The only such glyph name *groff* predefines is the minus sign, which can therefore be accessed as **\C'−'** or **\[−]**.

\[*base-char composite-1 composite-2 . . . composite-n***]**
is a composite glyph. Glyphs like a lowercase "e" with an acute accent, as in the word "café", can be expressed as **\[e aa]**.

Normally, the formatter advances the drawing position after setting a special character, as it does for ordinary ones. *groff*'s **composite** request designates a special character as combining, suppressing advancement.

You can obtain a report of mappings defined by the **composite** request on the standard error stream with the **pcomposite** request. The *composite.tmac* macro file, loaded automatically by the default *troffrc*, maps certain special characters to combining characters as shown in subsection "Accents" below.

Unicode encodes far more characters than *groff* has names for; special character escape forms based on numerical code points enable access to any of them. Frequently used glyphs or glyph combinations can be stored in strings, and new glyph names can be created *ad hoc* with the **char** request; see *groff* (7).

\[u*nnnn*[*n*[*n*]]**]**
is a Unicode numeric special character escape sequence. Any Unicode code point can be accessed with four to six hexadecimal digits, with hexadecimal letters accepted in uppercase form only. Thus, **\[u02DA]** accesses the (spacing) ring accent, producing "˚".

Unicode code points can be composed as well; when they are, GNU *troff* requires NFD (Normalization Form D), where all Unicode glyphs are maximally decomposed. (Exception: precomposed characters in the Latin-1 supplement described above are also accepted. Do not count on this exception remaining in a future GNU *troff* that accepts UTF-8 input directly.) Thus, GNU *troff* accepts "caf**\['e]**", "caf**\[e aa]**", and "caf**\[u0065_0301]**", as ways to input "café". (Due to its legacy 8-bit encoding compatibility, at present it also accepts "caf**\[u00E9]**" on ISO Latin-1 systems.)

\[u*base-char*[*_combining-component*]...]
: constructs a composite glyph from Unicode numeric special character escape sequences. The code points of the base glyph and the combining components are each expressed in hexadecimal, with an underscore (_) separating each component. Thus, **\[u006E_0303]** produces "ñ".

\[char*nnn***]**
: expresses an eight-bit code point where *nnn* is the code point of the character, a decimal number between 0 and 255 without leading zeroes. This legacy numeric special character escape sequence is used to map characters onto glyphs via the **trin** request in macro files loaded by *grotty*(1).

Glyph tables

In this section, *groff*'s glyph name repertoire is presented in tabular form. The meanings of the columns are as follows.

Output shows the glyph as it appears on the device used to render this document; although it can have a notably different shape on other devices (and is subject to user-directed translation and replacement), *groff* attempts reasonable equivalency on all output devices.

Input shows the *groff* character (ordinary or special) that normally produces the glyph. Some code points have multiple glyph names.

Unicode is the code point notation for the glyph or combining glyph sequence as described in subsection "Special character escape forms" above. It corresponds to the standard notation for Unicode short identifiers such that *groff*'s **u***nnnn* is equivalent to Unicode's U+*nnnn*.

Notes describes the glyph, elucidating the mnemonic value of the glyph name where possible.

A plus sign "+" indicates that the glyph name appears in the AT&T *troff* user's manual, CSTR #54 (1992 revision). When using the AT&T special character syntax **\(***xx*, widespread portability can be expected from such names.

Entries marked with "***" denote glyphs used for mathematical purposes. On typesetting devices, such glyphs are typically drawn from a *special* font (see *groff_font*(5)). Often, such glyphs lack bold or italic style forms or have metrics that look incongruous in ordinary prose. A few that are not uncommon in running text have "text variants", which should work better in that context. Conversely, a handful of glyphs that are normally drawn from a text font may be required in mathematical expressions. Both sets of exceptions are noted in the tables where they appear ("Logical symbols" and "Mathematical symbols").

Basic Latin

Apart from basic Latin characters with special mappings, described in subsection "Fundamental character set" above, a few others in that range have special character glyph names. These were defined for ease of input on non-U.S. keyboards lacking keycaps for them, or for symmetry with other special character glyph names serving a similar purpose.

The vertical bar is overloaded; the **\[ba]** and **\[or]** escape sequences may render differently. See subsection "Mathematical symbols" below for unstyled variants of the plus, minus, and equals signs normally drawn from this range.

Output	Input	Unicode	Notes
"	\[dq]	u0022	neutral double quote
#	\[sh]	u0023	number sign
$	\[Do]	u0024	dollar sign
'	\[aq]	u0027	apostrophe, neutral single quote
/	\[sl]	u002F	slash, solidus +
@	\[at]	u0040	at sign
[	\[lB]	u005B	left square bracket
\	\[rs]	u005C	reverse solidus
]	\[rB]	u005D	right square bracket
^	\[ha]	u005E	circumflex, caret, “hat”
{	\[lC]	u007B	left brace
|	|	u007C	bar +
|	\[ba]	u007C	bar
|	\[or]	u007C	bitwise or +
}	\[rC]	u007D	right brace
~	\[ti]	u007E	tilde

Supplementary Latin letters

Historically, **\[ss]** developed from a ligature of “sz”. An uppercase form is available as **\[u1E9E]**, but in the German language it is of specialized use; ß does *not* normally uppercase-transform to it, but rather to “SS”. “Lowercase f with hook” is also used as a function symbol; see subsection “Mathematical symbols” below.

Output	Input	Unicode	Notes
Ð	\[-D]	u00D0	uppercase eth
ð	\[Sd]	u00F0	lowercase eth
Þ	\[TP]	u00DE	uppercase thorn
þ	\[Tp]	u00FE	lowercase thorn
ß	\[ss]	u00DF	lowercase sharp s
ı	\[.i]	u0131	i without tittle
	\[.j]	u0237	j without tittle
f	\[Fn]	u0192	lowercase f with hook, function
Ł	\[/L]	u0141	L with stroke
ł	\[/l]	u0142	l with stroke
Ø	\[/O]	u00D8	O with stroke
ø	\[/o]	u00F8	o with stroke

Ligatures and digraphs

Output	Input	Unicode	Notes
ff	\[ff]	u0066_0066	ff ligature +
fi	\[fi]	u0066_0069	fi ligature +
fl	\[fl]	u0066_006C	fl ligature +
ffi	\[Fi]	u0066_0066_0069	ffi ligature +
ffl	\[Fl]	u0066_0066_006C	ffl ligature +
Æ	\[AE]	u00C6	AE ligature
æ	\[ae]	u00E6	ae ligature
Œ	\[OE]	u0152	OE ligature
œ	\[oe]	u0153	oe ligature
IJ	\[IJ]	u0132	IJ digraph
ij	\[ij]	u0133	ij digraph

Accents

The non-combining code point in parentheses is used when the special character occurs in isolation (compare “**caf\[e aa]**” and “**caf\[aa]e**”).

Output	Input	Unicode	Notes
˝	\[a"]	u030B (u02DD)	double acute accent
¯	\[a-]	u0304 (u00AF)	macron accent
˙	\[a.]	u0307 (u02D9)	dot accent
^	\[a^]	u0302 (u005E)	circumflex accent
´	\[aa]	u0301 (u00B4)	acute accent +
`	\[ga]	u0300 (u0060)	grave accent +
˘	\[ab]	u0306 (u02D8)	breve accent
¸	\[ac]	u0327 (u00B8)	cedilla accent
¨	\[ad]	u0308 (u00A8)	dieresis accent
ˇ	\[ah]	u030C (u02C7)	caron accent
°	\[ao]	u030A (u02DA)	ring accent
~	\[a~]	u0303 (u007E)	tilde accent
˛	\[ho]	u0328 (u02DB)	hook accent

Accented characters

All of these glyphs can be composed using combining glyph names as described in subsection "Special character escape forms" above; the names below are short aliases for convenience.

Output	Input	Unicode	Notes
Á	\['A]	u0041_0301	A acute
Ć	\['C]	u0043_0301	C acute
É	\['E]	u0045_0301	E acute
Í	\['I]	u0049_0301	I acute
Ó	\['O]	u004F_0301	O acute
Ú	\['U]	u0055_0301	U acute
Ý	\['Y]	u0059_0301	Y acute
á	\['a]	u0061_0301	a acute
ć	\['c]	u0063_0301	c acute
é	\['e]	u0065_0301	e acute
í	\['i]	u0069_0301	i acute
ó	\['o]	u006F_0301	o acute
ú	\['u]	u0075_0301	u acute
ý	\['y]	u0079_0301	y acute
Ä	\[:A]	u0041_0308	A dieresis
Ë	\[:E]	u0045_0308	E dieresis
Ï	\[:I]	u0049_0308	I dieresis
Ö	\[:O]	u004F_0308	O dieresis
Ü	\[:U]	u0055_0308	U dieresis
Ÿ	\[:Y]	u0059_0308	Y dieresis
ä	\[:a]	u0061_0308	a dieresis
ë	\[:e]	u0065_0308	e dieresis
ï	\[:i]	u0069_0308	i dieresis
ö	\[:o]	u006F_0308	o dieresis
ü	\[:u]	u0075_0308	u dieresis
ÿ	\[:y]	u0079_0308	y dieresis
Â	\[^A]	u0041_0302	A circumflex
Ê	\[^E]	u0045_0302	E circumflex
Î	\[^I]	u0049_0302	I circumflex
Ô	\[^O]	u004F_0302	O circumflex
Û	\[^U]	u0055_0302	U circumflex

â	`\[^a]`	u0061_0302	a circumflex
ê	`\[^e]`	u0065_0302	e circumflex
î	`\[^i]`	u0069_0302	i circumflex
ô	`\[^o]`	u006F_0302	o circumflex
û	`\[^u]`	u0075_0302	u circumflex
À	`\[`A]`	u0041_0300	A grave
È	`\[`E]`	u0045_0300	E grave
Ì	`\[`I]`	u0049_0300	I grave
Ò	`\[`O]`	u004F_0300	O grave
Ù	`\[`U]`	u0055_0300	U grave
à	`\[`a]`	u0061_0300	a grave
è	`\[`e]`	u0065_0300	e grave
ì	`\[`i]`	u0069_0300	i grave
ò	`\[`o]`	u006F_0300	o grave
ù	`\[`u]`	u0075_0300	u grave
Ã	`\[~A]`	u0041_0303	A tilde
Ñ	`\[~N]`	u004E_0303	N tilde
Õ	`\[~O]`	u004F_0303	O tilde
ã	`\[~a]`	u0061_0303	a tilde
ñ	`\[~n]`	u006E_0303	n tilde
õ	`\[~o]`	u006F_0303	o tilde
Š	`\[vS]`	u0053_030C	S caron
š	`\[vs]`	u0073_030C	s caron
Ž	`\[vZ]`	u005A_030C	Z caron
ž	`\[vz]`	u007A_030C	z caron
Ç	`\[,C]`	u0043_0327	C cedilla
ç	`\[,c]`	u0063_0327	c cedilla
Å	`\[oA]`	u0041_030A	A ring
å	`\[oa]`	u0061_030A	a ring

Quotation marks

The neutral double quote, useful in documenting programming languages, is also available as a special character for convenient embedding in macro arguments; see subsection “Fundamental character set” above.

Output	Input	Unicode	Notes
„	`\[Bq]`	u201E	low double comma quote
‚	`\[bq]`	u201A	low single comma quote
“	`\[lq]`	u201C	left double quote
”	`\[rq]`	u201D	right double quote
‘	`\[oq]`	u2018	single opening (left) quote
’	`\[cq]`	u2019	single closing (right) quote
'	`\[aq]`	u0027	apostrophe, neutral single quote
"	`"`	u0022	neutral double quote
"	`\[dq]`	u0022	neutral double quote
«	`\[Fo]`	u00AB	left double chevron
»	`\[Fc]`	u00BB	right double chevron
‹	`\[fo]`	u2039	left single chevron
›	`\[fc]`	u203A	right single chevron

Punctuation

The Unicode name for U+00B7 is "middle dot", which is unfortunately confusable with the *groff* mnemonic for the visually similar but semantically distinct multiplication dot; see subsection "Mathematical symbols" below.

Output	Input	Unicode	Notes
¡	`\[r!]`	u00A1	inverted exclamation mark
¿	`\[r?]`	u00BF	inverted question mark
·	`\[pc]`	u00B7	centered period
—	`\[em]`	u2014	em-dash +
–	`\[en]`	u2013	en-dash
-	–	u2010	hyphen +
-	`\[hy]`	u2010	hyphen +

Brackets

On typesetting devices, the bracket extensions are font-invariant glyphs; that is, they are rendered the same way regardless of font (with a drawing escape sequence). On terminals, they are *not* font-invariant; *groff* maps them arbitrarily to U+23AA ("curly bracket extension"). In AT&T *troff*, only one glyph was available to vertically extend brackets, braces, and parentheses: **\(bv**.

Not all devices supply bracket pieces that can be piled up with **\b** due to the restrictions of the formatter's piling algorithm. The following macro offers a more general bracket-building solution.

```
.\" Make a pile centered vertically 0.5em above the baseline.
.\" The first argument is placed at the top.
.\" The pile is returned in string 'pile'.
.eo
.de pile-make
.  nr pile-wd 0
.  nr pile-ht 0
.  ds pile-args
.
.  nr pile-# \n[.$]
.  while \n[pile-#] \{\
.    nr pile-wd (\n[pile-wd] >? \w'\$[\n[pile-#]]')
.    nr pile-ht +(\n[rst] - \n[rsb])
.    as pile-args \v'\n[rsb]u'\"
.    as pile-args \Z'\$[\n[pile-#]]'\"
.    as pile-args \v'-\n[rst]u'\"
.    nr pile-# -1
.  \}
.
.  ds pile \v'(-0.5m + (\n[pile-ht]u / 2u))'\"
.  as pile \*[pile-args]\"
.  as pile \v'((\n[pile-ht]u / 2u) + 0.5m)'\"
.  as pile \h'\n[pile-wd]u'\"
..
.ec
```

Further complicating matters is that some glyphs representing bracket pieces in AT&T *troff* can be used for other mathematical symbols as well; for example, **\(lf** and **\(rf** provide the floor operator. Some output devices, such as **dvi**, don't unify such glyphs. For this reason, the glyphs **\[lf]**, **\[rf]**, **\[lc]**, and **\[rc]** are not unified with similar-looking bracket pieces. In *groff*, only glyphs with long names are guaranteed to pile up correctly for all devices—provided those glyphs are available.

Output	Input	Unicode	Notes
[	`[`	u005B	left square bracket
[	`\[lB]`	u005B	left square bracket
]	`]`	u005D	right square bracket
]	`\[rB]`	u005D	right square bracket
{	`{`	u007B	left brace
{	`\[lC]`	u007B	left brace
}	`}`	u007D	right brace
}	`\[rC]`	u007D	right brace
⟨	`\[la]`	u27E8	left angle bracket
⟩	`\[ra]`	u27E9	right angle bracket
⎪	`\[bv]`	u23AA	brace vertical extension + ***
⎪	`\[braceex]`	u23AA	brace vertical extension
⎡	`\[bracketlefttp]`	u23A1	left square bracket top
⎢	`\[bracketleftex]`	u23A2	left square bracket extension
⎣	`\[bracketleftbt]`	u23A3	left square bracket bottom
⎤	`\[bracketrighttp]`	u23A4	right square bracket top
⎥	`\[bracketrightex]`	u23A5	right square bracket extension
⎦	`\[bracketrightbt]`	u23A6	right square bracket bottom
⎧	`\[lt]`	u23A7	left brace top +
⎨	`\[lk]`	u23A8	left brace middle +
⎩	`\[lb]`	u23A9	left brace bottom +
⎧	`\[bracelefttp]`	u23A7	left brace top
⎨	`\[braceleftmid]`	u23A8	left brace middle
⎩	`\[braceleftbt]`	u23A9	left brace bottom
⎪	`\[braceleftex]`	u23AA	left brace extension
⎫	`\[rt]`	u23AB	right brace top +
⎬	`\[rk]`	u23AC	right brace middle +
⎭	`\[rb]`	u23AD	right brace bottom +
⎫	`\[bracerighttp]`	u23AB	right brace top
⎬	`\[bracerightmid]`	u23AC	right brace middle
⎭	`\[bracerightbt]`	u23AD	right brace bottom
⎪	`\[bracerightex]`	u23AA	right brace extension
⎛	`\[parenlefttp]`	u239B	left parenthesis top
⎜	`\[parenleftex]`	u239C	left parenthesis extension
⎝	`\[parenleftbt]`	u239D	left parenthesis bottom
⎞	`\[parenrighttp]`	u239E	right parenthesis top
⎟	`\[parenrightex]`	u239F	right parenthesis extension
⎠	`\[parenrightbt]`	u23A0	right parenthesis bottom

Arrows

Output	Input	Unicode	Notes
←	`\[<-]`	u2190	horizontal arrow left +
→	`\[->]`	u2192	horizontal arrow right +
↔	`\[<>]`	u2194	bidirectional horizontal arrow
↓	`\[da]`	u2193	vertical arrow down +
↑	`\[ua]`	u2191	vertical arrow up +

↕	\[va]	u2195	bidirectional vertical arrow
⇐	\[lA]	u21D0	horizontal double arrow left
⇒	\[rA]	u21D2	horizontal double arrow right
⇔	\[hA]	u21D4	bidirectional horizontal double arrow
⇓	\[dA]	u21D3	vertical double arrow down
⇑	\[uA]	u21D1	vertical double arrow up
	\[vA]	u21D5	bidirectional vertical double arrow
—	\[an]	u23AF	horizontal arrow extension

Rules and lines

On typesetting devices, the font-invariant glyphs (see subsection "Brackets" above) **\[br]**, **\[ul]**, and **\[rn]** form corners when adjacent; they can be used to build boxes. On terminal devices, they are mapped as shown in the table. The Unicode-derived names of these three glyphs are approximations.

The ordinary character _ accesses the underscore glyph in a font; **\[ul]**, by contrast, may be font-invariant on typesetting devices.

The baseline rule **\[ru]** is a font-invariant glyph, namely a rule of one-half em.

In AT&T *troff*, **\[rn]** also served as a one en extension of the square root symbol. *groff* favors **\[radicalex]** for this purpose; see subsection "Mathematical symbols" below.

Output	Input	Unicode	Notes
|	|	u007C	bar
|	\[ba]	u007C	bar
│	\[br]	u2502	box rule +
_	_	u005F	underscore, low line +
_	\[ul]	---	underrule +
‾	\[rn]	u203E	overline +
_	\[ru]	---	baseline rule +
¦	\[bb]	u00A6	broken bar
/	/	u002F	slash, solidus +
/	\[sl]	u002F	slash, solidus +
\	\[rs]	u005C	reverse solidus

Text markers

Output	Input	Unicode	Notes
○	\[ci]	u25CB	circle +
•	\[bu]	u2022	bullet +
†	\[dg]	u2020	dagger +
‡	\[dd]	u2021	double dagger +
◊	\[lz]	u25CA	lozenge, diamond
□	\[sq]	u25A1	square +
¶	\[ps]	u00B6	pilcrow sign
§	\[sc]	u00A7	section sign +
☜	\[lh]	u261C	hand pointing left +
☞	\[rh]	u261E	hand pointing right +
@	@	u0040	at sign
@	\[at]	u0040	at sign
#	#	u0023	number sign
#	\[sh]	u0023	number sign
↵	\[CR]	u21B5	carriage return
✓	\[OK]	u2713	check mark

Legal symbols

The Bell System logo is not supported in *groff*.

Output	Input	Unicode	Notes
©	\[co]	u00A9	copyright sign +
®	\[rg]	u00AE	registered sign +
™	\[tm]	u2122	trade mark sign
	\[bs]	---	Bell System logo +

Currency symbols

Output	Input	Unicode	Notes
$	$	u0024	dollar sign
$	\[Do]	u0024	dollar sign
¢	\[ct]	u00A2	cent sign +
€	\[eu]	u20AC	Euro sign
€	\[Eu]	u20AC	variant Euro sign
¥	\[Ye]	u00A5	yen sign
£	\[Po]	u00A3	pound sign
¤	\[Cs]	u00A4	currency sign

Units

Output	Input	Unicode	Notes
°	\[de]	u00B0	degree sign +
‰	\[%0]	u2030	per thousand, per mille sign
′	\[fm]	u2032	arc minute sign, foot mark +
″	\[sd]	u2033	arc second sign
µ	\[mc]	u00B5	micro sign
ª	\[Of]	u00AA	feminine ordinal indicator
º	\[Om]	u00BA	masculine ordinal indicator

Logical symbols

The variants of the not sign may differ in appearance or spacing depending on the device and font selected. Unicode does not encode a discrete "bitwise or" sign: on typesetting devices, it is drawn shorter than the bar, about the same height as a capital letter. Terminal devices unify **\[ba]** and **\[or]**.

Output	Input	Unicode	Notes
∧	\[AN]	u2227	logical and
∨	\[OR]	u2228	logical or
¬	\[no]	u00AC	logical not + ***
¬	\[tno]	u00AC	text variant of **\[no]**
∃	\[te]	u2203	there exists
∀	\[fa]	u2200	for all
∋	\[st]	u220B	such that
∴	\[3d]	u2234	therefore
∴	\[tf]	u2234	therefore
|	|	u007C	bar +
|	\[or]	u007C	bitwise or +

Mathematical symbols

\[Fn] also appears in subsection "Supplementary Latin letters" above. The plus-minus, multiplication, and division signs, **\[+-]**, **\[mu]**, and **\[di]**, are normally drawn from the special font, but have text font variants. The plus, minus, and equals signs are normally drawn from text fonts, but have special font variants. Variants may differ in appearance or spacing depending on the device and font selected.

In AT&T *troff*, **\(rn** ("root en extender") served as the horizontal extension of the radical (square root) sign, **\(sr**, and was drawn at the maximum height of the typeface's bounding box, enabling it to double as an overline (see subsection "Rules and lines" above). A contemporary font's radical sign might not ascend to such an extreme. In *groff*, you can instead use **\[radicalex]** to continue the radical sign **\[sr]**; these special characters are intended for use with text fonts. **\[sqrt]** and **\[sqrtex]** are their unstyled counterparts.

Output	Input	Unicode	Notes
½	\[12]	u00BD	one half symbol +
¼	\[14]	u00BC	one quarter symbol +
¾	\[34]	u00BE	three quarters symbol +
⅛	\[18]	u215B	one eighth symbol
⅜	\[38]	u215C	three eighths symbol
⅝	\[58]	u215D	five eighths symbol
⅞	\[78]	u215E	seven eighths symbol
¹	\[S1]	u00B9	superscript one
²	\[S2]	u00B2	superscript two
³	\[S3]	u00B3	superscript three
+	+	u002B	plus
+	\[pl]	u002B	special variant of plus + ***
−	\[-]	u002D	minus
−	\[mi]	u2212	special variant of minus + ***
	\[-+]	u2213	minus-plus
±	\[+-]	u00B1	plus-minus + ***
±	\[t+-]	u00B1	text variant of **\[+−]**
·	\[md]	u22C5	multiplication dot
×	\[mu]	u00D7	multiplication sign + ***
×	\[tmu]	u00D7	text variant of **\[mu]**
⊗	\[c*]	u2297	circled times
⊕	\[c+]	u2295	circled plus
÷	\[di]	u00F7	division sign, obelus + ***
÷	\[tdi]	u00F7	text variant of **\[di]**
⁄	\[f/]	u2044	fraction slash
*	*	u002A	asterisk
∗	\[**]	u2217	mathematical asterisk +
≤	\[<=]	u2264	less than or equal to +
≥	\[>=]	u2265	greater than or equal to +
≪	\[<<]	u226A	much less than
≫	\[>>]	u226B	much greater than
=	=	u003D	equals
=	\[eq]	u003D	special variant of equals + ***
≠	\[!=]	u003D_0338	not equals +
≡	\[==]	u2261	equivalent +
≢	\[ne]	u2261_0338	not equivalent
≅	\[=~]	u2245	approximately equal to
≃	\[|=]	u2243	asymptotically equal to +
~	\[ti]	u007E	tilde +
~	\[ap]	u223C	similar to, tilde operator +
≈	\[~~]	u2248	almost equal to
≈	\[~=]	u2248	almost equal to
∝	\[pt]	u221D	proportional to +
∅	\[es]	u2205	empty set +
∈	\[mo]	u2208	element of a set +
∉	\[nm]	u2208_0338	not element of set
⊂	\[sb]	u2282	proper subset +
⊄	\[nb]	u2282_0338	not subset

⊃	\[sp]	u2283	proper superset +
⊄	\[nc]	u2283_0338	not superset
⊆	\[ib]	u2286	subset or equal +
⊇	\[ip]	u2287	superset or equal +
∩	\[ca]	u2229	intersection, cap +
∪	\[cu]	u222A	union, cup +
∠	\[/_]	u2220	angle
⊥	\[pp]	u22A5	perpendicular
∫	\[is]	u222B	integral +
∫	\[integral]	u222B	integral ***
∑	\[sum]	u2211	summation ***
∏	\[product]	u220F	product ***
	\[coproduct]	u2210	coproduct ***
∇	\[gr]	u2207	gradient +
√	\[sr]	u221A	radical sign, square root +
‾	\[rn]	u203E	overline +
‾	\[radicalex]	---	radical extension
√‾	\[sqrt]	u221A	radical sign, square root ***
	\[sqrtex]	---	radical extension ***
⌈	\[lc]	u2308	left ceiling +
⌉	\[rc]	u2309	right ceiling +
⌊	\[lf]	u230A	left floor +
⌋	\[rf]	u230B	right floor +
∞	\[if]	u221E	infinity +
ℵ	\[Ah]	u2135	aleph symbol
f	\[Fn]	u0192	lowercase f with hook, function
ℑ	\[Im]	u2111	blackletter I, imaginary part
ℜ	\[Re]	u211C	blackletter R, real part
℘	\[wp]	u2118	Weierstrass p
∂	\[pd]	u2202	partial differential
ℏ	\[-h]	u210F	h bar
ℏ	\[hbar]	u210F	h bar

Greek glyphs

These glyphs are intended for technical use, not for typesetting Greek language text; normally, the uppercase letters have upright shape, and the lowercase ones are slanted. For the latter, we include in parentheses under "Unicode" more appropriate code points from the Mathematical Alphanumeric Symbols block of the Supplementary Multilingual Plane.

Output	Input	Unicode	Notes
A	\[*A]	u0391	uppercase alpha +
B	\[*B]	u0392	uppercase beta +
Γ	\[*G]	u0393	uppercase gamma +
Δ	\[*D]	u0394	uppercase delta +
E	\[*E]	u0395	uppercase epsilon +
Z	\[*Z]	u0396	uppercase zeta +
H	\[*Y]	u0397	uppercase eta +
Θ	\[*H]	u0398	uppercase theta +
I	\[*I]	u0399	uppercase iota +
K	\[*K]	u039A	uppercase kappa +

Λ	\[*L]	u039B	uppercase lambda +
Μ	\[*M]	u039C	uppercase mu +
Ν	\[*N]	u039D	uppercase nu +
Ξ	\[*C]	u039E	uppercase xi +
Ο	\[*O]	u039F	uppercase omicron +
Π	\[*P]	u03A0	uppercase pi +
Ρ	\[*R]	u03A1	uppercase rho +
Σ	\[*S]	u03A3	uppercase sigma +
Τ	\[*T]	u03A4	uppercase tau +
Υ	\[*U]	u03A5	uppercase upsilon +
Φ	\[*F]	u03A6	uppercase phi +
Χ	\[*X]	u03A7	uppercase chi +
Ψ	\[*Q]	u03A8	uppercase psi +
Ω	\[*W]	u03A9	uppercase omega +
α	\[*a]	u03B1 (u1D6FC)	lowercase alpha +
β	\[*b]	u03B2 (u1D6FD)	lowercase beta +
γ	\[*g]	u03B3 (u1D6FE)	lowercase gamma +
δ	\[*d]	u03B4 (u1D6FF)	lowercase delta +
ε	\[*e]	u03B5 (u1D700)	lowercase epsilon +
ζ	\[*z]	u03B6 (u1D701)	lowercase zeta +
η	\[*y]	u03B7 (u1D702)	lowercase eta +
θ	\[*h]	u03B8 (u1D703)	lowercase theta +
ι	\[*i]	u03B9 (u1D704)	lowercase iota +
κ	\[*k]	u03BA (u1D705)	lowercase kappa +
λ	\[*l]	u03BB (u1D706)	lowercase lambda +
μ	\[*m]	u03BC (u1D707)	lowercase mu +
ν	\[*n]	u03BD (u1D708)	lowercase nu +
ξ	\[*c]	u03BE (u1D709)	lowercase xi +
o	\[*o]	u03BF (u1D70A)	lowercase omicron +
π	\[*p]	u03C0 (u1D70B)	lowercase pi +
ρ	\[*r]	u03C1 (u1D70C)	lowercase rho +
σ	\[*s]	u03C3 (u1D70E)	lowercase sigma +
τ	\[*t]	u03C4 (u1D70F)	lowercase tau +
υ	\[*u]	u03C5 (u1D710)	lowercase upsilon +
ϕ	\[*f]	u03D5 (u1D717)	lowercase phi +
χ	\[*x]	u03C7 (u1D712)	lowercase chi +
ψ	\[*q]	u03C8 (u1D713)	lowercase psi +
ω	\[*w]	u03C9 (u1D714)	lowercase omega +
	\[+e]	u03F5 (u1D716)	variant epsilon (lunate)
ϑ	\[+h]	u03D1 (u1D717)	variant theta (cursive form)
ϖ	\[+p]	u03D6 (u1D717)	variant pi (similar to omega)
φ	\[+f]	u03C6 (u1D71B)	variant phi (curly shape)
ς	\[ts]	u03C2 (u1D70D)	terminal lowercase sigma +

Playing card symbols

Output	Input	Unicode	Notes
♣	\[CL]	u2663	solid club suit
♠	\[SP]	u2660	solid spade suit
♥	\[HE]	u2665	solid heart suit
♦	\[DI]	u2666	solid diamond suit

History

A consideration of the typefaces originally available to AT&T *nroff* and *troff* illuminates many conventions that one might regard as idiosyncratic fifty years afterward. (See section "History" of *roff*(7) for more context.) The face used by the Teletype Model 37 terminals of the Murray Hill Unix Room was based on ASCII, but assigned multiple meanings to several code points, as suggested by that standard. Decimal 34 (**"**) served as a dieresis accent and neutral double quotation mark; decimal 39 (**'**) as an acute accent, apostrophe, and closing (right) single quotation mark; decimal 45 (**–**) as a hyphen and a minus sign; decimal 94 (**^**) as a circumflex accent and caret; decimal 96 (**`**) as a grave accent and opening (left) single quotation mark; and decimal 126 (**~**) as a tilde accent and (with a half-line motion) swung dash. The Model 37 bore an optional extended character set offering upright Greek letters and several mathematical symbols; these were documented as early as the *kbd*(VII) man page of the (First Edition) *Unix Programmer's Manual.*

At the time Graphic Systems delivered the C/A/T phototypesetter to AT&T, the ASCII character set was not considered a standard basis for a glyph repertoire by traditional typographers. In the stock Times roman, italic, and bold styles available, several ASCII characters were not present at all, nor was most of the Teletype's extended character set. AT&T commissioned a "special" font to retain their accustomed repertoire.

A representation of the coverage of the C/A/T's text fonts follows. The glyph resembling an underscore is a baseline rule, and that resembling a vertical line is a box rule. In italics, the box rule was not slanted. We also observe that the hyphen and minus sign were already "de-unified" by the fonts provided; a decision whither to map an input "–" therefore had to be taken.

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 fi fl ffi ffl
! $ % & ( ) ‘ ’ * + – . , / : ; = ? [ ] |
• □ — - _ ¼ ½ ¾ ° † ′ ¢ ® ©
```

The special font supplied the missing ASCII and Teletype extended glyphs, among several others. The plus, minus, and equals signs appeared in the special font despite availability in text fonts "to insulate the appearance of equations from the choice of standard [read: text] fonts"—a priority since *troff* was turned to the task of mathematical typesetting as soon as it was developed.

We note that AT&T took the opportunity to de-unify the apostrophe/right single quotation mark from the acute accent (a choice ISO later duplicated in its 8859 series of standards). A slash intended to be mirror-symmetric with the backslash was also included, as was the Bell System logo; we do not attempt to depict the latter.

```
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ ς τ υ φ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
" ´ \ ^ _ ` ~ / < > { } # @ + – = *
≥ ≤ ≡ ≈ ~ ≠ ↑ ↓ ← → × ÷ ± ∞ ∂ ∇ ¬ ∫ ∝ √ ¯ ∪ ∩ ⊂ ⊃ ⊆ ⊇ ∅ ∈
§ ‡ ☜ ☞ | ○ ⎧ ⎩ ⎫ ⎭ ⎨ ⎬ | ⎣ ⎦ ⎡ ⎤
```

One ASCII character as rendered by the Model 37 was apparently abandoned. That device printed decimal 124 (|) as a broken vertical line, like Unicode U+00A6 (¦). No equivalent was available on the C/A/T; the box rule **\[br]**, brace vertical extension **\[bv]**, and "or" operator **\[or]** were used as contextually appropriate.

Devices supported by AT&T device-independent *troff* exhibited some differences in glyph detail. For example, on the Autologic APS-5 phototypesetter, the square **\(sq** became filled in the Times bold face.

Files

The files below are loaded automatically by the default *troffrc*.

/usr/share/groff/1.24.2/tmac/composite.tmac
: assigns alternate mappings for identifiers after the first in a composite special character escape sequence. See subsection "Accents" above.

/usr/share/groff/1.24.2/tmac/fallbacks.tmac
: defines fallback mappings for Unicode code points such as the increment sign (U+2206) and upper- and lowercase Roman numerals.

Authors

This document was written by James Clark, with additions by Werner Lemberg and Bernd Warken, revised to use *tbl*(1) by Eric S. Raymond, and largely rewritten by G. Branden Robinson.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. Section "Using Symbols" may be of particular note. You can browse it interactively with "info '(groff) Using Symbols'".

"An extension to the *troff* character set for Europe", E.G. Keizer, K.J. Simonsen, J. Akkerhuis; EUUG Newsletter, Volume 9, No. 2, Summer 1989

The Unicode Standard

"7-bit Character Sets" by Tuomas Salste documents the inherent ambiguity and configurable code points of the ASCII encoding standard.

"Nroff/Troff User's Manual" by Joseph F. Ossanna, 1976, AT&T Bell Laboratories Computing Science Technical Report No. 54, features two tables that throw light on the glyph repertoire available to "typesetter *roff*" when it was first written. Be careful of re-typeset versions of this document that can be found on the Internet. Some do not accurately represent the original document: several glyphs are obviously missing. More subtly, lowercase Greek letters are rendered upright, not slanted as they appeared in the C/A/T's special font and as expected by *troff* users.

groff_rfc1345(7) describes an alternative set of special character glyph names, which extends and in some cases overrides the definitions listed above.

groff(1), *troff*(1), *groff*(7)

Name

groff_diff – differences between GNU *roff* and AT&T *troff*

Description

The GNU *roff* text processing system, *groff*, is a reimplementation and extension of AT&T *troff*, the typesetting system originating in Unix systems of the 1970s. *groff* removes many arbitrary limitations and adds features, both to the input language and to the page description language output by the *troff* formatter. We also note here differences arising from *groff*'s implementation of AT&T *troff* features. The following discussion assumes familiarity with those features; see *roff* (7) for background. GNU *troff* can operate in a manner that increases support for documents written for AT&T *troff*; see section "Compatibility mode" below.

Language

GNU *troff* features identifiers of arbitrary length; supports color output, non-integral type sizes, user-defined characters, and automatic hyphenation in languages other than English; adds more conditional expression operators; recognizes additional scaling units and arithmetic operators; enables general file I/O (in "unsafe mode" only); and exposes more formatter state.

Long names

GNU *troff* introduces many new requests; with three exceptions (**cp**, **do**, **rj**), they have names longer than two characters. The names of registers, fonts, strings/macros/diversions, environments, special characters, character classes, streams, hyphenation language codes, and colors can be of any length. Anywhere AT&T *troff* supports a parameterized escape sequence that uses an opening parenthesis "(" to introduce a two-character argument, *groff* supports a square-bracketed form "[]" where the argument within can be of arbitrary length.

Font families, abstract styles, and translation

GNU *troff* can group text typefaces into *families.* For example, *groff* ships with support for families containing each of the styles "**R**", "**I**", "**B**", and "**BI**" (roman [upright], bold, italic [slanted], and bold-italic). So that a document need not be coupled to a specific font family, an output device can associate a style in the abstract sense with a mounting position. Thus the default family can combine with a style dynamically, producing a *resolved font name.* A document can *translate,* or remap, font names with the **ftr** request.

Applying the requests **cs**, **bd**, **tkf**, **uf**, or **fspecial** to an abstract style affects the member of the default family corresponding to that style. The default family can be set with the **fam** request or **–f** command-line option. The **styles** directive in the output device's *DESC* file controls which mounting positions (if any) are initially associated with abstract styles rather than fonts, and the **sty** request can update this association.

Colors

groff supports color output with a variety of color spaces and up to 16 bits per channel. Some devices, particularly terminals, may be more limited. When color support is enabled, two colors are current at any given time: the *stroke color,* with which glyphs, rules (lines), and geometric figures are drawn, and the *fill color,* which paints the interior of filled geometric figures. The **color**, **defcolor**, **gcolor**, and **fcolor** requests; **\m** and **\M** escape sequences; and **.color**, **.m**, and **.M** registers exercise color support.

Hyphenation

GNU *troff* uses a hyphenation algorithm and language-specific pattern files (based on TEX's) to decide which words can be hyphenated and where. AT&T *troff*'s hyphenation system ("suftab") was specific to English.

New requests permit finer control over hyphenation breaking; hyphenation of a word might be suppressed due to a limit on consecutive hyphenated lines (**hlm**), a minimum line length threshold (**hym**), or because the line can instead be adjusted with additional inter-word space (**hys**). The **hla** request selects a hyphenation language, whereas **hpf** and **hpfa** respectively load and append to the language's hyphenation patterns. If no hyphenation language is set or no patterns are loaded, GNU *troff* does not perform automatic hyphenation.

For automatic hyphenation to work, the formatter must know which letters are equivalent. For example, the letter "E" behaves like "e"; only the latter typically appears in hyphenation pattern files. GNU *troff* expects characters that participate in automatic hyphenation to be assigned *hyphenation codes* that define these equivalence classes. At startup, GNU *troff* assigns hyphenation codes to the letters "a"–"z", applies the same codes to "A"–"Z" in one-to-one correspondence, and assigns a code of zero to all other characters.

The **hcode** request enables application of hyphenation codes to characters outside the Unicode basic Latin set; without doing so, words containing such letters won't hyphenate properly even if the corresponding hyphenation patterns contain them. Localization files for the input character set and language configure hyphenation codes; see *groff_tmac*(5).

GNU *troff*'s **\:** escape sequence works like **\%** but produces no hyphen if the word breaks at that location.

Fractional type sizes and new scaling units

When configuring the type size, AT&T *troff* ignored scaling units and interpreted all measurements in points. Combined with integer arithmetic, this design choice made it impossible to support, for instance, ten-and-a-half-point type. In GNU *troff* an output device can select a scaling factor that subdivides a point into "scaled points". A type size expressed in scaled points can thus represent a non-integral size in points.

A *scaled point,* scaling unit **s**, is equal to 1/*sizescale* points, where the device description file, *DESC*, specifies *sizescale* and otherwise defaults to 1; see *groff_font*(5). GNU *troff* also defines the *typographical point,* scaling unit **z**, which explicitly specifies a type size of potentially non-integral measure. The program multiplies typographical points by *sizescale* and converts the value to an integer. Arguments GNU *troff* interprets in **z** units by default comprise those to the escape sequences **\H** and **\s**, to the request **ps**, the third argument to the **cs** request, and the second and fourth arguments to the **tkf** request. In GNU *troff*, the register **\n[.s]** interpolates the type size in typographical points (**z**), whereas the register **\n[.ps]** interpolates it in scaled points (**s**). "**\n[.ps]s**", "**\n[.s]z**", and "**1m**" are co-equal by definition.

For example, if *sizescale* is 1000, then a scaled point is one thousandth of a point. Consequently, "**.ps 10.5**" is synonymous with "**.ps 10.5z**"; both set the type size to 10,500 scaled points or 10.5 typographical points.

It makes no sense to use the "**z**" scaling unit in a numeric expression whose default scaling unit is neither "**u**" nor "**z**", so GNU *troff* disallows this. Similarly, it is nonsensical to use scaling units other than "**p**", "**s**", "**z**", or "**u**", in a numeric expression whose default scaling unit is "**z**", and so GNU *troff* disallows those as well.

Output devices may be limited in the type sizes they can employ. The **.s** and **.ps** registers represent the type size selected by the formatter as it understands a device's capability. the read-only registers **.psr** and (string-valued) **.sr** interpolate the last *requested* in scaled points and in points as a decimal fraction, respectively. Like the actual current and previous type size, the requested ones are properties of an environment.

For example, if a document requests a type size of 10.95 points, and the nearest size permitted by a **sizes** request (or by the **sizes** or **sizescale** directives in the device's *DESC* file) is 11 points, *groff* uses the latter value.

A further two new measurement units available in *groff* are "**M**", which indicates hundredths of an em, and "**f**", which multiplies by 65,536. The latter provides convenient fractions for color definitions with the **defcolor** request. For example, 0.5f equals 32768u.

Special fonts

GNU *troff*'s " **special**" and **fspecial** requests permit a document to supplement the set of fonts the device configures for glyph search without having to use the **fp** request to manipulate the list of mounting positions, which can be tedious—by default, GNU *troff* mounts 40 fonts at startup when using the **ps** device.

Numeric expressions

GNU *troff* permits spaces in a numeric expression within parentheses, and offers three new operators.

*e1***>?***e2* Interpolate the greater of expressions *e1* and *e2*.

*e1***<?***e2* Interpolate the lesser of expressions *e1* and *e2*.

(*c***;***e***)** Interpolate expression *e* using *c* as the default scaling unit, ignoring scaling units in *e* if *c* is empty.

Arithmetic in GNU *troff* saturates instead of wrapping.

Conditional expressions

More conditions can be tested with the "**if**" and **ie** requests, as well as the new "**while**" request.

c *chr* True if a character *chr* is available; *chr* is an ordinary, special, or indexed character, whether defined by a font description file or a request.

d *nam* True if a string, macro, diversion, or request *nam* is defined.

F *fnt* True if a font *fnt* is available; *fnt* can be an abstract style or a font name. *fnt* is handled as if it were an argument to the **ft** request (that is, the default family is combined with an abstract style and font translation is applied), but *fnt* cannot be a mounting position, and no font is mounted.

m *col* True if a color *col* is defined.

r *reg* True if a register *reg* is defined.

S *sty* True if an abstract style *sty* is registered. Font translations apply.

v Always false. This condition exists for compatibility with certain other *troff* implementations. (We refer to *vtroff*, a translator that converted the C/A/T command stream produced by early-vintage AT&T *troff* to a form suitable for Versatec and Benson-Varian plotters.)

Drawing commands

GNU *troff* extends the **\D** escape sequence with drawing commands to create filled circles and ellipses, and polygons. Stroked (outlined) objects are drawn with the stroke color and filled (solid) ones shaded with the fill color. These are independent properties; if you want a filled, stroked figure, you must draw the same figure twice using each command. A filled figure is smaller than a stroked one using the same parameters because the former is drawn only within its defined area, whereas strokes have a line thickness, set with another new drawing command.

Notational conventions

In the request and escape sequence descriptions below, GNU *troff* reads arguments named *character-sequence, command, contents, file,* and *message* in copy mode until the end of the input line. A *character-sequence* comprises one or more ordinary, special, or indexed characters; spaces; or escape sequences that interpolate only these. We name the remaining arguments for clarity; they are also *character-sequences.* A neutral double quote ‘**"**’ can optionally prefix a *character-sequence;* the formatter discards one if present, permitting initial embedded spaces in the argument.

input refers to arbitrary character sequences (up to a newline or delimiter) that GNU *troff* fully interprets, in contrast to copy mode.

Escape sequences

groff introduces several new escape sequences, extends the syntax of the AT&T *troff* escape sequences **\D**, **\f**, **\k**, **\n**, **\s**, **\$**, and *****, and alters the behavior of **\X**. The following list collates escape sequences alphabetically at first, and then by symbol roughly in Unicode code point order. Neutral apostrophes **'** illustrate escape sequences with a user-selectable delimiter. Many others are available; see subsection “Miscellaneous” and section “Compatibility Mode” below.

\A'*input***'** Interpolate 1 if *input* is a valid identifier, and 0 otherwise. Because GNU *troff* ignores any input character with an invalid code when reading it, invalid identifiers are empty or contain spaces, tabs, newlines, or escape sequences that interpolate something other than a sequence of ordinary characters. You can employ **\A** to validate a macro argument before using it to construct another escape sequence or identifier.

\B'*input***'** Interpolate 1 if *input* is a valid numeric expression, and 0 otherwise. You might use **\B** along with the “**if**” request to filter out invalid macro arguments.

\D'C *d***'** Draw filled circle of diameter *d* with its leftmost point at the drawing position.

\D'E *h v***'** Draw filled ellipse of axis lengths *h* and *v*, with its leftmost point at the drawing position.

\D'p *h1 v1 . . . hn vn***'**
Draw polygon with vertices at the drawing position and each point in sequence. GNU *troff* closes the polygon by drawing a line from (*hn*, *vn*) back to the initial drawing position; DWB and Heirloom *troff*s do not. Afterward, the drawing position is left at (*hn*, *vn*).

\D'P *h1 v1 . . . hn vn***'**
As **\D'p'**, but the polygon is filled. *groff* does not specify how the output device must fill concave or self-intersecting polygons.

\D't *n***'** Set line thickness of geometric objects to *n* basic units. A zero *n* selects the minimum supported thickness. A negative *n* (the default) selects a thickness proportional to the type size.

\E Embed an escape character that is not interpreted in copy mode (compare with **\a** and **\t**). You can use it to ease the writing of nested macro definitions. It is also convenient to define strings containing escape sequences that need to work when used in copy mode (for example, as macro arguments), or that will be interpolated at varying macro nesting depths.

\f[*fnt***]** Select typeface *fnt*, which may be a mounting position, abstract style, or font name. **\f[]**, **\f[P]**, and **\fP** are synonyms; we recommend the first.

\F*f*
\F(*fm*
\F[*fml***]** Select default font family. **\F[]** makes the previous font family the default. **\FP** is unlike **\fP**; it selects font family "P" as the default. See the **fam** request below.

\k(*rg*
\k[*reg***]** Store the horizontal drawing position, relative to that corresponding to the start of the input line (ignoring page offset and indentation), in two-character register name *rg* or arbitrary register name *reg*.

\m*c*
\m(*cl*
\m[*col***]** Set stroke color to *col*. **\m[]** restores the previous stroke color, or the default if there is none.

\M*c*
\M(*cl*
\M[*col***]** Set fill color to *col*. **\M[]** restores the previous fill color, or the default if there is none.

\n[*reg***]** Interpolate register *reg*.

\O*n*
\O[*n***]** Suppress *troff* output of glyphs and geometric objects. The sequences **\O2**, **\O3**, **\O4**, and **\O5** are intended for internal use by *grohtml*(1).

\O0
\O1 Disable and enable, respectively, the emission of glyphs and geometric objects to the output driver, provided that this sequence occurs at the outermost suppression level (see **\O3** and **\O4**). Horizontal motions corresponding to non-overstruck glyph widths still occur. These sequences also reset the registers **opminx**, **opminy**, **opmaxx**, and **opmaxy** to −1. These four registers mark the top left and bottom right hand corners of a box encompassing all written or drawn output.

\O2 At the outermost suppression level, enable emission of glyphs and geometric objects, and write to the standard error stream the page number and values of the four aforementioned registers encompassing glyphs written since the last interpolation of a **\O** sequence, as well as the page offset, line length, image file name (if any), horizontal and vertical device motion quanta, and input file name. Numeric values are in basic units.

\O3
\O4 Begin and end a nested suppression level, respectively. *grohtml* uses this mechanism to create images of output preprocessed with *pic*, *eqn*, and *tbl*. At startup, *troff* is at the outermost suppression level. *pre−grohtml* generates these sequences when processing the document, using *troff* with the **ps** output device, Ghostscript, and the PNM tools to produce images in PNG format. These sequences start a new page if the device is not **html** or **xhtml**, to reduce the number of images crossing a page boundary.

\O5[*Pfile***]**
At the outermost suppression level, write the name *file* to the standard error stream at position *P*, which must be one of **l**, **r**, **c**, or **i**, corresponding to left, right, centered, and inline alignments within the document, respectively. *file* is a name associated with the production of the next image.

\R'*name* ±*n***'**
Synonymous with "**.nr** *name* ±*n*".

\s[0]
\s'0' Restore the previous type size; no operation if there is none.

\s[±*n***]**
\s±**[***n***]**
\s'±*n***'**
\s±**'***n***'** Set the type size to, or increment or decrement it by, *n* typographical points.

\V*e*
\V(*ev*
\V[*env***]** Interpolate contents of the system environment variable *env* (one-character name *e*, two-character name *ev)* as returned by *getenv*(3). **\V** is interpreted even in copy mode.

\X'*character-sequence***'**
Unlike AT&T *troff*, GNU *troff* performs some limited processing of the sequence of ordinary characters, special characters, and spaces in *character-sequence*.

The formatter's special character repertoire is unknown to output drivers outside of glyphs named in a device's fonts, and even then they may not possess complete coverage of the names documented in *groff_char*(7). Further, escape sequences that produce horizontal or vertical motions, hyphenation breaks, or that are dummy characters may appear in strings or be converted to nodes, particularly in diversions. When they occur in a device extension command, they produce warnings in category "**char**". These are not representable when interpolated directly into device-independent output, as might be done when writing out tag names for PDF bookmarks, which can appear in a viewer's navigation pane. This is also the case for a small number of special characters, such as **\[ru]**, the baseline rule, that lack a Unicode definition.

So that any Unicode code point can be represented in device extension commands, for example in an author's name in document metadata or as a usefully named bookmark or hyperlink anchor, GNU *troff* transforms its argument to represent characters outside the Unicode Basic Latin range as Unicode code points expressed in *groff*'s notation for these, "**\[u***XXXX***]**"; see*groff_char*(7). For these transformations, the formatter ignores character translations and definitions.

GNU *troff* converts several ordinary characters that typeset as non-basic Latin code points to code points outside that range so that they are used consistently whether they are formatted as glyphs or used in a device extension command argument. These ordinary characters are "**'**", "**–**", "**^**", "**`**", and "**~**"; others are written as-is. Thus, "**'**" transforms to "**\[u2019]**".

Contrariwise, GNU *troff* translates special characters that typeset as Unicode basic Latin characters to basic Latin characters accordingly. Thus, "**\[ga]**" transforms to "**`**", "**\[Do]**" to "**$**", and so on.

\Y*m*
\Y(*ma*
\Y[*mac***]** Interpolate a macro or string as a device extension command. As **\X'*[***mac***]'**, except that GNU *troff* does not interpret the contents of *mac*; further, *mac* can be a macro and thus contain newlines, unlike the argument to **\X .** This inclusion of newlines requires an extension to the AT&T *troff* device-independent page description language, and their presence confuses drivers that do not know about it (see subsection "Device control commands" of *groff_out*(5)).

\Z'*input***'** Save the drawing position, format *input*, then restore it. GNU *troff* ignores tabs and leaders in *input* with an error diagnostic.

\# Read everything up to and including the next newline in copy mode and discard it. **\#** is like **\"**, except that **\"** does not ignore a newline; the latter therefore cannot be used by itself for a whole-line comment—it leaves a blank line on the input stream.

\$0 Interpolate the name by which the macro being interpreted was called. In GNU *troff* this name can vary; see the **als** request.

\$(*nn*
\$[*nnn***]** In a macro or string definition, interpolate the *nn*th or *nnn*th argument. In GNU *troff*, macros and strings can have an unlimited number of arguments.

\$* In a macro or string definition, interpolate the catenation of all arguments, separated by spaces.

\$@ In a macro or string definition, interpolate the catenation of all arguments, with each surrounded by double quotes and separated by spaces.

\$^ In a macro or string definition, interpolate the catenation of all arguments constructed in a form suitable for passage to the **ds** request.

\) Interpolate a *transparent* dummy character—one that is ignored by end-of-sentence detection. It behaves as **\&**, except that **\&** is treated as letters and numerals normally are after ".", "?", and "!"; **\&** cancels end-of-sentence detection, and **\)** does not.

[string* [*arg* ...]**]**
Interpolate *string,* passing it *arg* ... as arguments.

\/ Apply an *italic correction*: modify the spacing of the preceding glyph so that the distance between it and the following glyph is correct if the latter is of upright shape. For example, if an italic "f" is followed immediately by a roman right parenthesis, then in many fonts the top right portion of the "f" overlaps the top of the right parenthesis, producing *f*), which is ugly. Inserting **\/** between them produces *f*) and avoids this problem. Consider using **\/** whenever a slanted glyph is immediately followed by an upright glyph without any intervening space.

\, Apply a *left italic correction*: modify the spacing of the following glyph so that the distance between it and the preceding glyph is correct if the latter is of upright shape. For example, if a roman left parenthesis is immediately followed by an italic "f", then in many fonts the bottom left portion of the "f" overlaps the bottom of the left parenthesis, producing (*f*, which is ugly. Inserting **\,** between them produces (*f* and avoids this problem. Consider using **\,** whenever an upright glyph is followed immediately by a slanted glyph without any intervening space.

\: Insert a non-printing break point. That is, a word can break there, but the soft hyphen character does not mark the break point if it does (in contrast to "**\%**"). The remainder of the word is subject to hyphenation as normal.

\?*character-sequence***\?**
Suppress formatting of *character-sequence.* This feature has two applications.

Surround operands to the output comparison operator with **\?** to compare them by character rather than as formatted output. Since GNU *troff* reads comparands protected with **\?** in copy mode, they need not even be valid *groff* syntax. The escape character is still lexically recognized, however, and consumes the next character.

When used in a diversion, **\?** transparently embeds input, read in copy mode, until its own next occurrence on the input line. Use **\!** if you want to embed newlines in a diversion. Unlike **\!**, **\?** is interpreted even in copy mode, and a *character-sequence* in the top-level diversion is not sent to device-independent output.

\[*char***]** Typeset the special character *char*. See *groff_char*(7).

\[*base-char combining-component* . . .**]**
Typeset a composite glyph consisting of *base-char* overlaid with one or more *combining-component*s. For example, " **\[A ho]**" is a capital letter "A" with a "hook accent" (ogonek). See the **composite** request below; *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual, for details of composite glyph name construction; and *groff_char*(7) for a list of components used in composite glyph names.

\~ Insert an adjustable, unbreakable space. As with ordinary spaces, GNU *troff* discards any sequence of these at the end of an output line if a break occurs.

Restricted requests

To mitigate risks from untrusted input documents, GNU *troff* disables the **cf**, **pi**, and **sy** requests by default. Its **−U** option enables "unsafe mode", restoring their function (and enabling additional *groff* extension requests, "**open**", **opena**, and **pso**).

Altered requests

.bd *special-font font*
Stop emboldening *special-font* when *font* is selected. *special-font* must be a font name, not a mounting position.

.cf ["]*file*
Break and copy the contents of *file* as "throughput" to GNU *troff*'s output. If a diversion is in use, GNU *troff* performs the copy only when the diversion is emitted. In AT&T *troff*, the contents of *file* are immediately copied to the output regardless of whether a diversion is being written to; this behavior is so anomalous that it must be considered a bug.

GNU *troff* removes a leading neutral double quote '"' from the argument, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode. If *file* does not exist or is not readable, a warning in category "**file**" is emitted and the request has no other effect.

.de *name* [*end-name*]
.am *name* [*end-name*]
.ds *name* [["]*contents*]
.as *name* [["]*contents*]
In compatibility mode, these requests behave as **de1**, **am1**, **ds1**, and **as1**, respectively: GNU *troff* inserts a *compatibility save* token at the beginning of the macro, string, or appendment thereto as applicable and a *compatibility restore* token at its end, enabling compatibility mode during its interpolation. Thus they work as expected even if the interpolation context disables compatibility mode.

.hy *n* New values 16 and 32 are available; the former enables hyphenation before the last character in a word, and the latter enables hyphenation after the first character in a word. If invoked without an argument, the mode configured by the **hydefault** request is selected.

.lf *input-line-number* [["]*file-identifier*]
In GNU *troff* the first argument becomes the input line number of the *next* line the formatter reads. It also removes a leading neutral double quote '"' from *file-identifier,* permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode.

.nx [["]*file*]
GNU *troff* removes a leading neutral double quote '"' from *file*, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode.

.pi ["]*command*
GNU *troff* strips a leading neutral double quote from the argument, permitting initial embedded spaces in it.

.pm *name* ...
GNU *troff* reports, to the standard error stream, the JSON-encoded name and contents of each macro, string, or diversion *name*.

.so ["]*file*
GNU *troff* removes a leading neutral double quote '"' from *file*, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode. GNU *troff* searches for *file* in any directories specified by **-I** command-line options, followed by the current working directory. If *file* does not exist or is not readable, GNU *troff* emits a warning in category "**file**".

.ss *word-space-size* [*additional-sentence-space-size*]
A second argument sets the amount of additional space separating sentences on the same output line. If omitted, this amount is set to *word-space-size*. Both arguments are in twelfths of current font's space width (typically one-fourth to one-third em for Western scripts; see *groff_font*(5)). The default for both parameters is 12. Negative values are erroneous.

.sy ["]*command*
GNU *troff* strips a leading neutral double quote from the argument, permitting initial embedded spaces in it.

.ta [[*n1 n2* ... *nn*]**T** *r1 r2* ... *rn*]
GNU *troff* supports an extended syntax to specify repeating tab stops after the "**T**" mark. These values are always taken as relative distances from the previous tab stop. This is the idiomatic way to specify tab stops at equal intervals in *groff*. GNU *troff*'s startup value is " **T 0.5i**".

The syntax summary above instructs *groff* to set tabs at positions *n1*, *n2*, ..., *nn*, then at *nn* + *r1*, *nn* + *r2*, ..., *nn* + *rn*, then at *nn* + *rn* + *r1*, *nn* + *rn* + *r2*, ..., *nn* + *rn* + *rn*, and so on.

New requests

Several GNU *troff* requests work like AT&T *troff*'s " **as**" and **ds** requests, accepting an optional leading neutral double quote, notated '["]', in an argument that the formatter reads in copy mode to the end of the input line, permitting inclusion of leading spaces.

.aln *new-register existing-register*
Create alias (additional name) *new-register* of *existing-register*, causing the names to refer to the same stored object. If *existing-register* is undefined, the formatter ignores the request. GNU *troff* produces a warning in category "**reg**". See section "Warnings" of *troff*(1) regarding the enablement and suppression of warnings. To remove a register alias, invoke **rr** on its name. A register's contents do not become inaccessible until it has no more names.

.als *new-name existing-name*
Create alias (additional name) *new-name* of request, string, macro, or diversion *existing-name*, causing the names to refer to the same stored object. If *existing-name* is undefined, the formatter ignores the request. GNU *troff* produces a warning in category "**mac**". If *new-name* already exists, its contents are lost unless already aliased. See section "Warnings" of *troff*(1) regarding the enablement and suppression of warnings. To remove an alias, invoke **rm** on its name. The object itself is not destroyed until it has no more names.

When a request, macro, string, or diversion is aliased, redefinitions and appendments "write through" alias names. To replace an alias with a separately defined object, remove its name first.

.am1 *name* [*end-name*]
As "**am**", but GNU *troff* disables compatibility mode while interpreting the appendment to *name:* it inserts a *compatibility save* token at the beginning of the appendment, and a *compatibility restore* token at its end. The requests "**am**", **am1**, **de**, and **de1** can thus be intermixed freely since the compatibility save/restore tokens affect only the parts of the macro populated by **am1** and **de1**.

.ami *name* [*end-name*]
Append to macro indirectly. See **dei** below.

.ami1 *name* [*end-name*]
: As **ami**, but GNU *troff* disables compatibility mode while interpreting the appendment to the macro named by the contents of string *name;* see **am1** above.

.as1 *name* [["]*contents*]
: As "**as**", but GNU *troff* disables compatibility mode while interpreting the appendment to the string *name:* it inserts a *compatibility save* token at the beginning of the appendment, and a *compatibility restore* token at its end. The requests "**as**", **as1**, **ds**, and **ds1** can thus be intermixed freely since the compatibility save/restore tokens affect only the portions of the string populated by **as1** and **ds1**.

.asciify *div*
: *Unformat* the diversion *div* in a way such that Unicode basic Latin (US-ASCII) characters, characters translated with the **trin** request, space characters, and some escape sequences that were formatted in the diversion *div* are treated like ordinary input characters when *div* is interpolated. Doing so can be useful in conjunction with the **writem** request.

  When transforming a glyph node back into an input sequence that demands expression as a special character escape sequence, GNU *troff* uses the default escape character.

  asciify cannot return all nodes in a diversion to their source equivalents: those produced by indexed characters (**\N**), for example, remain nodes, so the result cannot be guaranteed to be a character sequence as a macro or string is. Give the diversion name as an argument to the **pm** request to inspect its contents and node list. Glyph parameters such as the type face and size are not preserved; use "**unformat**" to achieve that.

.backtrace
: Write backtrace of input stack to the standard error stream. See the **−b** option of *troff* (1).

.blm [*name*]
: Set a blank line macro (trap). If a blank line macro is thus defined, *groff* executes *name* when a blank line is encountered in the input, instead of the usual behavior. A line consisting only of spaces is also treated as blank and subject to this trap. If no argument is supplied, the default blank line behavior is (re-)established.

.box [*name*]
.boxa [*name*]
: Divert (or append) output to *name,* similarly to the **di** and **da** requests, respectively. Any pending output line is *not* included in the diversion. Without an argument, stop diverting output; any pending output line inside the diversion is discarded.

.break Exit a "**while**" loop. Do not confuse this request with a typographical break or the **br** request. See "**continue**".

.brp Break and force adjustment of the output line per the current adjustment mode. Like **br**, it does nothing if invoked with the no-break control character.

.cflags *n c*...
: Assign properties encoded by non-negative integer *n* to each character or class *c*. Spaces need not separate *c* arguments.

  n is the sum of any of the following. Some combinations are nonsensical, such as "**33**" (1 + 32).

  1 Recognize the character as ending a sentence if followed by a newline or two spaces. Initially, characters "**.?!**" have this property.

  2 Enable breaks before the character. A line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This exception can be overridden by adding 64. Initially, no characters have this property.

4 Enable breaks after the character. A line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This exception can be overridden by adding 64. Initially, characters "**–\[hy]\[em]**" have this property.

8 Mark the glyph associated with this character as overlapping other instances of itself horizontally. Initially, characters "**\[ul]\[rn]\[ru]\[radicalex]\[sqrtex]**" have this property.

16 Mark the glyph associated with this character as overlapping other instances of itself vertically. Initially, the character "**\[br]**" has this property.

32 Mark the character as transparent for the purpose of end-of-sentence recognition. In other words, an end-of-sentence character followed by any number of characters with this property is treated as the end of a sentence if followed by a newline or two spaces. This is the same as having a zero space factor in TEX. Initially, characters "**' ")]*\[dg]\[dd]\[rq]\[cq]**" have this property.

64 Ignore hyphenation codes of the surrounding characters. Use this value in combination with values 2 and 4. Initially, no characters have this property.

The remaining values were implemented for East Asian language support; those who use alphabetic scripts exclusively can disregard them.

128 Prohibit a break before the character, but allow a break after the character. This works only in combination with values 256 and 512 and has no effect otherwise. Initially, no characters have this property.

256 Prohibit a break after the character, but allow a break before the character. This works only in combination with values 128 and 512 and has no effect otherwise. Initially, no characters have this property.

512 Allow a break before or after the character. This works only in combination with values 128 and 256 and has no effect otherwise. Initially, no characters have this property.

In contrast to values 2 and 4, the values 128, 256, and 512 work pairwise. If, for example, the left character has value 512, and the right character 128, no break will be automatically inserted between them. If we use value 6 instead for the left character, a break after the character can't be suppressed since the neighboring character on the right doesn't get examined.

.char *c* [**'**]*contents*]

Define an ordinary, special, or indexed character *c* as *contents.* Omitting *contents* gives *c* an empty definition.

Defining (or redefining) a character *c* creates a formatter object that GNU *troff* recognizes like any other ordinary, special, or indexed character on input, and produces *contents* on output. When formatting *c*, GNU *troff* processes *contents* in a temporary environment and encapsulates the result in a node (see section "GNU troff Internals" in *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual); disabling compatibility mode and setting the escape character to to \ while interpreting *contents.* Any emboldening, constant spacing, or track kerning applies to this object rather than to individual glyphs resulting from the formatting of *contents.*

A character defined by **char** can be used just like a glyph provided by the output device. In particular, other characters can be translated to it with the **tr** request; it can be made the tab or leader fill character with the **tc** and **lc** requests; sequences of it can be drawn with the **\l** and **\L** escape sequences; and, if the **hcode** request is used on *c*, it is subject to automatic hyphenation.

However, a user-defined character *c* does not participate at its boundaries in kerning adjustments or italic corrections.

The formatter prevents infinite recursion by treating an occurrence of a character in its own definition as if it were undefined; when interpolating such a character, GNU *troff* emits a warning in category "**char**". (Mutually recursive character definitions are handled similarly.)

The **tr** and **trin** requests take precedence if **char** also applies to *c*. The **rchar** request removes character definitions.

.chop *name*

Remove the last character from the macro, string, or diversion *name*. This is useful for removing the newline from the end of a diversion that is to be interpolated as a string. This request can be used repeatedly on the same *name*; see section "GNU troff Internals" in *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual, for discussion of nodes inserted by *groff*.

.class *ident c* ...

Define a character class (or simply "class") *ident* comprising the members *c*..., where each *c* is an ordinary, special, or indexed character; or a *range expression.* A class thus defined can then be referred to in a **cflags** request in lieu of listing all the characters within it.

Since class and special character names share the same name space, we recommend starting and ending the class name with "[" and "]", respectively, to avoid collisions with existing special character names defined by GNU *troff* or the user (with **char** and related requests). This practice applies the presence of "]" in the class name to prevent the usage of the special character escape form "**\[**...**]**", you must therefore access a class thus named via the **\C** escape sequence.

An argument *c* can alternatively be a *range expression* consisting of a start character followed by "**–**" and then an end character. Internally, GNU *troff* converts these two character names to Unicode code points (according to the *groff* glyph list [GGL]), which determine the start and end values of the range. If that conversion fails, GNU *troff* skips the range expression and any remaining arguments.

.close *stream*

Close the named *stream*, invalidating it as an argument to the " **write**" request. See "**open**".

.composite *c1 c2*

Map ordinary or special character *c1* to *c2* when *c1* is a combining component in a composite character. Typically, **composite** is used to map a spacing character to a combining one. See *groff_char*(7).

.continue

Skip the remainder of a "**while**" loop's body, immediately retesting its conditional expression. See "**break**" above.

.color [*b*]

Enable or disable output of color-related device-independent output commands per Boolean expression *b*. It is enabled by default, and if *b* is omitted.

.cp [*b*] Enable or disable AT&T *troff* compatibility mode per Boolean expression *b*. It is disabled by default, and enabled if *b* is omitted. In compatibility mode, long names are not recognized, and the incompatibilities they cause do not arise.

.defcolor *ident scheme color-component* ...

Define a color named *ident. scheme* identifies a color space and determines the number of required *color-component*s; it must be one of " **rgb**" (three components), "**cmy**" (three components), "**cmyk**" (four components), or "**gray**" (one component). "**grey**" is accepted as a synonym of "**gray**". Each color component can be encoded as a hexadecimal value starting with **#** or **##**. The former indicates that each component is in the range 0–255 (0–FF), the latter the range 0–65535 (0–FFFF). Alternatively, a component can be specified as a decimal fraction in the range 0–1, interpreted using a default scaling unit of "**f**", which multiplies its value by 65,536 (but clamps it at 65,535).

Each output device has a color named "**default**", which cannot be redefined. A device's default stroke and fill colors are not necessarily the same.

.de1 *ident* [*end-name*]
: As "**de**", but GNU *troff* disables compatibility mode while interpreting *name:* it inserts a *compatibility save* token at the beginning of the macro definition, and a *compatibility restore* token at its end. See **.am1** above.

.dei *name* [*end-name*]
: Define macro indirectly, with the name of the macro to be defined in string *name* and the name of the end macro terminating its definition in string *end-name*.

.dei1 *name* [*end-name*]
: As **dei**, but GNU *troff* disables compatibility mode while interpreting the macro named by the contents of string *name.* See **am1** and **de1** above.

.device [**"**]*character-sequence*]
: Embed *character-sequence* into GNU *troff* output as parameters to an "x**X"** device extension command; see *groff_out*(5). The output driver or other postprocessor interprets *character-sequence* as it sees fit.

.devicem *name*
: Write contents of macro or string *name* to *troff* output as the argument to a device extension command.

.do *name* [*argument* ...]
: Interpret the string, request, diversion, or macro *name* (along with any further arguments) with compatibility mode disabled. Compatibility mode is restored (only if it was active) when the interpolation of *name* is interpreted; that is, the restored compatibility state applies to the request or contents of the macro, string, or diversion *name*, its arguments, and data read from files or pipes if *name* is the "**so**", **soquiet**, **mso**, **msoquiet**, or **pso** request.

.ds1 *name* [**"**]*contents*]
: As **ds**, but GNU *troff* disables AT&T compatibility mode while interpreting *name*: it inserts a "compatibility save" token at the beginning of *contents*, and a "compatibility restore" token after it.

.ecr
: Restore the escape character saved with **ecs**, or set escape character to "\" if none has been saved.

.ecs
: Save the current escape character.

.evc *env*
: Copy the properties of environment *env* to the current environment, except for:

  - a partially collected line, if present;
  - the interruption status of the previous input line (due to use of the **\c** escape sequence);
  - the count of remaining lines to center, to right-align, or to underline (with or without underlined spaces)—these are set to zero;
  - the activation status of temporary indentation;
  - input traps and their associated data;
  - the activation status of line numbering (which can be reactivated with "**.nm +0**"); and
  - the count of consecutive hyphenated lines (set to zero).

  Copying an environment to itself discards the foregoing data.

.fam [*fml*]
: Set default font family to *fml*. With no argument, the previous font family is selected, and if none, the formatter's default family. This default is "T" (Times), but can be overridden by the output device—see *groff_font*(5). The default font family is associated with the environment. See **\F**.

.fchar *c* [**'**]*contents*]
Define fallback character *c* as *contents*. As **char**, but while that request hides a glyph with the same name in the selected font, **fchar** definitions are used only if the font lacks a glyph for *c*. GNU *troff* performs this test before searching special fonts.

.fcolor [*col*]
Select *col* as the environment's fill color, or, without an argument, restore the previous fill color, or the default if there is none.

.fschar *f c* [**'**]*contents*]
Define fallback special character *c* for font *f* as *contents*. As **char**, but GNU *troff* locates a character defined by **fschar** after any fonts named as arguments to the **fspecial** are searched and before those named as arguments to the "**special**" request.

.fspecial *f* [*s* ...]
Declare each font *s* as special only when font **f** is selected. Initially, a font *f*'s list of associated special fonts is empty for all *f*. GNU *troff* searches fonts specified as arguments to the "**special**" request after those given as arguments to the **fspecial** request. See "**special**".

.ftr *f* [*g*]
Translate font *f* to *g*. Whenever a font named *f* is referred to in an **\f** escape sequence, in the **F** and **S** conditional expression operators, or in the **ft**, **ul**, **bd**, **cs**, **tkf**, "**special**", **fspecial**, **fp**, or **sty** requests, font *g* is used. If *g* is missing or identical to *f*, then font *f* is not translated.

.fzoom *f* [*zoom*]
Set zoom factor *zoom* for font *f*. *zoom* must be a non-negative integer; it scales the magnification by thousandths with 1000 as a basis. If *zoom* is missing or equal to zero or 1000, font *f* is not magnified. *f* must be a resolved font name, not an abstract style or mounting position.

.gcolor [*col*]
Select *col* as the environment's stroke color, or, without an argument, restore the previous stroke color, or the default if there is none.

.hcode *dst1 src1* [*dst2 src2*] ...
Set the hyphenation code of character *dst1* to that of *src1*, and so on. *dst1* must be an ordinary character (other than a numeral) or a special character, and *src1* must be an ordinary character (other than a numeral) or a special character to which a hyphenation code has already been applied. Assigning the code of an ordinary character to itself effectively creates a unique hyphenation code (which can then be copied to others). **hcode** ignores spaces between arguments. If any argument is invalid, **hcode** reports an error and stops reading them.

.hla [*lang*]
Set the hyphenation language to *lang*, or clear it if there is no argument. Hyphenation exceptions specified with the **hw** request and hyphenation patterns and exceptions specified with the **hpf** and **hpfa** requests are associated with the hyphenation language. The **hla** request is usually invoked by a localization file, which is in turn loaded by the *troffrc* or *troffrc−end* file; see the **hpf** request below.

.hlm [*n*]
Set the consecutive automatically hyphenated line limit to *n*. A negative value means "no limit". Omitting *n* implies a limit of **−1**. This value is associated with the environment. Only lines output from a given environment count toward the maximum associated with that environment. Hyphens resulting from **\%** are counted; explicit hyphens are not.

.hpf [**'**]*pattern-file*
Read hyphenation patterns from *pattern-file*. This file is sought in the same way that macro files are with the **mso** request.

The *pattern-file* should have the same format as (simple) TEX pattern files. The following scanning rules are implemented.

- A percent sign starts a comment (up to the end of the line) even if preceded by a backslash.
- "Digraphs" like **\$** are not supported.
- "**^^***xx*" (where each *x* is 0–9 or a–f) and **^^***c* (character *c* in the code point range 0–127 decimal) are recognized; other uses of **^** cause an error.
- No macro expansion is performed.
- **hpf** checks for the expression **\patterns{**...**}** (possibly with whitespace before or after the braces). Everything between the braces is taken as hyphenation patterns. Consequently, "**{**" and "**}**" are not allowed in patterns.
- Similarly, **\hyphenation{**...**}** gives a list of hyphenation exceptions.
- **\endinput** is recognized also.
- For backward compatibility, if **\patterns** is missing, the whole file is treated as a list of hyphenation patterns (but the "**%**" character is still recognized as the start of a comment).

Use the **hcode** request (see above) to map the encoding used in hyphenation pattern files to *groff*'s input encoding.

GNU *troff* ties the set of hyphenation patterns to the hyphenation language code selected by the **hla** request. The **hpf** request is usually invoked by a localization file loaded by the *troffrc* file. *groff* provides localization files for several languages; See subsection "Localization packages" of *groff_tmac*(5). For Western languages, the localization file sets the default hyphenation mode and loads hyphenation patterns and exceptions. By default, *troffrc* loads the localization file for English.

A second call to **hpf** (for the same language) replaces the old patterns with the new ones.

Invoking **hpf** causes an error if there is no hyphenation language.

If no **hpf** request is specified (either in the document, in a file loaded at startup, or in a macro package), GNU *troff* won't automatically hyphenate at all.

.hpfa ["]*pattern-file*
As **hpf**, except that the hyphenation patterns and exceptions from *pattern-file* are appended to the patterns already applied to the hyphenation language of the environment.

.hpfcode *a b* [*c d*] ...
Caution: This request will be withdrawn in a future *groff* release. Use **hcode** instead.

Define mapping values for character codes in pattern files. **hpf** or **hpfa** apply the mapping after reading or appending to the active list of patterns. Its arguments are pairs of character codes—integers from 0 to 255. The request maps character code *a* to code *b*, code *c* to code *d*, and so on. Character codes that would otherwise be invalid in GNU *troff* can be used. By default, every code maps to itself except those for letters "A" to "Z", which map to those for "a" to "z".

.hydefault *mode*
Set hyphenation mode default to *mode*. When the **hy** request is invoked without an argument, this mode is selected. The hyphenation mode default is associated with the environment. The formatter's default is **1** for AT&T *troff* compatibility. *groff* locale files generally set a more appropriate one; see *groff_tmac*(5).

.hym [*length*]
Set the (right) hyphenation margin to *length*. If the adjustment mode is not "**b**" or "**n**", the line is not hyphenated if it is shorter than *length*. Without an argument, the default hyphenation margin is reset to its default value, 0. The default scaling unit is "**m**". The hyphenation margin is associated with the environment. A negative argument resets the hyphenation margin to zero, emitting a warning in category "**range**".

.hys [*hyphenation-space*]
: Suppress hyphenation of the line in adjustment modes "**b**" or "**n**", if that adjustment can be achieved by adding no more than *hyphenation-space* extra space to each inter-word space. Without an argument, the hyphenation space adjustment threshold is set to its default value, 0. The default scaling unit is "**m**". The hyphenation space adjustment threshold is associated with the environment. A negative argument resets the hyphenation space adjustment threshold to zero, emitting a warning in category "**range**".

.itc n [*mac*]
: As "**it**", but lines interrupted with the **\c** escape sequence do not apply to the line count.

.kern [*b*]
: Enable or disable pairwise kerning of glyphs in the environment per Boolean expression *b*. It is enabled by default, and if *b* is omitted.

.length *reg* [["]*contents*]
: Compute the number of characters in *contents* and store the count in the register *reg*. If *reg* doesn't exist, GNU *troff* creates it.

  Caution: If you interpolate a macro or diversion in *contents* (see section "Punning Names" in *groff* (7)), the **length** request counts characters (or nodes) only up to the first newline, and leaves the rest on the input stream. In conventional circumstances, that means the remainder is interpreted, and may be formatted. To discover the length of any string, macro, or diversion, use the **pm** request. See section "Debugging" below.

.linetabs [*b*]
: Activate or deactivate line-tabs in the environment per Boolean expression *b*. They are inactive by default, and activated if *b* is omitted. When line-tabs are active, tab stops are computed relative to the start of the pending output line instead of the drawing position corresponding to the start of the input line.

.lsm [*name*]
: Set a leading space trap, calling the macro *name* when GNU *troff* encounters leading spaces on a text line; the implicit line break that normally happens in this case is suppressed. The formatter stores the count of leading spaces on the text line in register **lsn**, and the amount of corresponding horizontal motion in register **lss**, irrespective of whether a leading space trap is set. When it is, GNU *troff* removes the leading spaces from the input line and produces no motion before calling *name*.

  If no argument is supplied, GNU *troff* reëstablishes the default handling of leading spaces on text lines (breaking the line when filling, and formatting a horizontal motion of **\n[lsn]** word spaces).

.mso ["]*file*
: As "**so**", except that GNU *troff* searches for the specified *file* in the same directories as macro files; see *GROFF_TMAC_PATH* in section "Environment" of *groff* (1) and **–m** in section "Options" of the same page. If *file* does not exist or is not readable, a warning in category "**file**" is emitted and the request has no other effect.

.msoquiet ["]*file*
: As **mso**, but no warning is emitted if *file* does not exist.

.nop [*input*]
: Interpret *input* as if it were an input line. **nop** resembles "**.if 1**"; it puts a break on the output if *input* is empty. Unlike "**if**", it cannot govern conditional blocks. Its application is to maintain consistent indentation within macro definitions even when formatting output.

.nroff Make the **n** conditional expression evaluate true and **t** false. See **troff**.

.open *ident* [**"**]*file*
Open *file* for writing and associate a stream named *ident* with it, making it available for "**write**" requests. Unsafe request; disabled by default. Also see **writec** and "**close**".

.opena *ident* [**"**]*file*
As "**open**", but appends to *file* if it already exists instead of overwriting it.

.output [**"**]*character-sequence*
Emit *character-sequence*, transparently to GNU *troff* output; this usage is similar to that of **\!** when it occurs in the top-level diversion.

.pchar *c* ...
Report, to the standard error stream, information about each character (be it ordinary, special, or indexed) or character class *c*. A character defined by a request (**char**, **fchar**, **fschar**, or **schar**) reports its contents as a JSON-encoded string, but the output is not otherwise in JSON format.

.pcolor [*col* ...]
Report, to the standard error stream, each defined color named *col*, its color space identifier, and channel value assignments, or, without arguments, those of all defined colors. A device's default stroke and/or fill colors, "default", are not listed since they are immutable and their details unknown to the formatter.

.pcomposite
Report, to the standard error stream, the list of configured composite character mappings. See "**composite**" above. The "from" code point is listed first, followed by its "to" mapping.

.pev Report the state of the current environment followed by that of all other environments to the standard error stream.

.pfp Report, to the standard error stream, the list of occupied font mounting positions. Occupied mounting positions are listed, one per line, in increasing order, followed by the typeface name; if the name corresponds to an abstract style, the entry ends there. Otherwise, the name of the font description file and the font's "internal name" datum, the meaning of which varies by output device, follow.

.pftr Report, to the standard error stream, the list of font translations. See **pftr** above. The "from" font identifier is listed first, followed by its "to" translation.

.phw Report, to the standard error stream, the list of hyphenation exception words associated with the hyphenation language selected by the **hla** request. A "**–**" marks each hyphenation point. A word prefixed with "**–**" is not hyphenated at all. The report suffixes words to which automatic hyphenation applies (meaning those defined in a hyphenation pattern file rather than with the **hw** request) with a tab and asterisk "*****".

.pline Report, in JSON syntax to the standard error stream, the list of output nodes corresponding to the pending output line. In JSON, a pair of empty brackets "**[ ]**" represents an empty list. A *pending* output line has not yet undergone adjustment, and lacks a line number and margin character (all as applicable).

.pnr [*reg* ...]
Report the name and value and, if its type is numeric, the autoincrement amount and assigned format of each register *reg*, or, without arguments, those of all defined registers, to the standard error stream.

.psbb *file*
Get the bounding box of a PostScript image *file*. This file must conform to Adobe's Document Structuring Conventions; the request attempts to extract the bounding box values from a **%%BoundingBox** comment. After invocation, the *x* and *y* coordinates (in PostScript units) of the lower left and upper right corners can be found in the registers **\n[llx]**, **\n[lly]**, **\n[urx]**, and **\n[ury]**, respectively. If an error occurs, these four registers are set to zero.

.pso ["]*command*

As "**so**", except that input comes from the standard output stream of *command*, which is passed to *popen*(3).

.pstream

Report, in JSON syntax to the standard error stream, the list of open streams, including the name of each open stream, the name of the file backing it, and its mode (writing or appending).

.pwh Report the names and vertical positions, in basic units, of all page location traps to the standard error stream. GNU *troff* reports empty slots in the list, where a trap had been planted but subsequently (re)moved, because they can affect the visibility of subsequently planted traps.

.pvs ±*n*

Set the post-vertical line spacing to *n*; default scaling unit is " **p**". With no argument, the post-vertical line space is set to its previous value.

In GNU *troff*, the distance between text baselines consists of the extra pre-vertical line spacing set by the most negative **\x** argument on the pending output line, the vertical spacing (**vs**), the extra post-vertical line spacing set by the most positive **\x** argument on the pending output line, and the post-vertical line spacing set by this request.

.rchar *c*...

Remove definition of each ordinary, special, or indexed character *c*, undoing the effect of a **char**, **fchar**, or **schar** request. Spaces need not separate *c* arguments. The character definition removed (if any) is the first encountered in the resolution process documented in section "Using Symbols" of *Groff: The GNU Implementation of troff*. Glyphs, which are defined by font description files, cannot be removed.

.return [*input*]

Stop interpreting an interpolated macro, skipping to the end of its definition. Do not confuse "**return**" with **rt**. If called with an argument *input,* GNU *troff* performs the skip twice—once within the macro being interpreted and once in an enclosing macro, permitting a macro to wrap the request (as *trace.tmac* does).

.rfschar *f c* ...

Remove each fallback special character *c* for font *f*. Spaces need not separate *c* arguments. See **fschar**.

.rj [*n*] Break, right-align the next *n* (default: 1) input lines, then break again. **rj** implies "**.ce 0**", and **ce** implies "**.rj 0**". Invoking the request with the no-break control character suppresses the first break.

.rnn *r1 r2*

Rename register *r1* to *r2*. If *r1* doesn't exist, the request is ignored.

.schar *c* [["]*contents*]

Define global fallback character *c* as *contents*. As **char**, but GNU *troff* locates a character defined with **schar** after any fonts named as arguments to the "**special**" request and before any mounted special fonts.

.shc [*c*]

Set the soft hyphen character, inserted when a word is hyphenated automatically or at a hyphenation character, to *c*. If *c* is omitted, the soft hyphen character is set to the default, **\[hy]**. If the selected glyph does not exist in the font in use at a potential hyphenation point, then the line is not broken at that point. Neither character definitions (**char** and similar) nor translations (**tr** and similar) are considered when assigning the soft hyphen character.

.shift [*n*]

Shift macro or string parameters *n* places (by 1 if *n* omitted): argument *i* becomes argument $i - n$; arguments 1 to *n* become unavailable. Shifting by a non-positive amount, or outside of a macro or string definition, performs no operation. The register **.$** adjusts its value accordingly.

.sizes *s1 s2 . . . sn* [**0**]
: Set the available type sizes to *s1*, *s2*, . . . *sn* scaled points. The list of sizes can be terminated by an optional "**0**". Each *si* can also be a range *m–n*. In contrast to the device description file directive of the same name (see *groff_font*(5)), the argument list can't extend over more than one line.

.soquiet ["]*file*
: As "**so**", but no warning is emitted if *file* does not exist.

.special [*s . . .*]
: Declare each font *s* as special, irrespective of its description file, populating a list that GNU *troff* searches, in order, to find the glyph demanded. GNU *troff* mounts each font *s*. Invoking **special** without arguments empties the list. A font is not automatically unmounted if a subsequent **special** request removes it from the list. Initially, the list is empty. GNU *troff* searches fonts specified as arguments to the "**special**" request after those given as arguments to the **fspecial** request.

.spreadwarn [*limit*]
: Emit a "**break**" warning if the additional space inserted for each space between words in an output line adjusted to both margins with "**.ad b**" is larger than or equal to *limit*. A negative value is treated as zero; an absent argument toggles the warning on and off without changing *limit*. The default scaling unit is **m**. At startup, **spreadwarn** is inactive and *limit* is 3 m.

  For example, "**.spreadwarn 0.2m**" warns if *troff* must add 0.2 m or more to each inter-word space in a line.

.stringdown *str*
.stringup *str*
: Alter the string named *str* by replacing each of its bytes with its lowercase (**down**) or uppercase (**up**) version (if one exists). Special characters (see *groff_char*(7)) will often transform in the expected way due to the regular naming convention for accented characters. When they do not, use substrings and/or catenation.

.sty *pos style*
: Associate abstract *style* with non-negative font mounting position *pos*.

.substring *str start* [*end*]
: Replace the string named *str* with its substring bounded by the indices *start* and *end*, inclusively. The first character in the string has index 0. Negative indices count backward from the end of the string: the last character has index −1, the character before the last has index −2, and so on. If *end* is omitted, **−1** is implied.

.tkf *f s1 n1 s2 n2*
: Enable track kerning for font *f*. When the current font is *f*, the width of every glyph is increased by an amount between *n1* and *n2*; when the current type size is less than or equal to *s1*, the width is increased by *n1*; when it is greater than or equal to *s2*, the width is increased by *n2*; when the type size is greater than or equal to *s1* and less than or equal to *s2*, the increase in width is a linear function of the type size.

.tm1 [["]*message*]
: As **tm**, but removes a leading neutral double quote '"' from *message*, permitting initial embedded spaces in it.

.tmc [["]*message*]
: As **tm1**, but does not append a newline.

.trf *file*
: Break and copy *file* as "throughput" to GNU *troff* output, discarding characters that are invalid as input; contrast with **cf**. Each line of *file* is output as if preceded by **\!**, but is not interpreted by the formatter. If *file* does not end with a newline, **trf** appends one. Invoking the request with the no-break control character suppresses the break.

.trin *abcd*
: As the **tr** request, but the **asciify** request uses the character code (if any) before the character translation.

.trnt *abcd*
: As the **tr** request, but the translations do not apply to text that is transparently throughput into a diversion with **\!**.

.troff
: Make the **t** conditional expression evaluate true and **n** false. See **nroff**.

.unformat *div*
: Unformat the diversion *div*. Unlike **asciify**, "**unformat**" handles only tabs and spaces between words, the latter usually arising from spaces or newlines in the input. Tabs are treated as input tokens, and spaces become adjustable again. The vertical sizes of lines are not preserved, but glyph information (font, type size, space width, and so on) is retained.

.vpt [*b*]
: Enable or disable vertical position traps per Boolean expression *b*. They are enabled by default, and if *b* is omitted. Vertical position traps are those set by the **ch**, **wh**, and **dt** requests. Vertical position trap enablement is global.

.warn [*n*]
: Select categories of warnings to be reported. *n* is the sum of the numeric codes associated with each warning category that is to be enabled; all other categories are disabled. The categories and their associated codes are listed in section "Warnings" of *troff*(1). For example, "**.warn 0**" disables all warnings, and "**.warn 1**" disables all warnings except those about missing glyphs. If no argument is given, all warning categories are enabled.

.warnscale *scaling-unit*
: Select scaling unit used in certain warnings (one of **u**, **i**, **c**, **p**, or **P**; default: **i**). Ignored on *nroff*-mode output devices, for which these diagnostics report the vertical page location in lines, and the horizontal page location in ens.

.while *cond-expr input*
: Evaluate the conditional expression *cond-expr*, and repeatedly execute *input* unless and until *cond-expr* evaluates false. *input,* which is often a conditional block, is referred to as the "**while**" request's *body.*

  GNU *troff* treats the body of a "**while**" request similarly to that of a **de** request (albeit one not read in copy mode), but stores it under an internal name and deletes it when the loop finishes. The operation of a macro containing a "**while**" request can slow significantly if its body is large. Each time GNU *troff* interpolates the macro, it parses and stores the "**while**" body again. An often better solution—and one that is more portable, since AT&T *troff* lacked the "**while**" request—is to instead write a recursive macro, which is parsed only once (unless you redefine it). To prevent infinite loops, GNU *troff* limits the default number of available recursion levels to 1,000 or somewhat less (because things other than macro calls can be on the input stack). You can disable this protective measure, or alter the limit, by setting the **slimit** register. See section "Debugging" below.

  If a "**while**" body begins with a conditional block, its closing brace must end an input line.

  The "**break**" and "**continue**" requests alter a "**while**" loop's flow of control.

.write *stream* [**'**]*character-sequence*]
: Write *character-sequence*, a sequence of ordinary characters, spaces, or tabs read in copy mode, to *stream*, which must previously have been the subject of an " **open**" (or **opena**) request, followed by a newline. GNU *troff* flushes the stream after writing to it.

.writec *stream* [**'**]*character-sequence*]
: As "**write**", but does not append a newline to *contents*.

.writem *stream name*

Write the contents of the macro or string *name* to *stream*, which must previously have been the subject of an "**open**" (or **opena**) request. GNU *troff* reads the contents of *name* in copy mode.

Altered registers

GNU *troff* alters the semantics of two AT&T *troff* registers and imparts semantics to two others that CSTR #54 documented as "reserved".

\n[.R] Because GNU *troff* dynamically manages register storage, it repurposes the **.R** register to interpolate the maximum integer representable in the formatter. Favor its use over numeric literals with many zeroes or nines to indicate an arbitrary large quantity.

\n[.s] In GNU *troff*, the **.s** register is string-valued; it interpolates the type size in typographical points, which can be represented as a decimal fraction.

\n[.x] Interpolate major version number of the running *troff* formatter. For example, if the version number is 1.23.0, then **\n[.x]** contains 1.

\n[.y] Interpolate minor version number of the running *troff* formatter. For example, if the version number is 1.23.0, then **\n[.y]** contains 23. Also see **\n[.Y]** below.

New registers

GNU *troff* exposes more formatter state via many new read-only registers. Their names often correspond to the requests that affect them.

\n[.br] Within a macro definition, interpolate 1 if the macro is called with the "normal" control character (".") by default), and 0 otherwise. This facility allows requests to be reliably wrapped by a macro. Interpolating the **.br** register outside of a macro definition makes no sense.

\n[.C] Interpolate 1 if AT&T *troff* compatibility mode is in effect, 0 otherwise. See **cp**.

\n[.cdp] Interpolate depth of last glyph added to the environment. It is positive if the glyph extends below the baseline.

\n[.ce] Interpolate count of input lines remaining to be centered in the environment.

\n[.cht] Interpolate height of last glyph added to the environment. It is positive if the glyph extends above the baseline.

\n[.color] Interpolate 1 if color output is enabled, 0 otherwise.

\n[.cp] Within a "**do**" request, interpolate the saved value of compatibility mode (see **\n[.C]** above).

\n[.csk] Interpolate skew of last glyph added to the environment. The *skew* of a glyph is how far to the right of the center of a glyph the center of an accent over that glyph is to be placed.

\n[.ev] Interpolate name of current environment. This is a string-valued register.

\n[.fam] Interpolate name of the environment's default font family. This is a string-valued register.

\n[.fn] Interpolate resolved name of the font selected in the environment. This is a string-valued register.

\n[.fp] Interpolate next free non-zero font mounting position.

\n[.g] Interpolate 1. Test with "**if**" or **ie** to check whether GNU *troff* is the formatter.

\n[.height] Interpolate the rescaled height of the environment's selected font, in scaled points. It is zero if the font height is not rescaled. See **\H**.

\n[.hla] Interpolate hyphenation language of the environment. This is a string-valued register.

\n[.hlc] Interpolate count of immediately preceding consecutive hyphenated lines in the environment.

\n[.hlm] Interpolate maximum number of consecutive hyphenated lines allowed in the environment.

\n[.hy] Interpolate automatic hyphenation mode of the environment.

\n[.hydefault] Interpolate hyphenation mode default of the environment.

\n[.hym] Inteprolate hyphenation margin of the environment.

\n[.hys] Interpolate hyphenation space adjustment threshold of the environment.

\n[.in] Interpolate indentation amount applicable to the output line pending in the environment.

\n[.int] Interpolate 1 if the text most recently formatted in the environment was "interrupted" or continued with **\c**, 0 otherwise.

\n[.it] Interpolate count of input lines remaining in the environment's pending input trap.

\n[.itc] Interpolate 1 if the environment's pending input trap honors the output line continuation escape sequence (**\c**), 0 otherwise.

\n[.itm] Interpolate the name of the macro associated with the environment's pending input trap. This is a string-valued register.

\n[.kern] Interpolate 1 if pairwise kerning is enabled, 0 otherwise.

\n[.lg] Interpolate ligature mode.

\n[.linetabs] Interpolate 1 if line-tabs mode is enabled in the environment, 0 otherwise.

\n[.ll] Interpolate line length applicable to the environment's pending output line.

\n[.lt] Interpolate the environment's title line length.

\n[.m] Interpolate name of the environment's selected stroke color. This is a string-valued register.

\n[.M] Interpolate name of the environment's selected fill color. This is a string-valued register.

\n[.ne] Interpolate amount of space demanded by the most recent **ne** request that sprang a page location trap. See **\n[.trunc]**.

\n[.nm] Interpolate 1 if output line numbering is enabled in the environment (even if temporarily suppressed), 0 otherwise.

\n[.nn] Interpolate count of lines remaining in the environment for which numbering is suppressed while output line numbering is enabled.

\n[.ns] Interpolate 1 if no-space mode is enabled, 0 otherwise.

\n[.O] Interpolate output suppression level. See **\O**.

\n[.P] Interpolate 1 if the current page is selected for output, 0 otherwise. See **−o** command-line option to *troff* (1).

\n[.pe] Interpolate 1 during page ejection, 0 otherwise.

\n[.pn] Interpolate next page number (either that set by **pn**, or that of the current page plus 1).

\n[.ps] Interpolate the environment's type size in scaled points.

\n[.psr] Interpolate the environment's most recently requested type size in scaled points.

\n[.pvs] Interpolate the environment's post-vertical line spacing amount.

\n[.rj] Interpolate count of input lines remaining to be right-aligned in the environment.

\n[.slant] Interpolate slant in degrees of the environment's selected font. See **\S**.

\n[.sr] Interpolate the environment's most recently requested type size in typographical points. This is a string-valued register.

\n[.ss]

\n[.sss] Interpolate values of the environment's minimum inter-word space and supplemental inter-sentence space, respectively, in twelfths of the space width of the selected font.

\n[.sty] Interpolate the environment's selected abstract font style, if any. This is a string-valued register.

\n[.tabs] Interpolate the environment's tab stop settings (if any) in a form suitable for passage to the **ta** request. This is a string-valued register.

\n[.trap] Interpolate the name of the next vertical position trap after the vertical drawing position. This is a string-valued register.

\n[.trunc] Interpolate amount of vertical space truncated by the most recently sprung page location trap, or, if the trap was sprung by an **ne** request, minus the amount of vertical motion produced by the **ne** request. In other words, at the point a trap is sprung, **\n[.trunc]** represents the difference of what the vertical position would have been but for the trap, and what the vertical position actually is. See **\n[.ne]**.

\n[.U] Interpolate 1 if in unsafe mode, 0 otherwise. See **−U** command-line option to *troff* (1).

\n[.vpt] Interpolate 1 if vertical position traps are enabled, 0 otherwise.

\n[.warn] Interpolate warning mask. See section "Warnings" of *troff* (1).

\n[.Y] Interpolate revision number of the running *troff* formatter. For example, if the version number is 1.23.0, then **\n[.Y]** contains 0. Also see **\n[.x]** and **\n[.y]** above.

\n[.zoom] Interpolate magnification of the environment's selected font, in thousandths, or 0 if magnification unused. See **fzoom**.

The following (writable) registers are set by the **psbb** request.

\n[llx]
\n[lly]
\n[urx]
\n[ury] Interpolate the (upper, lower, left, right) bounding box values (in PostScript units) of the most recently processed PostScript image.

The following (writable) registers are set by the **\w** escape sequence.

\n[rst]
\n[rsb] Like **\n[st]** and **\n[sb]**, but taking account of the heights and depths of glyphs. In other words, these registers store the highest and lowest vertical positions attained by the argument formatted by the **\w** escape sequence, doing what AT&T *troff* documented **\n[st]** and **\n[sb]** as doing.

\n[ssc] The amount of (possibly negative) horizontal space to add to the last glyph before a subscript.

\n[skw] How far to right of the center of the last glyph in the **\w** argument, to place the center of an accent from a roman font over that glyph.

Other writable registers are as follows. Those relating to date and time are initialized using *localtime*(3) at formatter startup.

\n[c.] Interpolate input line number. **\n[.c]** is a read-only alias of this register.

\n[hours] Interpolate number of hours elapsed since midnight.

\n[hp] Interpolate horizontal position relative to that at the start of the input line.

\n[lsn]
\n[lss] Interpolate count of leading spaces on input line and amount of corresponding horizontal motion, respectively.

\n[minutes] Interpolate number of minutes elapsed in the hour.

\n[seconds] Interpolate number of seconds elapsed in the minute.

\n[systat] Interpolate return value of *system*(3) function executed by most recent **sy** request.

\n[slimit] Interpolates maximum quantity of objects on *troff*'s internal input stack (default: 1000). If non-positive, there is no limit: recursion can continue until program memory is exhausted.

\n[year] Interpolate Gregorian year. AT&T *troff*'s **\n[yr]** interpolates the Gregorian year minus 1900.

Delimiters

AT&T *troff* recognized slightly varying sets of delimiters when expecting numerical expressions (as with the **\h** escape sequence), string expressions (as with the **\w** escape sequence and **tl** request), and output comparisons (as in "**.if #foo#bar# .tm match**"). GNU *troff*, when not in compatibility mode, recognizes a single consistent set of delimiters. Compatibility mode emulates AT&T *troff* only up to a point. GNU *troff* accepts leaders and tabs as delimiters, as well as Control+D (EOT or EOF), Control+H (BS or backspace), and Control+L (FF or form feed), all of which, when used as delimiters, cause AT&T *troff* to behave in ways difficult to predict.

Miscellaneous

A font not listed in the output device's *DESC* file's **fonts** directive is automatically mounted at the next available font position when it is selected. If you mount a font explicitly with the **fp** request, you should do so on the first unused position, which can be found in the **.fp** register.

Unparameterized string interpolation does not conceal the arguments to a macro being interpreted. Thus, in a macro definition, the call of another macro with the existing argument list,

```
.xx \\$@
```

is more efficiently done with

```
\\*[xx]\\
```

(that is, with string interpolation). The trailing backslashes prevent the final newline in the macro definition from being interpolated, potentially putting an unwanted blank line on the output. See section "Punning Names" in *groff*(7).

If a font description file contains pairwise kerning information, glyphs from that font are kerned. Kerning between two glyphs can be inhibited by placing a dummy character **\&** between them.

GNU *troff* keeps track of the nesting depth of escape sequence interpolations and other uses of delimiters, as in the **tl** request and the output comparison operator (that is, input like **'foo'bar'** as a conditional expression), so the only characters you need to avoid using as delimiters are those that appear in the arguments you input, not any that result from interpolation. Typically, **'** works fine. Use visible characters as delimiters in GNU *troff*, not US-ASCII controls like BEL (Control+G). The implementation of **\$@** ensures that the double quotes surrounding an argument appear at an interpolation depth different from that of the arguments themselves. Similarly, in bracket-form escape sequences like **\f[ZCMI],** a right bracket **]** does not end the sequence unless it occurs at the same interpolation depth as the opening **[**. In compatibility mode, no attention is paid to the interpolation depth.

In GNU *troff*, the **tr** request can map characters to the unbreakable space escape sequence **\~** as a special case (**tr** normally operates only on *characters*). This feature replaces the odd-parity **tr** mapping trick used in AT&T *troff* documents, where a character, often **~**, was "sacrificed" by mapping it to "nothing", drafting it into use as an unadjustable, unbreakable space. (This feature was gratuitous even in early AT&T *troff,* which supported the *****space* escape sequence by 1976.) Often, it makes more sense to use GNU *troff*'s **\~** escape sequence instead, which has been adopted by every other active *troff* implementation except that of Illumos, as well as by the non-*troff* *mandoc*. Translation of a character to **\~** is generally unnecessary, but might be employed to obtain an unbreakable space when the escape character will subsequently be disabled.

GNU *troff* permits tabs and spaces after the first dot on a control line that ends a macro definition.

Formatter output

The page description language output by GNU *troff* is modeled after that used by AT&T *troff* once the latter adopted a device-independent approach in the early 1980s. Only the differences are documented here. For a fuller discussion, see *groff_out*(5).

Glyph and font names can be of arbitrary length; postprocessors should not assume that they are at most two characters. A glyph to be formatted is always drawn from the current font; in contrast to AT&T device-independent *troff*, drivers need not search special fonts to find a glyph.

Units

The argument to the **s** command is in scaled points (units of points/n, where n is the argument to the **sizescale** command in the *DESC* file). The argument to the "**x H**" command is also in scaled points.

Simple commands

If the **tcommand** directive is present in the output device's *DESC* file, GNU *troff* employs the following two commands.

t *xyz*...
: Typeset word *xyz*; that is, set a sequence of ordinary glyphs named x, y, z, ..., terminated by a space or newline; an optional second integer argument is ignored (this allows the formatter to generate an even number of arguments). Each glyph is set at the current drawing position, and the position is then advanced horizontally by the glyph's width. A glyph's width is read from its metrics in the font description file, scaled to the current type size, and rounded to a multiple of the horizontal motion quantum. Use the **C** command to emplace glyphs of special characters.

u *n xyz*...
: Typeset word *xyz* with track kerning. As **t**, but after placing each glyph, the drawing position is further advanced horizontally by n basic units.

New commands implement color support.

mc *cyan magenta yellow*
md
mg *gray*
mk *cyan magenta yellow black*
mr *red green blue*
: Set the components of the stroke color with respect to various color spaces. **md** resets the stroke color to the default value. The arguments are integers in the range 0 to 65535.

A new device control subcommand is available.

x u *n* If n is 1, start underlining of spaces. If n is 0, stop underlining of spaces. This facility is needed for the **cu** request in *nroff* mode and is ignored otherwise.

Extended drawing commands

GNU *pic* does not produce *troff* escape sequences employing these extensions if its **−n** option is given.

Df *n* Set the shade of gray used to fill geometric objects to n, which must be an integer. 0 corresponds to white and 1000 to black. A grayscale ramp spans the two. A value outside this range uses the stroke color as the fill color. The fill color is opaque. Normally the default is black, but some drivers may provide a way of changing this. **Df** is obsolete since 2002, superseded by **DFg** below.

The corresponding **\D'f'** escape sequence should not be used: its argument is rounded to an integer multiple of the horizontal motion quantum, which can limit the precision of n.

DC *d* Draw a filled circle of diameter d with its leftmost point at the drawing position.

DE *h v* Draw a filled ellipse, of horizontal axis h and vertical axis v, with its leftmost point at the drawing position.

Dp dx_1 dy_1 ... dx_n dy_n

Draw a polygon with, for $i = 1, \ldots, n+1$, its ith vertex at the drawing position $+ \sum_{j=1}^{i-1} (dx_j, dy_j)$. *groff* output drivers automatically close polygons, drawing a line from (dx_n, dy_n) back to (dx_1, dy_1). The drawing position is left at the last *specified* vertex, but this may change in a future version of GNU *troff*. Heirloom Doctools *troff*, like DWB *troff*, by default does not close the polygon. In its *groff*

compatibility mode, Heirloom closes the polygon but leaves the drawing position *unchanged*—that is, at the polygon's *initial* drawing position.

DP dx_1 dy_1 ... dx_n dy_n
As **Dp**, but draw a filled rather than a stroked polygon.

Dt *n* Set the line thickness to *n* basic units. AT&T *troff* output drivers use a thickness proportional to the type size; this is the GNU *troff* default. A negative *n* requests this explicitly. An *n* of zero selects the smallest available line thickness.

A difficulty arises in how the drawing position should be changed after the execution of these commands. This has little importance to most users, since the output of GNU *grn* and *pic* does not depend on it. Given a drawing command of the form **D***z* x_1 y_1 ... x_n y_n, where z is not **c** or **e**, AT&T *troff* treats each x_i as a horizontal motion, each y_i as a vertical one, and therefore assumes that the width of the drawn object is $\sum_{i=1}^{n} x_i$, and its height is $\sum_{i=1}^{n} y_i$. (Verify its assumption about height by examining the **st** and **sb** registers after using such a drawing command in a **\w** escape sequence). Thus after executing a **D** command of the form **D***z* x_1 y_1 ... x_n y_n, the drawing position increases by $(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i)$. For the sake of compatibility, GNU *troff* follows this rule, even though it frustrates extensions to the **D** command that set drawing parameters rather than rendering objects, producing ugly results in the case of **Dt** and **Df**, or otherwise don't parameterize objects as a series of vertices, as with GNU *troff*'s filled ellipse, **DE**. In a future release, GNU *troff* and its output drivers may abandon the application of this assumption to drawing commands not explicitly specified in the AT&T "Troff User's Manual". You can ensure predictable output by enclosing drawing commands in the zero-motion escape sequence **\Z**.

GNU *troff* implements fill color selection with another set of extensions.

DFc *cyan magenta yellow*
DFd
DFg *gray*
DFk *cyan magenta yellow black*
DFr *red green blue*
Set the components of the fill color as described under the **\M** escape sequence above. **DFd** restores the device's default fill color. The drawing position is not updated, in contrast to **Df**.

Device control syntax extension

GNU *troff* introduces a line continuation convention, permitting the argument to the **x X** command to contain newlines. A newline in the input is transformed to the sequence "*newline*+". When interpreting an **x X** command, a postprocessor should therefore be prepared for a plus sign after a newline; if it occurs, preserve the newline, discard the plus sign, and continue to collect the input into the argument of the **x X** command. A newline *not* followed by a plus sign terminates the **x X** command. An application of this feature is the embedding of PostScript or PDF language command streams into *troff* output.

GNU *troff* guarantees that the first three output commands it emits are as follows.

```
x T device
x res n h v
x init
```

Debugging

In addition to AT&T *troff*'s debugging features, GNU *troff* emits more error diagnostics when syntactical or semantic nonsense is encountered and supports several warning categories; the output of these can be selected with "**warn**". Also see the **−E**, **−w**, and **−W** options of *troff* (1).

A trace of the formatter's input processing stack can be emitted when errors or warnings occur by means of GNU *troff* **−b** option, or produced on demand with the **backtrace** request.

groff also adds more flexible diagnostic output requests (**tmc** and **tm1**). Examine the state of the formatter with requests that write lists of defined colors (**pcolor**), composite character mappings (**pcomposite**), environments (**pev**), font translations (**pftr**), automatic hyphenation codes (**pchar**) and exceptions (**phw**), registers (**pnr**), open streams (**pstream**), and page location traps (**pwh**). Requests can also disclose to the standard error stream the internal properties and representations of characters and classes (**pchar**), macros (and strings and diversions) (**pm**), and the list of output nodes corresponding to the pending input line (**pline**).

Compatibility mode

Some syntactical and behavioral differences between AT&T and GNU *troff*s are thought too important to neglect; GNU *troff* therefore makes available a *compatibility mode* in an effort to keep documents prepared for AT&T *troff* rendering well.

Identifiers of arbitrary length may be GNU *troff*'s most obvious innovation. AT&T *troff* interprets "**.dsabcd**" as defining a string "**ab**" with contents "**cd**". Normally, GNU *troff* interprets this input as calling a macro named "**dsabcd**". AT&T *troff* also interprets ***[** and **\n[** as interpolating a string or register, respectively, named "[". GNU *troff*, however, normally interprets " [" as bracketing a long name (with "]" at the distal end). In compatibility mode, GNU *troff* interprets names in the traditional way, they thus can be two characters long at most. See the **−C** option in *troff* (1) and, above, the **.C** and **.cp** registers, and **cp** and "**do**" requests, for more on compatibility mode.

The register **\n[.cp]** is specialized and may require a statement of rationale. When writing macro packages or documents that use GNU *troff* features and which may be mixed with other packages or documents that do not—common scenarios include serial processing of man pages or use of the "**so**" or **mso** requests—you may desire correct operation regardless of compatibility mode enablement in the surrounding context. It may occur to you to save the existing value of **\n(.C** into a register, say, **_C**, at the beginning of your file, turn compatibility mode off with "**.cp 0**", then restore it from that register at the end with "**.cp \n(_C**". At the same time, a modular design of a document or macro package may lead you to multiple layers of inclusion. You cannot use the same register name everywhere lest you "clobber" the value from a preceding or enclosing context. The two-character register name space of AT&T *troff* is confining, but employing GNU *troff*'s more capacious one, as with "**.nr _my_saved_C \n(.C**" does not work in compatibility mode; the register name is too long. Employing the "**do**" request is no help: "**.do nr _my_saved_C \n(.C**" always saves zero to the register, because "**do**" turns compatibility mode *off* while it interprets its argument list.

GNU *troff* normally tracks the interpolation depth of escape sequence parameters and other delimited structures, but not in compatibility mode. See section "Miscellaneous" above.

The escape sequences **\f**, **\H**, **\m**, **\M**, **\R**, **\s**, and **\S** are transparent to control character recognition at the beginning of an input line, or after the conditional expression of an "**if**" or **ie** request, only in compatibility mode. That is, upon interpreting them, GNU *troff* normally no longer recognizes a control character on the input line; but in compatibility mode, it does, just like AT&T *troff*.

Normally, the syntax form **\s***n* accepts only a single character (a digit) for *n*, consistently with other forms that originated in AT&T *troff*, like *****, **\$**, **\f**, **\g**, **\k**, **\n**, and **\z**. In compatibility mode only, a non-zero *n* must be in the range 4–39. Legacy documents relying upon this quirk of parsing should migrate to another **\s** form. [Background: The Graphic Systems C/A/T phototypesetter (the original device target for AT&T *troff*) supported only a few discrete type sizes in the range 6–36 points, so Ossanna contrived a special case in the parser to do what the user must have meant. Kernighan warned of this in the 1992 revision of CSTR #54 (§2.3), and more recently, McIlroy referred to it as a "living fossil".]

Other differences

GNU *troff* does not emit output if it has nothing to format. For example, it treats an input document consisting solely of **nr** and **tm** requests as empty, and produces nothing on its standard output stream. AT&T *troff* does, creating a blank page.

Use of C0 control characters in identifiers is not portable; Solaris, Plan 9, and Heirloom Doctools *troff*s accept Control+B, Control+C, Control+E, Control+F, and Control+G (only); DWB 3.3 *troff* does not. GNU *troff* rejects C0 controls in identifiers with an error diagnostic.

Formatters that don't implement GNU *troff* extension request names tend to ignore them, and if they don't support a GNU *troff* extension escape sequence, they are liable to format its function selector character as text. For example, the adjustable, non-breaking space escape sequence **\~** is also supported by Heirloom Doctools *troff* 050915 (September 2005), *mandoc* 1.9.5 (2009-09-21), *neatroff* (commit 1c6ab0f6e, 2016-09-13), and Plan 9 from User Space *troff* (commit 93f8143600, 2022-08-12), but not by Solaris or Documenter's Workbench *troff*s, which both render it as " ~". The **\A** escape sequence (see subsection "Escape sequences" above) may be helpful in avoiding their use.

AT&T *troff* discards trailing spaces from input lines, like GNU *troff*, but when it does so, AT&T *troff* also cancels end-of-sentence detection. Use of the dummy character escape sequence **\&** is more portable.

When adjusting output lines to both margins, AT&T *troff* at first adjusts spaces starting from the right; GNU *troff* begins from the left. Both implementations adjust spaces from opposite ends on alternating output lines in this adjustment mode to prevent "rivers" in the text.

GNU *troff* does not always hyphenate words as AT&T *troff* does. The AT&T implementation uses a set of hard-coded rules specific to U.S. English, while GNU *troff* uses language-specific hyphenation pattern files derived from TEX. Some versions of *troff* reserved meager storage for hyphenation exception words (arguments to the **hw** request); GNU *troff* has no such restriction. When the **hy** request is invoked without an argument, GNU *troff* sets the automatic hyphenation mode to the value of the **.hydefault** register; the AT&T implementation sets it to "**1**", which is not suitable in GNU *troff* for some languages, including English.

Unlike GNU *troff*, AT&T *troff* does not recognize an occurrence of **\%** at the beginning of a word as suppressing its hyphenation; instead, it (uselessly) marks the start of the word as a potential hyphenation point, permitting output lines to end with hyphens that are not interior to a word.

GNU *troff* handles the dummy character **\&** differently from AT&T *troff* when it is followed by the hyphenation control escape sequence **\%** at the beginning of a word. GNU *troff* does not regard the dummy character as "starting" the word; AT&T *troff* does. Further, Heirloom Doctools *troff* does not honor an explicit hyphenation point marked with **\%** after a word-initial one.

GNU *troff* interprets request arguments representing file names and system commands in the same way it does the *contents* argument to the **ds** and "**as**" requests: it removes a leading neutral double quote '**"**' from the argument to the **cf**, **nx**, **pi**, "**so**", and **sy** requests, and the second argument (if present) to the **lf** request, permitting initial embedded spaces in it, and reads it to the end of the input line in copy mode. This difference permits the formatter to handle files with spaces in their names, but requires more care with trailing comments, and doubling of an initial neutral double quote "**"**" if the file name has one.

The existence of the **.T** string is a common feature of device-independent *troff*s—DWB 3.3, Solaris, Heirloom Doctools, and Plan 9 *troff* all support it—but valid values are specific to each implementation.

The (read-only) register **.T** interpolates 1 if GNU *troff* is run with the **−T** option, and 0 otherwise. In contrast, AT&T *troff* interpolated 1 only if *nroff* was the formatter and was run with **−T**.

AT&T *troff* ignored attempts to remove read-only registers; GNU *troff* honors such requests.

The **lf** request sets the number of the *current* input line in AT&T *troff*, and the *next* in GNU *troff*.

AT&T *troff* had only environments named "**0**", "**1**", and "**2**". In GNU *troff*, any number of environments may exist, using any valid identifiers for their names.

As noted above in "Fractional type sizes and new scaling units", AT&T *troff*'s **ps** request ignores scaling units and thus "**.ps 10u**" sets the type size to 10 points, whereas in GNU *troff* it sets the type size to 10 *scaled* points, possibly a much smaller measurement. AT&T's behavior also means that "**.ps 10p**" and "**.ps 10z**" are portable.

The **ab** request differs from AT&T *troff*: GNU *troff* writes no message to the standard error stream if no arguments are given, and it exits with a failure status instead of a successful one.

The **bp** request differs from AT&T *troff*: GNU *troff* does not accept a scaling unit on the argument, a page number; the former does (uselessly).

In AT&T *troff* the **pm** request reports macro, string, and diversion sizes in units of 128-byte blocks, and an argument reduces the report to a sum of the above in the same units. GNU *troff* reports their lengths in characters or nodes if given no arguments, and otherwise dumps the JSON-encoded name, contents, and other properties of each named argument.

AT&T *troff* ignores the **ss** request if the output is a terminal device; GNU *troff* rounds down the values of minimum inter-word and additional inter-sentence space each to the nearest multiple of 12.

GNU *troff* distinguishes characters from glyphs. Characters can be ordinary, special, or indexed, and populate strings and macros. Characters *per se* have not (yet) been formatted. Glyphs represent graphemes (supplied by the output device) and populate diversions. Formatting converts characters into (sequences of) glyphs. GNU *troff* stores properties of the environment that affect how a glyph is rendered with the glyph node's data. Thus, subsequent formatting operations do not affect it, including **bd**, **cs**, **tkf**, **tr**, and **fp** requests. Normally, a macro or string contains only a list of characters and a diversion contains only a list of nodes. However, applying the **asciify** or **unformat** requests to a diversion converts some of its nodes back into characters. Where the formatter cannot recover the character representation of a node, it stores a null character in the character list corresponding to a single node in the node list. Consequently, a glyph node does not behave as a character does in macro interpolation: it does not inherit special properties that the character from which it was constructed might have had.

One way to format a backslash in most documents is with the **\e** escape sequence; this formats the glyph of the current escape character, regardless of whether it is used in a diversion; it also works in both GNU *troff* and AT&T *troff*. (Naturally, if you've changed the escape character, you need to prefix the " **e**" with whatever it is—and you'll likely get something other than a backslash in the output.)

The other correct way, appropriate in contexts independent of the backslash's common use as a *roff* escape character—perhaps in discussion of character sets or other programming languages—is the special character escape sequence **\(rs** or **\[rs]**, for "reverse solidus", from its name in the ECMA-6 and ISO 10646 standards. [AT&T *troff 's* font description files did not define the **rs** special character, but those of its descendant Heirloom Doctools *troff* do, as of its 060716 release (July 2006).]

To store an escape sequence in a diversion that is interpreted when the diversion is interpolated, either use the traditional **\!** transparent output facility, or, if this is unsuitable, the new **\?** escape sequence. See subsection "Escape sequences" above and sections "Diversions" and "GNU troff Internals" in *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual.

Like AT&T *troff,* GNU *troff* maintains a buffer of device-independent output commands, populating the buffer as formatted output accumulates. GNU *troff* always flushes this buffer when processing a break; AT&T *troff* does so according to no obvious schedule (perhaps, if the buffer is of fixed size, the formatter performs the flush when the buffer runs out of room).

In the somewhat pathological case where a diversion exists containing a partially collected line and a partially collected line at the top-level diversion has never existed, AT&T *troff* outputs a partially collected but otherwise empty line (as if "**\c**" were in the top-level diversion) at the end of input; GNU *troff* does not.

Formatter output incompatibilities

Its extensions notwithstanding, GNU *troff*'s page description language has some incompatibilities with that of AT&T *troff*, but better compatibility is sought; problem reports and patches are welcome. The following incompatibilities are known.

- The drawing position after rendering polygons is inconsistent with AT&T *troff* practice. Other implementations have diverged on this point as well.
- The output cannot be easily rescaled to other devices as AT&T *troff*'s could.

Authors

This document was written by James Clark, Werner Lemberg, Bernd Warken, and G. Branden Robinson.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

"Troff User's Manual" by Joseph F. Ossanna, 1976 (revised by Brian W. Kernighan, 1992), AT&T Bell Laboratories Computing Science Technical Report No. 54, widely called simply "CSTR #54", documents the language, device and font description file formats, and page description language referred to collectively in *groff* documentation as AT&T *troff*.

"A Typesetter-independent TROFF" by Brian W. Kernighan, 1982, AT&T Bell Laboratories Computing Science Technical Report No. 97, provides additional insights into the device and font description file formats and page description language.

groff (1), *groff* (7), *roff* (7)

Name

groff_hdtbl – Heidelberger table macros for GNU *roff*

Description

The *hdtbl* macros consist of four base and three optional macros, controlled by about twenty arguments. The syntax is simple and similar to the HTML table model and nearly as flexible: you can write sequences of tokens (macro calls with their arguments and content data), separated by blanks and beginning with a macro call, into the same line to get compact and cleanly arranged input. An advantage of *hdtbl* is that the tables are constructed without calling a preprocessor; this means that *groff*(7)'s full macro capabilities are available. On the other hand, table processing with *hdtbl* is much slower than using the *tbl*(1) preprocessor. A further advantage is that the HTML-like syntax of *hdtbl* will be easily converted to HTML; this is not implemented yet.

Usage

In this and the next section, we present examples to help users understand the basic workflow of *hdtbl*. First of all, you must load the *hdtbl.tmac* file. As with nearly all other *groff* macro packages, there are two possibilities to do so: Either add the line

```
.mso hdtbl.tmac
```

to your *roff* file before using any macros of the *hdtbl* package, or add the option

```
-m hdtbl
```

to the command line of groff (before the document file which contains *hdtbl* macros). Then you can include on or more tables in your document, where each one must be started and ended with the `.TBL` and `.ETB` macros, respectively.

In this man page, we approximate the result of each example as terminal output to be as generic as possible since *hdtbl* currently only supports the **ps** and **pdf** output drivers.

The simplest well-formed table consists of just single calls to the four base table macros in the right order. Here we construct a table with only one cell.

```
.TBL
.TR
.TD
contents of the table cell
.ETB
```

A terminal representation is

```
+-----------------------------------------------------+
| contents-of-the-table-cell                          |
+-----------------------------------------------------+
```

Equivalent to the above is the following notation.

```
.TBL .TR .TD "contents of the table cell" .ETB
```

By default, the formatted table is inserted into the surrounding text at the place of its definition. If the vertical space isn't sufficient, it is placed at the top of the next page. Tables can also be stored for later insertion.

Using '*row-number*`*`*column-number*' as the data for the table cells, a table with two rows and two columns can be written as

```
.TBL cols=2
.  TR .TD 1*1 .TD 1*2
.  TR .TD 2*1 .TD 2*2
.ETB
```

A terminal representation is

```
+-------------------------+-------------------------+
| 1*1                     | 1*2                     |
+-------------------------+-------------------------+
| 2*1                     | 2*2                     |
+-------------------------+-------------------------+
```

Here we see a difference from HTML tables: The number of columns must be explicitly specified using the '`cols=`*m*' argument (or indirectly via the '`width`' argument, see below).

The contents of a table cell is arbitrary; for example, it can be another table, without restriction to the nesting depth. A given table layout can be either constructed with suitably nested tables or with proper arguments to `.TD` and `.TH`, controlling column and row spanning. Note, however, that this table

```
.TBL
.  TR
.    TD
.      nop 1*1 1*2
.  TR
.    TD
.      TBL cols=2 border=
.        TR
.          TD
.            nop 2*1
.          TD
.            nop 2*2
.      ETB
.ETB
```

and this table

```
.TBL cols=2
.  TR
.    TD colspan=2
.      nop 1*1 1*2
.  TR
.    TD
.      nop 2*1
.    TD
.      nop 2*2
.ETB
```

are similar but not identical (the use of `.nop` is purely cosmetic to get proper indentation).

The first table looks like

```
+-----------------------------------------------------+
| 1*1 1*2                                             |
+-----------------------------------------------------+
|                                                     |
| 2*1                         2*2                     |
|                                                     |
+-----------------------------------------------------+
```

and the second one like

```
+-----------------------------------------------------+
| 1*1 1*2                                             |
+--------------------------+--------------------------+
| 2*1                      | 2*2                      |
+--------------------------+--------------------------+
```

Here is the latter table in a more compact form.

```
.TBL cols=2 .TR ".TD colspan=2" 1*1 1*2
.            TR .TD 2*1 .TD 2*2 .ETB
```

If a macro has one or more arguments (see below), and it is not starting a line, everything belonging to this macro including the macro itself must be enclosed in double quotes.

Macros and arguments

The order of macro calls and other tokens follows the HTML model. In the following list, valid predecessors and successors of all *hdtbl* macros are given, together with the possible arguments.

Macro arguments are separated by blanks. The order of arguments is arbitrary; they are of the form

`key=`*`value`*

or

`key='`*`value1`* `[`*`value2`* `[...]]'`

with the only exception of the optional argument of the macro `.ETB`, which is the string '`hold`'. Another possible form is

`"key=`*`value1`* `[`*`value2`* `[...]]"`

However, this is limited to the case where the macro is the first one in the line and not already enclosed in double quotes.

Argument values specified below as *c* are colors predefined by *groff* or colors defined by the user with the `.defcolor` request. Argument values *d* are decimal numbers with or without decimal point. Argument values *m* are natural numbers. Argument values *n* are numerical values with the usual *groff* scaling indicators. Some of the arguments are specific to one or two macros, but most of them can be specified with `.TBL`, `.TR`, `.TD`, and `.TH`. These common arguments are explained in the next subsection.

Most of the argument default values can be changed by the user by setting corresponding default registers or strings, as listed below.

`.TBL` [*args*]

Begin a new table.

predecessor: `.TD`, `.TH`, `.ETB`, `cell contents`
successor: `.CPTN`, `.TR`
arguments:

`border=`[*n*]
: Thickness of the surrounding box border. '`border=`' (no value) means neither a surrounding box border nor any horizontal or vertical separator lines between the table rows and cells. '`border=0`' suppresses the surrounding box border, but still allows separator lines between cells and rows.
Default: '`border=.1n`' (register '`t*b`').

`bc=`*c*
: Border color.
Default: '`bc=red4`' (string '`t*bc`').

`cols=`*m*
: Number of table columns. This argument is necessary if more than one column is in the table and no '`width`' arguments are present.
Default: '`cols=1`' (register '`t*cols`').

`cpd=`*n*
: Cell padding, i.e., the extra space between the cell space border and the cell contents.
Default: '`cpd=.5n`' (register '`t*cpd`').

`csp=`*n*
: Cell spacing, i.e., the extra space between the table border or vertical or horizontal lines between cells and the cellspace.
Default: '`csp=.5n`' (register '`t*csp`').

`tal=l|c|r`
: Horizontal alignment of the table, if it is smaller than the line width. '`tal=l`': left alignment. '`tal=c`': centered alignment. '`tal=r`': right alignment.
Default: '`tal=l`' (register '`t*tal`').

`width='`*w1* [*w2* [...]]`'`
: Widths of table cells. *w1*, *w2*, ... are either numbers of type *n* or natural numbers with the pseudo-scaling indicator '`%`', with the meaning "percent of the actual line length (or column length for inner tables, respectively)". If there are less width values than table columns, the last width value is used for the remaining cells. The argument

```
width='1.5i 10%'
```

for example indicates that the first column is 1.5 inches wide; the remaining columns take 1/10 of the column length each.
Default: The table width equals the outer line length or column length; the columns have equal widths.

`height=`*n*
: Height of the table. If the table with its contents is lower than *n*, the last row is stretched to this value.

.CPTN [*args*]
: Text of caption.

The (optionally numbered) table caption. `.CPTN` is optional.

predecessor: `.TBL`
successor: `.TR`
arguments:

`val=t|b`
: Vertical alignment of the table caption. '`val=t`': The caption is placed above the table. '`val=b`': The caption is placed below the table.
Default: '`val=t`' (string '`t*cptn`').

.TR [*args*]
: Begin a new table row.

predecessor: `.TBL`, `.CPTN`, `.TD`, `.TH`, `.ETB`, `cell contents`
successor: `.TD`, `.TH`
arguments:

`height=`*n*
: The height of the row. If a cell in the row is higher than *n*, this value is ignored; otherwise the row height is stretched to *n*.

.TD [*args* [*cell contents*]]
: Begin a table data cell.

.TH [*args* [*cell contents*]]
Begin a table header cell.

Arguments and cell contents can be mixed. The macro `.TH` is not really necessary and differs from `.TD` only in three default settings, similar to the <TH> and <TD> HTML tags: The contents of `.TH` is horizontally and vertically centered and typeset in boldface.

predecessor: `.TR`, `.TD`, `.TH`, `.ETB`, `cell contents`
successor: `.TD`, `.TH`, `.TR`, `.ETB`, `cell contents`
arguments:

`colspan=`*m*
The width of this cell is the sum of the widths of the *m* cells above and below this row.

`rowspan=`*m*
The height of this cell is the sum of the heights of the *m* cells left and right of this column.

Remark: Overlapping of column and row spanning, as in the following table fragment (the overlapping happens in the second cell in the second row), is invalid and causes incorrect results.

```
.TR .TD 1*1 ".TD 1*2 rowspan=2" .TD 1*3
.TR ".TD 2*1 colspan=2"         .TD 2*3
```

A working example for headers and cells with **colspan** is

```
.TBL cols=3
.  TR ".TH colspan=2" header1+2 .TH header3
.  TR .TD 1*1 .TD 1*2 .TD 1*3
.  TR .TD 2*1 ".TD colspan=2" 2*2+3
.ETB
```

This looks like

```
+------------------------------+---------------+
|          header1+2           |    header3    |
+--------------+---------------+---------------+
| 1*1          | 1*2           | 1*3           |
+--------------+---------------+---------------+
| 2*1          | 2*2+3                         |
+--------------+-------------------------------+
```

A working example with **rowspan** is

```
.TBL cols=3
.  TR
.  TD 1*1
.  TD rowspan=2 1+2*2
.  TD 1*3
.
.  TR
.  TD 2*1
.  TD 2*3
.ETB
```

which looks like

```
+--------------+---------------+---------------+
| 1*1          | 1+2*2         | 1*3           |
+--------------+               +---------------+
| 2*1          |               | 2*3           |
+--------------+---------------+---------------+
```

.ETB [**hold**]
End of the table.

This macro finishes a table. It causes one of the following actions.

- If the argument '`hold`' is given, the table is held until it is freed by calling the macro `.t*free`, which in turn prints the table immediately, either at the current position or at the top of the next page if its height is larger than the remaining space on the page.
- Otherwise, if the table is higher than the remaining space on the page, it is printed at the top of the next page.
- If neither of the two above constraints hold, the table is printed immediately at the place of its definition.

predecessor: `.TD`, `.TH`, `.ETB`, `cell contents`
successor: `.TBL`, `.TR`, `.TD`, `.TH`, `.ETB`, `cell contents`
arguments:

`hold` Prevent the table from being printed until it is freed by calling the macro `.t*free`. This argument is ignored for inner (nested) tables.

.t*free [*n*]
Free the next held table or *n* held tables. Call this utility macro to print tables which are held by using the '`hold`' argument of the `.ETB` macro.

Arguments common to .TBL, .TR, .TD, and .TH

The arguments described in this section can be specified with the `.TBL` and `.TR` macros, but they are eventually passed on to the table cells. If omitted, the defaults take place, which the user can change by setting the corresponding default registers or strings, as documented below. Setting an argument with the `.TBL` macro has the same effect as setting it for all rows in the table. Setting an argument with a `.TR` macro has the same effect as setting it for all the `.TH` or `.TD` macro in this row.

`bgc=`[*c*]
The background color of the table cells. This includes the area specified with the '`csp`' argument. The argument '`bgc=`' (no value) suppresses a background color; this makes the background transparent.
Default: '`bgc=bisque`' (string '`t*bgc`').

`fgc=`*c*
The foreground color of the cell contents.
Default: '`fgc=red4`' (string '`t*fgc`').

`ff=`*name*
The font family for the table. *name* is a *groff* font family identifier, such as `A` for Avant Garde or `HN` for Helvetica Narrow.
Default: The font family found before the table (string '`t*ff`').

`fst=`*style*
The font style for the table. One of `R`, `B`, `I`, or `BI` for roman, **bold**, *italic*, or ***bold italic***, respectively. As with *roff*'s **.ft** request, the '`fst`' argument can be used to specify the font family and font style together, for example '`fst=HNBI`' instead of '`ff=HN`' and '`fst=BI`'.
Default: The font style in use right before the table (string '`t*fst`').

`fsz='`*d1* [*d2*]`'`
: A decimal or fractional factor *d1*, by which the point size for the table is changed, and *d2*, by which the vertical line spacing is changed. If *d2* is omitted, value *d1* is taken for both.
Default: '`fsz='1.0 1.0' `' (string '`t*fsz`').

`hal=l|c|b|r`
: Horizontal alignment of the cell contents in the table. '`hal=l`': left alignment. '`hal=c`': centered alignment. '`hal=b`': both (left and right) alignment. '`hal=r`': right alignment.
Default: '`hal=b`' (string '`t*hal`').

`val=t|m|b`
: Vertical alignment of the cell contents in the table for cells lower than the current row. '`val=t`': alignment below the top of the cell. '`val=m`': alignment in the middle of the cell. '`val=b`': alignment above the cell bottom.
Default: '`val=t`' (string '`t*val`').

`hl=[s|d]`
: Horizontal line between the rows. If specified with `.TD` or `.TH` this is a separator line to the cell below. '`hl=`' (no value): no separator line. '`hl=s`': a single separator line between the rows. '`hl=d`': a double separator line.

  The thickness of the separator lines is the half of the border thickness, but at least 0.1 inches. The distance between the double lines is equal to the line thickness.

  Remark: Together with '`border=0`' for proper formatting the value of '`csp`' must be at least .05 inches for single separator lines and .15 inches for double separator lines.
  Default: '`hl=s`' (string '`t*hl`').

`vl=[s|d]`
: Vertical separator line between the cells. If specified with `.TD` or `.TH` this is a separator line to the cell on the right. '`vl=s`': a single separator line between the cells. '`vl=d`': a double separator line. '`vl=`' (no value): no vertical cell separator lines. For more information see the documentation of the '`hl`' argument above.
Default: '`vl=s`' (string '`t*vl`').

hdtbl customization

Before creating the first table, you should configure default values to minimize the markup needed in each table. The following example sets up defaults suitable for typical papers:

```
.ds t*bgc white\" background color
.ds t*fgc black\" foreground color
.ds t*bc black\"  border color
.nr t*cpd 0.1n\"  cell padding
```

The file */usr/share/doc/groff/examples/hdtbl/common.roff* provides another example setup in the "minimal Page setup" section.

A table which does not fit on a partially filled page is printed automatically on the top of the next page if you append the little utility macro `t*hm` to the page header macro of your document's main macro package. For example, say

```
.am pg@top
.  t*hm
..
```

if you use the *ms* macro package.

The macro `t*EM` checks for held or kept tables, and for missing `ETB` macros (table not closed). You can call this macro by appending it the to end-of-input macro of the main, or "full-service", macro package your document uses. For example, try

```
.am pg@end-text
```

```
.  t*EM
..
```

if you use the *ms* package.

Bugs and suggestions

Please send your comments to the *groff* mailing list or directly to the author.

Authors

The *hdtbl* macro package was written by Joachim Walsdorff.

See also

groff(1)
: provides an overview of GNU *roff* and details how to invoke *groff* at the command line.

groff(7)
: summarizes the *roff* language and GNU extensions to it.

tbl(1) describes the traditional *roff* preprocessor for tables.

Name

groff_man – compose manual pages with GNU *roff*

Synopsis

groff –man [*option* . . .] [*file* . . .]
groff –m man [*option* . . .] [*file* . . .]

Description

The GNU implementation of the *man* macro package is part of the *groff* document formatting system. It is used to compose manual pages ("man pages") like the one you are reading. This document presents the macros thematically; for those needing only a quick reference, the following table lists them alphabetically, with references to appropriate subsections below.

Readers who are not already experienced *groff* users should consult *groff_man_style*(7), an expanded version of this document, for additional explanations and advice. It covers only those concepts required for man page document maintenance, and not the full breadth of the *groff* typesetting system.

Macro	Meaning	Subsection
.B	Bold	Font style macros
.BI	Bold, italic alternating	Font style macros
.BR	Bold, roman alternating	Font style macros
.EE	Example end	Document structure macros
.EX	Example begin	Document structure macros
.HP	Begin hanging paragraph	Paragraphing macros
.I	Italic	Font style macros
.IB	Italic, bold alternating	Font style macros
.IP	Indented paragraph	Paragraphing macros
.IR	Italic, roman alternating	Font style macros
.LP	Begin paragraph	Paragraphing macros
.ME	Mail-to end	Hyperlink macros
.MR	Man page cross reference	Hyperlink macros
.MT	Mail-to start	Hyperlink macros
.P	Begin paragraph	Paragraphing macros
.PP	Begin paragraph	Paragraphing macros
.RB	Roman, bold alternating	Font style macros
.RE	Relative inset end	Document structure macros
.RI	Roman, italic alternating	Font style macros
.RS	Relative inset start	Document structure macros
.SH	Section heading	Document structure macros
.SM	Small	Font style macros
.SS	Subsection heading	Document structure macros
.SY	Synopsis start	Synopsis macros
.TH	Title heading	Document structure macros
.TP	Tagged paragraph	Paragraphing macros
.TQ	Supplemental paragraph tag	Paragraphing macros
.UE	URI end	Hyperlink macros
.UR	URI start	Hyperlink macros
.YS	Synopsis end	Synopsis macros

We discuss other macros (**AT**, **DT**, **OP**, **PD**, **SB**, and **UC**) in subsection "Deprecated features" below.

Throughout Unix documentation, a manual entry is referred to simply as a "man page", regardless of its length, without gendered implication, and irrespective of the macro package selected for its composition.

Macro reference preliminaries

A tagged paragraph describes each macro. We present coupled pairs together, as with **EX** and **EE**. If you require an empty macro argument, specify it as a pair of neutral double quotes (**""**). Most macro arguments are formatted as text in the output; exceptions are noted.

Document structure macros

Document structure macros organize a man page's content. All of them break the output line. **TH** (title heading) identifies the document as a man page and configures the page headers and footers. Section headings (**SH**), one of which is mandatory and many of which are conventionally expected, facilitate location of material by the reader and aid the man page writer to discuss all essential aspects of the topics presented. Subsection headings (**SS**) are optional and permit sections that grow long to develop in a controlled way. Many technical discussions benefit from examples; lengthy ones, especially those reflecting multiple lines of input to or output from the system, are usefully bracketed by **EX** and **EE**. When none of the foregoing meets a structural demand, use **RS/RE** to inset a region within a (sub)section.

.TH *identifier section* [*footer-middle* [*footer-inside* [*header-middle*]]]

Break the page, reset the page number to 1 (unless the **−rC1** option is given), and use the arguments to populate the page header and footer. Together, *identifier* and the *section* of the manual to which it belongs can uniquely identify a *man* document on the system. See *man*(1) or *intro*(1) for the manual sectioning applicable to your system. *identifier* and *section* are positioned at the left and right in the header; the latter is set after the former, in parentheses and without space. *footer-middle* is centered in the footer. By default, *footer-inside* is positioned at the bottom left. Use of the double-sided layout option **−rD1** places *footer-inside* at the bottom left on recto (odd-numbered) pages, and the bottom right on verso (even-numbered) pages. By default, the outside footer is the page number. Use of the continuous-rendering option **−rcR=1** replaces it with *identifier* and *section,* as in the header. *header-middle* is centered in the header. If *section* is an integer between 1 and 9 (inclusive), there is no need to specify *header-middle; an.tmac* supplies text for it. If *identifier* or *footer-inside* would overrun the space available in the header and/or footer, this package may abbreviate them with ellipses. *groff man* suppresses headers and footers in HTML output.

A valid *man* document calls **TH** only once, early in the file, prior to any other macro calls.

.SH [*heading-text*]

Set *heading-text* as a section heading. Given no argument, **SH** plants a one-line input trap; text on the next line becomes *heading-text.* The heading text is set in bold (or the font specified by the string **HF**), and, on typesetters, slightly larger than the base type size. If the heading font ***[HF]** is bold, use of an italic style in *heading-text* is mapped to the bold-italic style if available in the font family. The inset level is reset to 1; see subsection "Horizontal and vertical spacing" below. Text lines after the call are set as an ordinary paragraph (**P**).

The content of *heading-text* and ordering of sections follows a set of common practices, as does much of the layout of material within sections. For example, a section called "Name" or "NAME" must exist, must be the first section after the **TH** call, and must contain only text of the form

topic[**,** *another-topic*]... \− *summary-description*

for tools like *makewhatis*(8) or *mandb*(8) to index them.

.SS [*subheading-text*]

Set *subheading-text* as a subsection heading indented between a section heading and an ordinary paragraph (**P**). Given no argument, **SS** plants a one-line input trap; text on the next line becomes *subheading-text.* The subheading text is set in bold (or the font specified by the string **HF**). If the heading font ***[HF]** is bold, use of an italic style in *subheading-text* is mapped to the bold-italic style if available in the font family. The inset level is reset to 1; see subsection "Horizontal and vertical spacing" below. Text lines after the call are set as an ordinary paragraph (**P**).

.EX
.EE Begin and end example. After **EX**, filling is disabled (and, on typesetters, a monospaced font family is selected). Calling **EE** enables filling (and restores the previous family).

Ninth Edition Unix introduced the **EX** and **EE** extensions. Documenter's Workbench (DWB), Heirloom Doctools, and Plan 9 *troff* s, and *mandoc* (since 1.12.2) support them. Solaris *troff* does not.

.RS [*inset-amount*]
: Start new relative inset. *man* saves any current inset amount and moves right by: *inset-amount,* if specified; the *indentation* amount of the preceding **IP**, **TP**, or **HP** macro call if no (sub-)sectioning or ordinary paragraphing macro has intervened; or the amount of the **IN** register. **RS** calls can nest; each increments by 1 the *level* used by **RE**. The level prior to any **RS** call is 1.

.RE [*inset-level*]
: End a relative inset, reducing it to that of *inset-level* (or by 1 if not specified) and restoring the corresponding inset amount.

Paragraphing macros

These macros break the output line. An *ordinary* paragraph (**P**) indents all output lines by the same amount. A hanging paragraph (**HP**) is a cosmetic variant of **P** with a hanging indent. Definition lists frequently occur in man pages; these can be set as *tagged paragraphs,* which have one (**TP**) or more (**TQ**) leading tags followed by a paragraph that has an additional indentation. The indented paragraph (**IP**) macro can continue the indented content of a narrative started with **TP**, or present an itemized or ordered list. If a paragraphing macro has been called since **SH** or **SS**, all except **TQ** follow the break with vertical space (in an amount configured by the deprecated **PD** macro); see subsection "Horizontal and vertical spacing" below. Except for **TQ**, these macros reset the type size, hyphenation, and adjustment to (configured) defaults, and the font style to roman.

.P
.LP
.PP
: Begin a new paragraph; these macros are synonymous. Any indentation from use of **IP**, **TP**, or **HP** is cleared. The inset amount, as affected by **RS** and **RE**, is not.

.HP [*indentation*]
: Set a paragraph with a hanging indentation. Text on output lines after the first is indented by *indentation,* if specified, and by the amount of the **IN** register otherwise.

  Caution: A hanging indentation cannot be expressed naturally in (pure) HTML, a hanging paragraph is not distinguishable from an ordinary one if it formats on only one output line, and non-*roff*-based man page interpreters may treat **HP** as an ordinary paragraph anyway. Thus, information or distinctions you mean to express with indentation may be lost.

.TP [*indentation*]
: Set an indented paragraph with a leading unindented tag. The macro plants a one-line input trap that honors the **\c** escape sequence; text on the next line becomes the tag, set without indentation. Text on subsequent lines is indented by *indentation,* if specified, and by the amount of the **IN** register otherwise. If the tag, plus the "tag spacing" stored in the **TS** register (see section "Options" below) is wider than the indentation, the package breaks the line after the tag.

.TQ
: Set an additional tag for a paragraph tagged with **TP**, planting a one-line input trap as with **TP**.

  TQ is a GNU extension supported by Heirloom Doctools *troff* (since Git snapshot 151218) and *mandoc* (since 1.14.5) but not by DWB, Plan 9, or Solaris *troff*s.

.IP [*mark* [*indentation*]]
: Set an indented paragraph with an optional mark. Arguments, if present, are handled as with **TP**, except that the *mark* argument to **IP** cannot include a macro call, and the tag separation amount stored in the **TS** register is not enforced.

Synopsis macros

Use **SY** and **YS** to summarize syntax using familiar Unix conventions. Heirloom Doctools *troff* (since Git snapshot 151218) and *mandoc* (since 1.14.5) support these GNU extensions; DWB, Plan 9, and Solaris *troff*s do not.

.SY *keyword* [*suffix*]
: Begin synopsis. Adjustment and automatic hyphenation are disabled. If **SY** has already been called without a corresponding **YS**, a break is performed. *keyword* and any *suffix* are set in bold. When *suffix* is present, the package sets the next word after it without intervening space. If a break is

required in subsequent text (up to a paragraphing, sectioning, or **YS** macro call), lines after the first are indented. Unless the previous synopsis's indentation is reused (see **YS** below), output lines after the first indent by the width of the pending output line up to the end of *keyword* plus a space, if *keyword* is the only argument, and up to the end of *suffix* otherwise.

.YS [*reuse-indentation*]
End synopsis, breaking the line and restoring indentation, adjustment, and hyphenation to their previous states. If an argument is given, the indentation corresponding to the previous **SY** call is reused by the next **SY** call instead of being computed.

Hyperlink macros

Man page cross references are best presented with **MR**. Mark email addresses with **MT/ME** and other sorts of URI with **UR/UE**. To hyperlink text, terminals and pager programs must support ECMA-48 OSC 8 escape sequences (see *grotty*(1)). When device support is unavailable or disabled with the **U** register (see section "Options" below), *groff man* renders these URIs between angle brackets (⟨ ⟩) after the linked text.

MT, **ME**, **UR**, and **UE** are GNU extensions supported by Heirloom Doctools *troff* (since Git snapshot 151218) and *mandoc* (**UR/UE** since 1.12.3; **MT/ME** since 1.14.2) but not by DWB, Plan 9 (original), or Solaris *troff*s. Plan 9 from User Space's *troff* implements **MR**.

Prepare arguments to **MR**, **MT**, and **UR** for typesetting; they can appear in the output. Use special character escape sequences to encode Unicode basic Latin characters where necessary, particularly the hyphen-minus.

.MR *topic* [*manual-section* [*trailing-text*]]
(since groff *1.23)* Set a man page cross reference as "*topic*(*manual-section*)". If *manual-section* is absent, the package omits the surrounding parentheses. If *trailing-text* (typically punctuation) is specified, it follows the closing parenthesis without intervening space. Hyphenation is disabled while the cross reference is set. *topic* is set in the font specified by the **MF** string. If *manual-section* is present, the cross reference hyperlinks to a URI of the form "**man:***topic*(*manual-section*)".

.MT *address*
.ME [*trailing-text*]
Identify *address* as an RFC 6068 *addr-spec* for a "mailto:" URI with the text between the two macro calls as the link text. An argument to **ME** is placed after the link text without intervening space. *address* may not be visible in the rendered document if hyperlinks are enabled and supported by the output driver. If they are not, *address* is set in angle brackets after the link text and before *trailing-text.* If hyperlinking is enabled but there is no link text, *address* is formatted and hyperlinked *without* angle brackets, except when *address* appears as a **TP** paragraph tag.

.UR *uri*
.UE [*trailing-text*]
Identify *uri* as an RFC 3986 URI hyperlink with the text between the two macro calls as the link text. An argument to **UE** is placed after the link text without intervening space. *uri* may not be visible in the rendered document if hyperlinks are enabled and supported by the output driver. If they are not, *uri* is set in angle brackets after the link text and before *trailing-text.* If hyperlinking is enabled but there is no link text, *uri* is formatted and hyperlinked *without* angle brackets, except when *uri* appears as a **TP** paragraph tag.

If a **UR/UE** or **MT/ME** pair occurs in a **TP** tag and hyperlinking is unavailable, *groff man* sets the link target at the beginning of the indented paragraph, not as part of the tag, unless there is no link text.

Font style macros

The *man* macro package is limited in its font styling options, offering only **bold** (**B**), *italic* (**I**), and roman. Italic text may instead render underscored on terminals. **SM** sets text at a smaller type size, which differs visually from regular-sized text only on typesetters. The macros **BI**, **BR**, **IB**, **IR**, **RB**, and **RI** set their odd- and even-numbered arguments as text in the alternating styles their names indicate, with no space separating them.

The default type size and family for typesetters is 10-point Times, except on the **X75–12** and **X100–12** devices where the type size is 12 points. The default style is roman.

.B [*text*]
Set *text* in bold. Given no argument, **B** plants a one-line input trap; text on the next line, which can be further formatted with a macro, is set in bold.

.I [*text*]
Set *text* in an italic or oblique face. Given no argument, **I** plants a one-line input trap; text on the next line, which can be further formatted with a macro, is set in an italic or oblique face.

.SM [*text*]
Set *text* one point smaller than the default type size on typesetters. Given no argument, **SM** plants a one-line input trap; text on the next line, which can be further formatted with a macro, is set smaller.

Unlike the above font style macros, the font style alternation macros below set no input traps; they must be given arguments to have effect. They apply italic corrections as appropriate.

.BI *bold-text italic-text* . . .
Set each argument in bold and italics, alternately.

.BR *bold-text roman-text* . . .
Set each argument in bold and roman, alternately.

.IB *italic-text bold-text* . . .
Set each argument in italics and bold, alternately.

.IR *italic-text roman-text* . . .
Set each argument in italics and roman, alternately.

.RB *roman-text bold-text* . . .
Set each argument in roman and bold, alternately.

.RI *roman-text italic-text* . . .
Set each argument in roman and italics, alternately.

Horizontal and vertical spacing

The package sets all text inboard of the left edge of the output medium by the amount of the *page offset;* see register **PO** in section "Options" below. Headers, footers (both set with **TH**), and section headings (**SH**) lie at the page offset. *groff man* indents subsection headings (**SS**) by the amount in the **SN** register.

Ordinary paragraphs not within an **RS**/**RE** inset region are inset by the amount stored in the **BP** register; see section "Options" below. The **IN** register configures the default indentation amount used by **RS** (as the *inset-amount*), **IP**, **TP**, and **HP**; an overriding argument is a number plus an optional scaling unit. If no scaling unit is given, the *man* package assumes "n". An indentation specified in a call to **IP**, **TP**, or **HP** persists until (1) another of these macros is called with an *indentation* argument, or (2) **SH**, **SS**, or **P** or its synonyms is called; these clear the indentation entirely.

Several macros insert vertical space: **SH**, **SS**, **TP**, **P** (and its synonyms), **IP**, and **HP**. They then enable *no-space mode;* see *groff*(7). The default inter-section and inter-paragraph spacing is 1v for terminals and 0.4v for typesetters. (The deprecated macro **PD** can change this vertical spacing, but we discourage its use.) Between **EX** and **EE** calls, the inter-paragraph spacing is 1v regardless of output device.

Registers

Registers are described in section "Options" below. They can be set not only on the command line but in the site *man.local* file as well; see section "Files" below.

Strings

The following strings are defined for use in man pages. None of these is necessary in a contemporary man page; see *groff_man_style*(7). *groff man* supports others for configuration of rendering parameters; see section "Options" below.

***R** interpolates a special character escape sequence for the "registered sign" glyph, **\(rg**, if available, and "(Reg.)" otherwise.

***S** interpolates an escape sequence setting the type size to the document default.

***(lq**
***(rq** interpolate special character escape sequences for left and right double-quotation marks, **\(lq** and **\(rq**, respectively.

***(Tm** interpolates a special character escape sequence for the "trade mark sign" glyph, **\(tm**, if available, and "(TM)" otherwise.

Hooks

Two macros, both GNU extensions, are called internally by the *groff man* package to format page headers and footers and can be redefined by the administrator in a site's *man.local* file (see section "Files" below). The presentation of **TH** above describes the default headers and footers. Because these macros are hooks for *groff man* internals, man pages have no reason to call them. Such hook definitions typically consist of "sp" and "tl" requests. **PT** furthermore has the responsibility of emitting a PDF bookmark after writing the first page header in a document. Consult the existing implementations in *an.tmac* when drafting replacements.

.BT Set the page footer text ("bottom trap").

.PT Set the page header text ("page trap").

To remove a page header or footer entirely, define the appropriate macro as empty rather than deleting it.

Deprecated features

Use of the following in man pages for public distribution is discouraged.

.AT [*system* [*release*]]

Alter the footer for use with legacy AT&T man pages, overriding any definition of the *footer-inside* argument to **TH**. This macro exists only to render man pages from historical systems.

The inside footer is populated per the value of *system.*

3 7th edition *(default)*

4 System III

5 System V

The optional *release* argument specifies the release number, as in "System V Release 3".

.DT Reset tab stops to the default (every 0.5i).

Use of this presentation-oriented macro is deprecated. It translates poorly to HTML, under which exact space control and tabulation are not readily available. Thus, information or distinctions that you use tab stops to express are likely to be lost. If you feel tempted to change the tab stops such that calling this macro later to restore them is desirable, consider composing a table using *tbl*(1) instead.

.OP *option-name* [*option-argument*]

Indicate an optional command parameter called *option-name*, which is set in bold. If the option takes an argument, specify *option-argument* using a noun, abbreviation, or hyphenated noun phrase. If present, *option-argument* is preceded by a space and set in italics. Square brackets in roman surround both arguments.

Use of this quasi-semantic macro, an extension whose name originated in DWB *troff*, is deprecated; *groff*'s interface differs. Neither can easily be used to annotate options that take optional arguments or options whose arguments have internal structure (such as a mixture of literal and variable components). One could work around these limitations with font selection escape sequences, but font style alternation macros are preferable; they are more flexible and perform italic corrections on typesetters.

.PD [*vertical-space*]

Configure the amount of vertical space between paragraphs or (sub)sections. The optional argument *vertical-space* specifies the amount; the default scaling unit is "v". Without an argument, inter-paragraph spacing resets to its default value; see subsection "Horizontal and vertical spacing" above.

Use of this presentation-oriented macro is deprecated. It translates poorly to HTML, under which exact control of inter-paragraph spacing is not readily available. Thus, information or distinctions that you use **PD** to express are likely to be lost.

.SB [*text*]
Set *text* in bold and (on typesetters) one point smaller than the default type size. Given no argument, **SB** plants a one-line input trap; text on the next line, which can be further formatted with a macro, is set smaller and in bold. Use of this macro, an extension originating in SunOS 4.0 *troff*, is deprecated. **SM** without an argument, followed immediately by "**B** *text*", produces the same output more portably. The macros' order is interchangeable; put *text* with the latter.

.UC [*version*]
Alter the footer for use with legacy BSD man pages, overriding any definition of the *footer-inside* argument to **TH**. This macro exists only to render man pages from historical systems.

The inside footer is populated per the value of *version.*

3	3rd Berkeley Distribution *(default)*
4	4th Berkeley Distribution
5	4.2 Berkeley Distribution
6	4.3 Berkeley Distribution
7	4.4 Berkeley Distribution

History

M. Douglas McIlroy designed, implemented, and documented the AT&T *man* macros for Unix Version 7 (1979) and employed them to edit Volume 1 of its *Programmer's Manual*, a compilation of all man pages supplied by the system. The package supported the macros listed in this page not described as extensions, except **P** and the deprecated **AT** and **UC**. It documented no registers and defined only **R** and **S** strings.

UC appeared in 3BSD (1980). Unix System III (1980) introduced **P** and exposed the registers **IN** and **LL**, which had been internal to Seventh Edition Unix *man*. PWB/Unix 2.0 (1980) added the **Tm** string. 4BSD (1980) added **lq** and **rq** strings. SunOS 2.0 (1985) recognized **C**, **D**, **P**, and **X** registers. 4.3BSD (1986) added **AT** and **P**. Ninth Edition Unix (1986) introduced **EX** and **EE**. SunOS 4.0 (1988) added **SB**. Unix System V (1988) incorporated the **lq** and **rq** strings.

Except for **EX/EE**, James Clark implemented the foregoing features in early versions of *groff*. Later, *groff* 1.20 (2009) resurrected **EX/EE** and originated **SY/YS**, **TQ**, **MT/ME**, and **UR/UE**. Plan 9 from User Space's *troff* introduced **MR** in 2020, and incorporated the **lq** and **rq** strings in 2025.

Options

The following *groff* options set registers (with **−r**) and strings (with **−d**) recognized and used by the *man* macro package. To ensure rendering consistent with output device capabilities and reader preferences, man pages should never manipulate them.

−dAD=*adjustment-mode*
Set line adjustment to *adjustment-mode,* which is typically "**b**" for adjustment to both margins (the default), or "**l**" for left alignment (ragged right margin). Any valid argument to *groff*'s "ad" request may be used. See *groff*(7) for less-common choices.

−rBP=*base-paragraph-inset*
Set the inset amount for ordinary paragraphs not within an **RS/RE** inset. The default is 5n.

−rcR=1 Enable continuous rendering. Output is not paginated; instead, one (potentially very long) page is produced. This is the default for terminal and HTML devices. Use **−rcR=0** to disable it on terminals; on HTML devices, it cannot be disabled.

−rC1 Number output pages consecutively, in strictly increasing sequence, rather than resetting the page number to 1 (or the value of register **P**) with each new *man* document.

–rCHECKSTYLE=*n*
: Report problems with usage of this macro package exhibited by the input at verbosity level *n*, where *n* is an integer in the range 0–3, inclusive; **0** disables the messages and is the default. This feature is a development and debugging aid for man page maintainers; the problems diagnosed, and range and meanings of the supported levels, are subject to change.

–rCS=1
: Set section headings (the argument(s) to **SH**) in full capitals. This transformation is off by default because it discards lettercase distinctions.

–rCT=1
: Set the man page identifier (the first argument to **TH**) in full capitals in headers and footers. This transformation is off by default because it discards lettercase distinctions.

–rD1
: Enable double-sided layout, formatting footers for even and odd pages differently; see the description of **TH** in subsection "Document structure macros" above.

–rFT=*footer-distance*
: Set distance of the footer relative to the bottom of the page to *footer-distance;* this amount is always negative. At one half-inch above this location, the page text is broken before writing the footer. Ignored if continuous rendering is enabled. The default is "–0.5i – 1v".

–dHF=*heading-font*
: Select the font used for section and subsection headings; the default is "**B**" (bold style of the default family). Any valid argument to *groff*'s "ft" request may be used. See *groff*(7).

–rHY=0
: Disable automatic hyphenation. Normally, it is enabled (1). The hyphenation mode is determined by the *groff* locale; see section "Localization" of *groff*(7).

–rIN=*standard-indentation*
: Set the default indentation amount used by **IP**, **TP**, and **HP**, and the inset amount used by **RS**. The default is 7n on terminals and 7.2n on typesetters. Use only integer multiples of unit "n" on terminals for consistent indentation.

–rLL=*line-length*
: Set line length; the default is 80n on terminals and 6.5i on typesetters.

–rLT=*title-length*
: Set the line length for titles. By default, it is set to the line length (see **–rLL** above).

–dMF=*man-page-topic-font*
: Select the font used for man page identifiers in **TH** calls and topics named in **MR** calls; the default is "**I**" (italic style of the default family). Any valid argument to *groff*'s "ft" request may be used. If the **MF** string ends in "I", the package assumes it to be an oblique typeface, and applies italic corrections before and after man page topics and identifiers.

–rP*n*
: Start enumeration of pages at *n*. The default is 1.

–rPO=*page-offset*
: Set page offset; the default is 0 on terminals and 1i on typesetters.

–rS*type-size*
: Use *type-size* for the document's body text; acceptable values are 10, 11, or 12 points. See subsection "Font style macros" above for the default.

–rSN=*subsection-indentation*
: Set indentation of subsection headings to *subsection-indentation.* The default is 3n.

–rTS=*separation*
: Require the given *separation* between a **TP** paragraph's tag and its body. The default is 2n.

–rU0
: Disable generation of URI hyperlinks in output drivers capable of them, making the arguments to **MT** and **UR** calls visible as formatted text. *grohtml*(1), *gropdf*(1), and *grotty*(1) enable hyperlinks by default (the last only if not in its legacy output mode).

−rX*p* Number successors of page *p* as *p*a, *p*b, *p*c, and so forth. The register tracking the suffixed page letter uses format "a" (see the "af" request in *groff* (7)).

Files

/usr/share/groff/1.24.2/tmac/an.tmac
Most *man* macros are defined in this file. It also loads extensions from *an−ext.tmac* (see below).

/usr/share/groff/1.24.2/tmac/andoc.tmac
This brief *groff* program detects whether the *man* or *mdoc* macro package is used by a document and loads the correct macro definitions, taking advantage of the fact that pages using them must call **TH** or **Dd**, respectively, before any other macros. A *man* program or a user typing, for example, "**groff −mandoc page.1**", need not know which package the file *page.1* uses. Multiple man pages, in either format, can be handled; *andoc* reloads each macro package as necessary. Page-local redefinitions of names used by the *man* or *mdoc* packages prior to **TH** or **Dd** calls are "clobbered" by the reloading process. If you want to provide your own definition of an extension macro to ensure its availability, the *an−ext.tmac* entry below offers advice.

/usr/share/groff/1.24.2/tmac/an−ext.tmac
Definitions of macros described above as extensions (and not deprecated) are contained in this file; in some cases, they are simpler versions of definitions appearing in *an.tmac*, and are ignored if the formatter is GNU *troff*. They are written to be compatible with AT&T *troff* and permissively licensed—not copylefted. To reduce the risk of name space collisions, string and register names begin only with "**m**". We encourage man page authors who are concerned about portability to legacy Unix systems to copy these definitions into their pages, and maintainers of *troff* implementations or workalike systems that format man pages to re-use them. To ensure reliable rendering, define them after your page calls **TH**; see the discussion of *andoc.tmac* above. Further, it is wise to define such page-local macros (if at all) after the "Name" section to accommodate timid *makewhatis*(8) or *mandb*(8) implementations that easily give up scanning for indexing material.

/usr/share/groff/1.24.2/tmac/man.tmac
is a wrapper enabling the package to be loaded with the option "**−m man**".

/usr/share/groff/1.24.2/tmac/mandoc.tmac
is a wrapper enabling *andoc.tmac* to be loaded with the option "**−m mandoc**".

/etc/groff/site−tmac/man.local
Put site-local changes and customizations into this file.

Authors

James Clark wrote the initial GNU implementation of the *man* macro package. Later, Werner Lemberg supplied the **S**, **LT**, and **cR** registers, the last a 4.3BSD-Reno *mdoc*(7) feature. Larry Kollar added the **FT**, **HY**, and **SN** registers; the **HF** string; and the **PT** and **BT** macros in *groff* 1.19 (2003). Lemberg and Eric S. Raymond contributed **EX/EE**, **MT/ME**, **UR/UE**, **TQ**, and an early version of the **SY/YS** macros to *groff* 1.20 (2009). G. Branden Robinson implemented the **AD** and **MF** strings; **CS**, **CT**, and **U** registers; and the **MR** macro for *groff* 1.23 (2023), and the **BP**, **PO**, and **TS** registers and a revised implementation of the **SY/YS** macros for *groff* 1.24 (2026).

Susan G. Kleinmann wrote the initial version of this document for the Debian GNU/Linux system. Lemberg imported it to *groff*. He and Robinson revised and updated it. Raymond and Robinson documented the extension macros.

See also

tbl(1), *eqn*(1), and *refer*(1) are preprocessors used with man pages. *man*(1) describes the man page librarian on your system. *groff_mdoc*(7) details the *groff* version of BSD's alternative macro package for man pages.

groff_man_style(7), *groff* (7), *groff_char*(7)

Name

groff_man_style – GNU *roff* man page tutorial and style guide

Synopsis

groff –man [*option* ...] [*file* ...]
groff –m man [*option* ...] [*file* ...]

Description

The GNU implementation of the *man* macro package is part of the *groff* document formatting system. It is used to compose manual pages ("man pages") like the one you are reading. This document presents the macros thematically; for those needing only a quick reference, the following table lists them alphabetically, with references to appropriate subsections below. Experienced *man* authors may prefer *groff_man*(7).

Macro	Meaning	Subsection
.B	Bold	Font style macros
.BI	Bold, italic alternating	Font style macros
.BR	Bold, roman alternating	Font style macros
.EE	Example end	Document structure macros
.EX	Example begin	Document structure macros
.HP	Begin hanging paragraph	Paragraphing macros
.I	Italic	Font style macros
.IB	Italic, bold alternating	Font style macros
.IP	Indented paragraph	Paragraphing macros
.IR	Italic, roman alternating	Font style macros
.LP	Begin paragraph	Paragraphing macros
.ME	Mail-to end	Hyperlink macros
.MR	Man page cross reference	Hyperlink macros
.MT	Mail-to start	Hyperlink macros
.P	Begin paragraph	Paragraphing macros
.PP	Begin paragraph	Paragraphing macros
.RB	Roman, bold alternating	Font style macros
.RE	Relative inset end	Document structure macros
.RI	Roman, italic alternating	Font style macros
.RS	Relative inset start	Document structure macros
.SH	Section heading	Document structure macros
.SM	Small	Font style macros
.SS	Subsection heading	Document structure macros
.SY	Synopsis start	Synopsis macros
.TH	Title heading	Document structure macros
.TP	Tagged paragraph	Paragraphing macros
.TQ	Supplemental paragraph tag	Paragraphing macros
.UE	URI end	Hyperlink macros
.UR	URI start	Hyperlink macros
.YS	Synopsis end	Synopsis macros

We discuss other macros (**AT**, **DT**, **OP**, **PD**, **SB**, and **UC**) in subsection "Deprecated features" below.

Throughout Unix documentation, a manual entry is referred to simply as a "man page", regardless of its length, without gendered implication, and irrespective of the macro package selected for its composition.

A man page employs the Unix line-ending convention (U+000A only). Some basic Latin characters have special meaning to *roff;* see subsection "Portability" below.

Fundamental concepts

groff is a programming system for typesetting: we thus often use the verb "to set" in the sense "to typeset". The formatter *troff*(1) collects words from the input and *fills* output lines with as many as can fit. *Words* are separated by spaces and newlines. A transition to a new output line is called a *break.* Breaks can occur at explicit hyphens, at **\%** or **\:** escape sequences (see subsection "Portability" below), or at predetermined

locations in a word if automatic hyphenation is enabled (see the **−rHY** option in section "Options" below). An output line may be supplemented with *inter-sentence space,* then potentially *adjusted* with more space to a consistent length (see the **−dAD** option). *roff*(7) details these processes. The formatter prepares output for *terminals* or for more capable *typesetters* that can change the type size and font family.

A *roff* document can contain *control lines,* which start with a dot (**.**) or neutral apostrophe (**'**). All other input lines are *text lines* to be formatted. A *macro* collects control and/or text lines to ease document composition. *man* is a *macro package.* To call a macro, put its name after a dot on a control line. Some macros interpret *arguments,* words that follow its name. A newline, unless escaped (see subsection "Portability" below), marks the end of the macro call. A control line with no macro name on it is called an *empty request;* it does nothing.

We describe below several *man* macros that plant one-line *input traps:* the next input line that directly produces formatted output is treated specially. For *man* documents that follow the advice in section "Portability" below, this means that control lines using the empty request and uncommented input lines ending with an escaped newline do not spring the trap; anything else does (but see the **TP** macro description).

Macro reference preliminaries

A tagged paragraph describes each macro. We present coupled pairs together, as with **EX** and **EE**. Square brackets surround optional macro arguments. If a macro accepts multiple arguments, those containing space characters must be double-quoted to be interpreted correctly. If you require an empty macro argument, specify it as a pair of neutral double quotes (**""**). See section "Notes" below for examples of cases where better alternatives to empty arguments in macro calls are available. Most macro arguments are formatted as text in the output; exceptions are noted. We identify some macros as *extensions* to the set originally implemented. They are not supported everywhere; see subsection "Use of extensions" below.

Document structure macros

Document structure macros organize a man page's content. All of them break the output line. **TH** (title heading) identifies the document as a man page and configures the page headers and footers. Section headings (**SH**), one of which is mandatory and many of which are conventionally expected, facilitate location of material by the reader and aid the man page writer to discuss all essential aspects of the topics presented. Subsection headings (**SS**) are optional and permit sections that grow long to develop in a controlled way. Many technical discussions benefit from examples; lengthy ones, especially those reflecting multiple lines of input to or output from the system, are usefully bracketed by **EX** and **EE**. When none of the foregoing meets a structural demand, use **RS**/**RE** to inset a region within a (sub)section.

.TH *identifier section* [*footer-middle* [*footer-inside* [*header-middle*]]]

Break the page, reset the page number to 1 (unless the **−rC1** option is given), and use the arguments to populate the page header and footer. *roff* systems refer to these collectively as "titles". Together, *identifier* and the *section* of the manual to which it belongs can uniquely identify a *man* document on the system. This use of "section" has nothing to do with the section headings otherwise discussed in this page; it arises from the organizational scheme of printed and bound Unix manuals. See *man*(1) or *intro*(1) for the manual sectioning applicable to your system. *identifier* and *section* are positioned at the left and right in the header; the latter is set after the former, in parentheses and without space. *footer-middle* is centered in the footer. By default, *footer-inside* is positioned at the bottom left. Use of the double-sided layout option **−rD1** places *footer-inside* at the bottom left on recto (odd-numbered) pages, and the bottom right on verso (even-numbered) pages. By default, the outside footer is the page number. Use of the continuous-rendering option **−rcR=1** replaces it with *identifier* and *section,* as in the header. *header-middle* is centered in the header. If *section* is an integer between 1 and 9 (inclusive), there is no need to specify *header-middle; an.tmac* supplies text for it. If *identifier* or *footer-inside* would overrun the space available in the header and/or footer, this package may abbreviate them with ellipses (**...**). *groff man* suppresses headers and footers in HTML output.

A valid *man* document calls **TH** only once, early in the file, prior to any other macro calls. By convention, *footer-middle* is the date of the most recent modification to the source document, in ISO 8601 format (*YYYY-MM-DD*), and *footer-inside* is the name and version or release of the project providing it.

.SH [*heading-text*]

Set *heading-text* as a section heading. Given no argument, **SH** plants a one-line input trap; text on the next line becomes *heading-text*. The heading text is set in bold (or the font specified by the string **HF**), and, on typesetters, slightly larger than the base type size. If the heading font ***[HF]** is bold, use of an italic style in *heading-text* is mapped to the bold-italic style if available in the font family. The inset level is reset to 1; see subsection "Horizontal and vertical spacing" below. Text lines after the call are set as an ordinary paragraph (**P**).

The content of *heading-text* and ordering of sections follows a set of common practices, as does much of the layout of material within sections. For example, a section called "Name" or "NAME" must exist, must be the first section after the **TH** call, and must contain only text of the form

topic[**,** *another-topic*]... \− *summary-description*

for tools like *makewhatis*(8) or *mandb*(8) to index them.

.SS [*subheading-text*]

Set *subheading-text* as a subsection heading indented between a section heading and an ordinary paragraph (**P**). Given no argument, **SS** plants a one-line input trap; text on the next line becomes *subheading-text*. The subheading text is set in bold (or the font specified by the string **HF**). If the heading font ***[HF]** is bold, use of an italic style in *subheading-text* is mapped to the bold-italic style if available in the font family. The inset level is reset to 1; see subsection "Horizontal and vertical spacing" below. Text lines after the call are set as an ordinary paragraph (**P**).

.EX

.EE Begin and end example. After **EX**, filling is disabled (and, on typesetters, a monospaced font family is selected). Calling **EE** enables filling (and restores the previous family).

Example regions are useful for formatting code, shell sessions, and text file contents. An example region is not a "literal mode" of any sort: special character escape sequences must still be used to produce correct glyphs for **'**, **−**, ****, **^**, **`**, and **~** (see subsection "Portability" below). Sentence endings are still detected and supplemental inter-sentence space applied. If the amount of supplemental inter-sentence spacing is altered, the rendering of, for instance, regular expressions using **.** or **?** followed by multiple spaces can change. Use the dummy character escape sequence **\&** before the spaces.

Ninth Edition Unix introduced the **EX** and **EE** extensions. Documenter's Workbench (DWB), Heirloom Doctools, and Plan 9 *troff*s, and *mandoc* (since 1.12.2) support them. Solaris *troff* does not.

.RS [*inset-amount*]

Start new relative inset. *man* saves any current inset amount and moves right by: *inset-amount,* if specified; the *indentation* amount of the preceding **IP**, **TP**, or **HP** macro call if no (sub-)sectioning or ordinary paragraphing macro has intervened; or the amount of the **IN** register. **RS** calls can nest; each increments by 1 the *level* used by **RE**. The level prior to any **RS** call is 1.

.RE [*inset-level*]

End a relative inset, reducing it to that of *inset-level* (or by 1 if not specified) and restoring the corresponding inset amount.

Paragraphing macros

These macros break the output line. An *ordinary* paragraph (**P**) like this one indents all output lines by the same amount. A hanging paragraph (**HP**) is a cosmetic variant of **P** with a hanging indent. Definition lists frequently occur in man pages; these can be set as *tagged paragraphs,* which have one (**TP**) or more (**TQ**) leading tags followed by a paragraph that has an additional indentation. The indented paragraph (**IP**) macro can continue the indented content of a narrative started with **TP**, or present an itemized or ordered list. If a paragraphing macro has been called since **SH** or **SS**, all except **TQ** follow the break with vertical space (in an amount configured by the deprecated **PD** macro); see subsection "Horizontal and vertical spacing" below. Except for **TQ**, these macros reset the type size, hyphenation, and adjustment to (configured) defaults, and the font style to roman.

.P
.LP
.PP Begin a new paragraph; these macros are synonymous. Any indentation from use of **IP**, **TP**, or **HP** is cleared. The inset amount, as affected by **RS** and **RE**, is not.

.HP [*indentation*]
Set a paragraph with a hanging indentation. Text on output lines after the first is indented by *indentation,* if specified, and by the amount of the **IN** register otherwise.

Caution: A hanging indentation cannot be expressed naturally in (pure) HTML, a hanging paragraph is not distinguishable from an ordinary one if it formats on only one output line, and non-*roff* -based man page interpreters may treat **HP** as an ordinary paragraph anyway. Thus, information or distinctions you mean to express with indentation may be lost.

.TP [*indentation*]
Set an indented paragraph with a leading unindented tag. The macro plants a one-line input trap that honors the **\c** escape sequence; text on the next line becomes the tag, set without indentation. Text on subsequent lines is indented by *indentation,* if specified, and by the amount of the **IN** register otherwise. If the tag, plus the "tag spacing" stored in the **TS** register (see section "Options" below) is wider than the indentation, the package breaks the line after the tag.

The line containing the tag can include a macro call, for instance to set the tag in bold with **B**. **TP** was used to write the first paragraph of this description of **TP**, and **IP** the subsequent one.

.TQ Set an additional tag for a paragraph tagged with **TP**, planting a one-line input trap as with **TP**.

TQ is a GNU extension supported by Heirloom Doctools *troff* (since Git snapshot 151218) and *mandoc* (since 1.14.5) but not by DWB, Plan 9, or Solaris *troff* s.

The description of **P**, **LP**, and **PP** above was written using **TP** and **TQ**.

.IP [*mark* [*indentation*]]
Set an indented paragraph with an optional mark. Arguments, if present, are handled as with **TP**, except that the *mark* argument to **IP** cannot include a macro call, and the tag separation amount stored in the **TS** register is not enforced.

Two convenient uses for **IP** are

(1) to start a new paragraph with the same indentation as an immediately preceding **IP** or **TP** paragraph, if no *indentation* argument is given; and

(2) to set a paragraph with a short *mark* that is not semantically important, such as a bullet (•)—obtained with the **\[bu]** special character escape sequence—or list enumerator, as seen in this very paragraph.

Synopsis macros

Use **SY** and **YS** to summarize syntax using familiar Unix conventions. Heirloom Doctools *troff* (since Git snapshot 151218) and *mandoc* (since 1.14.5) support these GNU extensions; DWB, Plan 9, and Solaris *troff* s do not.

.SY *keyword* [*suffix*]
Begin synopsis. Adjustment and automatic hyphenation are disabled. If **SY** has already been called without a corresponding **YS**, a break is performed. *keyword* and any *suffix* are set in bold. When *suffix* is present, the package sets the next word after it without intervening space. If a break is required in subsequent text (up to a paragraphing, sectioning, or **YS** macro call), lines after the first are indented. Unless the previous synopsis's indentation is reused (see **YS** below), output lines after the first indent by the width of the pending output line up to the end of *keyword* plus a space, if *keyword* is the only argument, and up to the end of *suffix* otherwise.

.YS [*reuse-indentation*]
: End synopsis, breaking the line and restoring indentation, adjustment, and hyphenation to their previous states. If an argument is given, the indentation corresponding to the previous **SY** call is reused by the next **SY** call instead of being computed.

Interleave multiple **SY/YS** blocks with paragraphing macros to distinguish differing modes of operation of a complex command like *tar*(1). Omit the paragraphing macro to indicate synonymous ways of invoking a particular mode of operation. Paragraphing macros can similarly manage grouping of function synopses.

groff's own command-line interface illustrates most specimens of synopsis syntax one may encounter.

```
.SY groff
.RB [ \-abcCeEgGijklNpRsStUVXzZ ]
.RB [ \-d\~\c
.IR ctext ]
.RB [ \-d\~\c
.IB string =\c
.IR text ]
.RB [ \-D\~\c
.IR fallback-encoding ]
(and so on similarly)
.RI [ file\~ .\|.\|.]
.YS
.
.
.P
.SY groff
.B \-h
.YS
.
.SY groff
.B \-\-help
.YS
.
.
.P
.SY groff
.B \-v
.RI [ option\~ .\|.\|.\&]
.RI [ file\~ .\|.\|.]
.YS
.
.SY groff
.B \-\-version
.RI [ option\~ .\|.\|.\&]
.RI [ file\~ .\|.\|.]
.YS
```

Given the foregoing input, *groff man* produces the following output.

groff [**–abcCeEgGijklNpRsStUVXzZ**] [**–d** *ctext*] [**–d** *string*=*text*] [**–D** *fallback-encoding*] [**–f** *font-family*] [**–F** *font-directory*] [**–I** *inclusion-directory*] [**–K** *input-encoding*] [**–L** *spooler-argument*] [**–m** *macro-package*] [**–M** *macro-directory*] [**–n** *page-number*] [**–o** *page-list*] [**–P** *postprocessor-argument*] [**–r** *cnumeric-expression*] [**–r** *register*=*numeric-expression*] [**–T** *output-device*] [**–w** *warning-category*] [**–W** *warning-category*] [*file* . . .]

groff –h
groff ––help

groff –v [*option* . . .] [*file* . . .]
groff ––version [*option* . . .] [*file* . . .]

Several features of the above example are of note.

- The empty request (.), which does nothing, vertically spaces the input file for readability by the document maintainer; see subsection "Portability" below regarding blank lines.
- Command and option names are presented in **bold** to cue the user that they should be input literally.
- Option dashes are specified with the **\–** escape sequence; this is an important practice to make them clearly visible and to facilitate copy-and-paste from the rendered man page to a shell prompt or text file.
- Option arguments and command operands are presented in *italics* (but see subsection "Font style macros" below regarding terminals) to cue the user that they must be replaced with appropriate input.
- Symbols that are neither to be typed literally nor replaced at the user's discretion appear in the roman style; brackets surround optional arguments, and an ellipsis indicates that the previous syntactic element may be repeated arbitrarily. Where whitespace separates optional arguments, a space precedes the ellipsis.
- The non-breaking adjustable space escape sequence **\~** prevents the output line from breaking within the option brackets; see subsection "Portability" below.
- The output line continuation escape sequence **\c** is used with font style alternation macros to allow all three font styles to be set without (breakable) space among them; see subsection "Portability" below.
- The dummy character escape sequence **\&** follows the ellipsis when further text is to follow after space on the output line, keeping its last period from being interpreted as the end of a sentence because it is followed by characters that are transparent to end-of-sentence detection, and/or a newline, which would in turn normally cause the formatter to place supplemental inter-sentence space after it. See subsection "Portability" below.

We might synopsize the standard C library function *bsearch*(3) as follows.

```
.P
.B void *\c
.SY bsearch (
.BI const\~void\~* key ,
.BI const\~void\~* base ,
.BI size_t\~ nmemb ,
.BI int\~(* compar )\c
.B (const\~void\~*, const\~void\~*));
.YS
```

groff man produces the following result.

void *bsearch(const void ***key*, const void ***base***, size_t** *nmemb***, int (***compar***)(const void *, const void *));**

Hyperlink macros

Man page cross references like *ls*(1) are best presented with **MR**. Mark email addresses with **MT/ME** and other sorts of URI with **UR/UE**. To hyperlink text, terminals and pager programs must support ECMA-48 OSC 8 escape sequences (see *grotty*(1)). When device support is unavailable or disabled with the **U** register (see section "Options" below), *groff man* renders these URIs between angle brackets (⟨ ⟩) after the linked text.

MT, **ME**, **UR**, and **UE** are GNU extensions supported by Heirloom Doctools *troff* (since Git snapshot 151218) and *mandoc* (**UR/UE** since 1.12.3; **MT/ME** since 1.14.2) but not by DWB, Plan 9 (original), or Solaris *troff*s. Plan 9 from User Space's *troff* implements **MR**.

Prepare arguments to **MR**, **MT**, and **UR** for typesetting; they can appear in the output. Use special character escape sequences to encode Unicode basic Latin characters where necessary, particularly the hyphen-minus. (See subsection "Portability" below.) URIs can be lengthy; rendering them can result in jarring adjustment or variations in line length, or *troff* warnings when one is longer than an output line. The application of non-printing break point escape sequences **\:** after each slash (or series thereof), and before each dot (or series thereof) is recommended as a rule of thumb. The former practice avoids forcing a trailing slash in a URI onto a separate output line, and the latter helps the reader to avoid mistakenly interpreting a dot at the end of a line as a period (or multiple dots as an ellipsis). Thus,

```
.UR http://\:example\:.com/\:fb8afcfbaebc74e\:.cc
```

has several potential break points in the URI shown. Consider adding break points before or after at signs in email addresses, and question marks, ampersands, and number signs in HTTP(S) URIs.

.MR *topic* [*manual-section* [*trailing-text*]]

(since groff *1.23)* Set a man page cross reference as "*topic*(*manual-section*)". If *manual-section* is absent, the package omits the surrounding parentheses. If *trailing-text* (typically punctuation) is specified, it follows the closing parenthesis without intervening space. Hyphenation is disabled while the cross reference is set. *topic* is set in the font specified by the **MF** string. If *manual-section* is present, the cross reference hyperlinks to a URI of the form "**man:***topic*(*manual-section*)".

```
The output driver
.MR grops 1
produces PostScript from
.I troff
output.
.
The Ghostscript program (\c
.MR gs 1 )
interprets PostScript and PDF.
```

.MT *address*

.ME [*trailing-text*]

Identify *address* as an RFC 6068 *addr-spec* for a "mailto:" URI with the text between the two macro calls as the link text. An argument to **ME** is placed after the link text without intervening space. *address* may not be visible in the rendered document if hyperlinks are enabled and supported by the output driver. If they are not, *address* is set in angle brackets after the link text and before *trailing-text.* If hyperlinking is enabled but there is no link text, *address* is formatted and hyperlinked *without* angle brackets, except when *address* appears as a **TP** paragraph tag.

When rendered by *groff* to a PostScript device,

```
Contact
.MT fred\:.foonly@\:fubar\:.net
Fred Foonly
.ME
for more information.
```

formats as "Contact Fred Foonly ⟨fred.foonly@fubar.net⟩ for more information.".

.UR *uri*

.UE [*trailing-text*]

Identify *uri* as an RFC 3986 URI hyperlink with the text between the two macro calls as the link text. An argument to **UE** is placed after the link text without intervening space. *uri* may not be visible in the rendered document if hyperlinks are enabled and supported by the output driver. If they are not, *uri* is set in angle brackets after the link text and before *trailing-text.* If hyperlinking is enabled but there is no link text, *uri* is formatted and hyperlinked *without* angle brackets, except when *uri* appears as a **TP** paragraph tag.

When rendered by *groff* to a PostScript device,

```
The GNU Project of the Free Software Foundation
hosts the
.UR https://\:www\:.gnu\:.org/\:software/\:groff/
.I groff
home page
.UE .
```

formats as "The GNU Project of the Free Software Foundation hosts the *groff* home page ⟨https://www.gnu.org/software/groff/⟩.".

If a **UR/UE** or **MT/ME** pair occurs in a **TP** tag and hyperlinking is unavailable, *groff man* sets the link target at the beginning of the indented paragraph, not as part of the tag, unless there is no link text.

Font style macros

The *man* macro package is limited in its font styling options, offering only **bold** (**B**), *italic* (**I**), and roman. Italic text may instead render underscored on terminals. **SM** sets text at a smaller type size, which differs visually from regular-sized text only on typesetters. The macros **BI**, **BR**, **IB**, **IR**, **RB**, and **RI** set their odd- and even-numbered arguments as text in the alternating styles their names indicate, with no space separating them.

Because font styles are presentational rather than semantic, conflicting traditions have arisen regarding which font styles should be used to mark file or path names, environment variables, and inlined literals.

The default type size and family for typesetters is 10-point Times, except on the **X75–12** and **X100–12** devices where the type size is 12 points. The default style is roman.

.B [*text*]

Set *text* in bold. Given no argument, **B** plants a one-line input trap; text on the next line, which can be further formatted with a macro, is set in bold.

Use bold for literal portions of syntax synopses, for command-line options in running text, and for literals that are major topics of the subject under discussion; for example, this page uses bold for macro, string, and register names. In an **EX/EE** example of interactive I/O (such as a shell session), set only user input in bold.

.I [*text*]

Set *text* in an italic or oblique face. Given no argument, **I** plants a one-line input trap; text on the next line, which can be further formatted with a macro, is set in an italic or oblique face.

Use italics for file and path names, for environment variables, for C data types, for enumeration or preprocessor constants in C, for variant (user-replaceable) portions of syntax synopses, for the first occurrence (only) of a technical concept being introduced, for names of journals and of literary works longer than an article, and anywhere a parameter requiring replacement by the user is encountered. An exception involves variant text in a context already typeset in italics, such as file or path names with replaceable components; in such cases, follow the convention of mathematical typography: set the file or path name in italics as usual but use roman for the variant part (see **IR** and **RI** below), and italics again in running roman text when referring to the variant material.

.SM [*text*]

Set *text* one point smaller than the default type size on typesetters. Given no argument, **SM** plants a one-line input trap; text on the next line, which can be further formatted with a macro, is set smaller.

Note: terminals render *text* at normal size instead. Do not rely upon **SM** to communicate semantic information distinct from using roman style at normal size; it is hidden from readers using such devices.

Observe what is *not* prescribed for setting in bold or italics above: elements of "synopsis language" such as ellipses and brackets around options; proper names and adjectives; titles of anything other than major works of literature; identifiers for standards documents or technical reports such as CSTR #54, RFC 1918, Unicode 16, or POSIX.1-2024; acronyms; and occurrences after the first of a technical term.

Use italics for emphasis rarely, and bold almost never. Brief specimens of literal text, such as article titles, inline examples, mentions of individual characters or short strings, and (sub)section headings in man pages, are suitable for quotation; see the **\[lq]**, **\[rq]**, **\[oq]**, and **\[cq]** escape sequences in subsection "Portability" below.

Unlike the above font style macros, the font style alternation macros below set no input traps; they must be given arguments to have effect. They apply italic corrections as appropriate. If a space is required within an argument, first consider whether the same result could be achieved with as much clarity by using single-style macros on separate input lines. When it cannot, double-quote an argument containing embedded space characters. Setting all three different styles within a word presents challenges; it is possible with the **\c** and/or **\f** escape sequences. See subsection "Portability" below for approaches.

.BI *bold-text italic-text* ...
Set each argument in bold and italics, alternately.

```
.BI -r\~ register = numeric-expression
```

.BR *bold-text roman-text* ...
Set each argument in bold and roman, alternately.

```
Set an ellipsis on the math axis with the GNU extension macro
.BR cdots .
```

.IB *italic-text bold-text* ...
Set each argument in italics and bold, alternately.

```
In places where
.IB n th
is allowed,
```

.IR *italic-text roman-text* ...
Set each argument in italics and roman, alternately.

```
Use GNU
.IR pic 's
.B figname
command to change the name of the vbox.
```

.RB *roman-text bold-text* ...
Set each argument in roman and bold, alternately.

```
.I file
is
.RB \[lq] \- \[rq],
.I groff
reads the standard input stream.
```

.RI *roman-text italic-text* ...
Set each argument in roman and italics, alternately.

```
.RI ( tpic
was a fork of AT&T
.I pic
by Tim Morgan of the University of California at Irvine
```

Horizontal and vertical spacing

The package sets all text inboard of the left edge of the output medium by the amount of the *page offset;* see register **PO** in section "Options" below. Headers, footers (both set with **TH**), and section headings (**SH**) lie at the page offset. *groff man* indents subsection headings (**SS**) by the amount in the **SN** register.

Ordinary paragraphs not within an **RS**/**RE** inset region are inset by the amount stored in the **BP** register; see section "Options" below. The **IN** register configures the default indentation amount used by **RS** (as the *inset-amount*), **IP**, **TP**, and **HP**; an overriding argument is a number plus an optional scaling unit. If no scaling unit

is given, the *man* package assumes "n"; that is, roughly the width of a letter "n" in the font current when the macro is called—see section "Measurements" in *groff*(7). An indentation specified in a call to **IP**, **TP**, or **HP** persists until (1) another of these macros is called with an *indentation* argument, or (2) **SH**, **SS**, or **P** or its synonyms is called; these clear the indentation entirely.

The inset amount and indentation are related but distinct parameters with the same defaults. The former is manipulated by **RS** and **RE** (and by **SH** and **SS**, which reset it to the default). Indentation is controlled by the paragraphing macros (though, again, **SH** and **SS** reset it); it is imposed by the **TP**, **IP**, and **HP** macros, and cancelled by **P** and its synonyms. An extensive example follows.

This ordinary (**P**) paragraph is not in a relative inset nor does it possess an indentation.

Now we have created a relative inset with **RS** and started another ordinary paragraph with **P**. We observe that all of its lines are inset and indented the same; contrast with a first-line indentation.

tag This tagged paragraph, set with **TP**, is still within the **RS** region, but lines after the first have a supplementary indentation that the tag lacks.

A paragraph like this one, set with **IP**, appears to the reader as also associated with the tag above, because **IP** re-uses the previous paragraph's indentation unless given an argument to change it. Both the inset amount (**RS**) and indentation (**IP**) affect this paragraph.

This table is affected both by the inset amount and indentation.

- This indented paragraph is marked with a bullet, contrasting the inset amount and the indentation; only the former affects the mark, but both affect the text of the paragraph.

This ordinary (**P**) paragraph resets the indentation, but the inset amount is unchanged.

This table is affected only by the inset amount.

Finally, we have ended the relative inset by using **RE**, which (because we used only one **RS**/**RE** pair) has restored the inset amount to its initial value. This is an ordinary **P** paragraph.

Resist the temptation to mock up tabular or multi-column output with tab characters or the indentation arguments to **IP**, **TP**, **RS**, or **HP**; the result may not be comprehensible on an output device you fail to check, or which is developed in the future. Consider the table preprocessor *tbl*(1) instead.

Several macros insert vertical space: **SH**, **SS**, **TP**, **P** (and its synonyms), **IP**, and **HP**. They then enable *no-space mode;* see *groff*(7). The default inter-section and inter-paragraph spacing is 1v for terminals and 0.4v for typesetters. "v" is a unit of vertical distance, where 1v is the distance between adjacent text baselines. (The deprecated macro **PD** can change this vertical spacing, but we discourage its use.) Between **EX** and **EE** calls, the inter-paragraph spacing is 1v regardless of output device.

Registers

Registers are described in section "Options" below. They can be set not only on the command line but in the site *man.local* file as well; see section "Files" below.

Strings

The following strings are defined for use in man pages. *groff man* supports others for configuration of rendering parameters; see section "Options" below.

***R** interpolates a special character escape sequence for the "registered sign" glyph, **\(rg**, if available, and "(Reg.)" otherwise.

***S** interpolates an escape sequence setting the type size to the document default.

***(lq**
***(rq** interpolate special character escape sequences for left and right double-quotation marks, **\(lq** and **\(rq**, respectively.

***(Tm** interpolates a special character escape sequence for the "trade mark sign" glyph, **\(tm**, if available, and "(TM)" otherwise.

A contemporary man page needs none of the above. ***S** is superfluous; type size changes are invisible on terminals, and macros that change it restore its original value afterward. Better alternatives exist for the rest; simply use the **\[rg]**, **\[lq]**, **\[rq]**, and **\[tm]** special character escape sequences directly. Unless you are aiming for a pathological level of portability—perhaps composing a man page for consumption on simulators of 1980s Unix systems (or Solaris *troff*, though even it supports " **\(rg**")—avoid using the above strings.

Use of extensions

To ensure that your man page formats reliably on a wide variety of viewers, write it solely with the macros described in this page (except for the ones identified as deprecated, which you should avoid). Macros described as extensions might be unsupported by a formatter that is important to your audience. Nevertheless, *groff*'s extensions are present because they perform tasks that are otherwise difficult or tedious to achieve portably. If you require an extension but expect your man page to be rendered on a system that doesn't support it, write a configuration test to measure a property of the system, and use *m4*(1), *sed*(1), or a similar tool to generate a *.man* file from a *.man.in* file, defining page-local versions of extension macros only where necessary. You can copy extension macro definitions from *groff*; see *an-ext.tmac* in section "Files" below.

For example, we might put a line

```
@DEFINE_MR@
```

in our *man* document at the end of the "Name" section, test a system for the availability of the *groff man* **MR** macro, remove the line if the macro is present, and "inline" a definition otherwise, as follows. (This version is slightly simplified and does not attempt to disable hyphenation when setting arguments to **MR**.)

```
have_MR=$(echo .pm | troff -man 2>&1 | grep 'MR[[:space:]]')
if [ -n "$have_MR" ]
then
  sed '/@DEFINE_MR@/d' myprog.man.in > myprog.man
else
  sed 's/@DEFINE_MR@/.de MR\
.  ie \\\\n(.$=1 .I \\\\$1\
.  el .IR \\\\$1 (\\\\$2)\\\\$3\
../' myprog.man.in > myprog.man
fi
```

(The profusion of backslashes is due to its status as an escape character in both *roff* and *sed*.)

If the foregoing method is too much trouble, you could apply the radical technique of reading your man page using every formatter of interest, confirming satisfactory output from each. Test documentation for syntactic validity and semantic correctness, just as you would test code.

Portability

GNU *troff* expects its input to contain Unicode basic Latin code points exclusively. One can *maintain* it in a more convenient encoding, using *preconv*(1), as with *groff*(1)'s **-k** option, to generate a basic Latin version that employs special character escape sequences to access other glyphs.

AT&T *troff*'s *man* package and those of many of its descendants format man pages using a line length of 65 ens (character cells) on terminals, whereas *groff man* uses 80 ens (and *mandoc*(1) 78). Readers with low vision may also benefit from the narrower line length. Inspect your documents with "**nroff -t -r LL=65n -man**" to ensure that their output lines don't overrun.

In *roff* systems, elemental functions called *requests* and *escape sequences* control formatting operations. A request appears on a control line. An escape sequence starts with a backslash (\) and can appear almost anywhere. However, use of *roff* requests (apart from the empty request "**.**") risks poor rendering when a page is processed by non-*roff* formatters that attempt to interpret page sources. (Historically, this was commonly attempted for HTML conversion.) Many of these programs don't interpret the full *roff* language (let alone extensions): they may be incapable of handling numeric expressions, control structures, or register, string, and macro definitions, causing a document's contents to be presented incomprehensibly or omitted entirely. If

your document uses formatter requests, or escape sequences not shown below, it accepts responsibility for restoring formatter state to what the *man* macro package expects.

Do not put blank (empty) lines in a man page source document. They can produce excessive space in the output, or less than is attempted; some *man*(1) programs "squeeze" multiple blank output lines into one.

The wise *man* author quotes multi-word section and subsection headings; the **SH** and **SS** macros of *man*(7) implementations descended from Seventh Edition Unix supported six arguments at most. This restriction also applied to the **B**, **I**, **SM**, and font style alternation macros.

Exercise restraint with escape sequences as with requests. Some escape sequences are however required for correct typesetting even in man pages and usually do not cause portability problems. Several of these render glyphs corresponding to punctuation code points in the Unicode basic Latin range (U+0020–U+007F) that are handled specially in *roff* input; the escape sequences below must be used to render them correctly and portably when documenting material that uses them as literals—namely, any of the set **' – \ ^ ` ~** (apostrophe, dash or hyphen-minus, backslash, caret, grave accent, tilde).

\" Comment. The formatter ignores everything after the double quote to the end of the input line. Place whole-line comments on a control line immediately after the empty request ("**.**").

*****_newline_ Join the next input line to the current one. Except for the update of the input line counter (used for diagnostic messages and related purposes), a series of lines ending in backslash-newline appears to *groff* as a single input line. Splitting excessively long input lines can ease document maintenance.

\% Control hyphenation. The location of this escape sequence within a word marks a hyphenation point, supplementing *groff*'s automatic hyphenation patterns. At the beginning of a word, it suppresses any hyphenation breaks within *except* those specified with **\%**.

\: Insert a non-printing break point. A word can break at such a point, but a hyphen glyph is not written to the output if it does. The remainder of the word is subject to hyphenation as normal. You can use **\:** and **\%** in combination to control breaking of a file name or URI or to permit hyphenation only after certain explicit hyphens within a word. See subsection "Hyperlink macros" above for an example.

\: is a GNU extension also supported by Heirloom Doctools *troff* 050915 (September 2005), *mandoc* 1.13.1 (2014-08-10), and *neatroff* (commit 399a4936, 2014-02-17), but not by DWB, Plan 9, or Solaris *troff*s.

\~ Adjustable non-breaking space. Use this escape sequence to prevent a break inside a short phrase or between a numerical quantity and its corresponding unit(s).

```
Before starting the motor,
set the output speed to\~1.
There are 1,024\~bytes in 1\~KiB.
CSTR\~#8 documents the B\~language.
```

\~ is a GNU extension also supported by Heirloom Doctools *troff* 050915 (September 2005), *mandoc* 1.9.14 (2009-11-16), *neatroff* (commit 1c6ab0f6e, 2016-09-13), and Plan 9 from User Space *troff* (commit 93f8143600, 2022-08-12), but not by DWB or Solaris *troff*s.

\& Dummy character. Prefix an input line with it to prevent a dot or apostrophe from being interpreted as beginning a *roff* control line. Append **\&** to an end-of-sentence punctuation sequence to keep it from being recognized as such.

\| Thin space (one-sixth em on typesetters, zero-width on terminals); a non-breaking space. Used primarily in ellipses (".\|.\|.") to space the dots more pleasantly on typesetters.

\c End a text line without inserting space or attempting a break. Nothing on the input line after this escape sequence is formatted. **\c** is useful when three font styles are needed in a single word, as in a command synopsis.

```
.RB [ \-\-stylesheet=\c
.IR name ]
```

\c also helps when changing font styles in **EX/EE** examples, since they are not filled.

```
.EX
$ \c
.B groff \-T utf8 \-Z \c
.I file \c
.B | grotty \-i
.EE
```

Normally, if filling is enabled, the formatter treats the end of a text line like a space. It checks for the end of a sentence, and *may* break the output line (if not, it inserts an adjustable space). If filling is disabled, the formatter *will* break the output line, as in **EX/EE** examples. The formatter interprets the next input line as usual, recognizing control lines, including macro calls (contrast with *****_newline_).

The **\f** font selection escape sequence is an alternative to **\c**; see below. Using **\c** to continue a **TP** paragraph tag across multiple input lines renders incorrectly with *groff* 1.22.3, *mandoc* 1.14.1, older versions of these programs, and perhaps with some other formatters.

\e Format the *roff* escape character on the output; widely used in man pages to render a backslash glyph. It works reliably as long as the "ec" request is not used, which should never happen in man pages, and it is slightly more portable than the more explicit **\[rs]** ("reverse solidus") special character escape sequence.

\fB, **\fI**, **\fR**, **\fP**
Switch to bold, italic, roman, or back to the previous style, respectively. Either **\f** or **\c** is needed when three different font styles are required in a word.

```
.RB [ \-\-reference\-dictionary=\fI\,name\/\fP ]

.RB [ \-\-reference\-dictionary=\c
.IR name ]
```

Style escape sequences may be more portable than **\c**. As shown above, it is up to you to account for italic corrections with "**\/**" and "**\,**", which are themselves GNU extensions, if desired and if supported by your implementation.

\fP reliably returns to the style in use immediately preceding the previous **\f** escape sequence only if no sectioning, paragraph, example, or style macro calls have intervened.

As long as at most two styles are needed in a word, style macros like **B** and **BI** usually result in more readable *roff* source than **\f** escape sequences do.

Several special characters are also widely portable. Except for **\–**, **\[em]**, and **\[ga]**, AT&T *troff* did not consistently define the characters listed below, but its descendants, like DWB, Plan 9, or Solaris *troff*, can be made to support them by defining them in font description files, making them aliases of existing glyphs if necessary; see *groff_font*(5). *groff*'s extended notation for special characters, **\[***xx***]**, is also supported by *mandoc*(1), Heirloom Doctools *troff*, and *neatroff*, but not DWB, Plan 9, or Solaris *troff*s.

\– Minus sign. **\–** produces the basic Latin hyphen-minus (U+002D) specifying Unix command-line options and frequently used in file names. "**–**" is a hyphen in *roff*; some output devices format it as U+2010 (hyphen).

\[aq] Basic Latin neutral apostrophe. Some output devices format "**'**" as a right single quotation mark.

\[oq]

\[cq] Opening (left) and closing (right) single quotation marks. Use these for paired directional single quotes, ‘like this’.

\[dq] Basic Latin quotation mark (double quote). Employ it in macro calls to work around interpretation of ‘**"**’ as an argument delimiter.

```
.TP
.BI "split \[dq]" text \[dq]
```

\[lq]
\[rq] Left and right double quotation marks. Use these for paired directional double quotes, “like this”.

\[em] Em dash. Use for an interruption—such as this one—in a sentence.

\[en] En dash. Use to separate the ends of a range, as between numbers; for example, “the digits 1–9”.

\[ga] Basic Latin grave accent. Some output devices format “**`**” as a left single quotation mark.

\[ha] Basic Latin circumflex accent (“hat”). Some output devices format “**^**” as U+02C6 (modifier letter circumflex accent).

\[rs] Reverse solidus (backslash). The backslash is the default escape character in the *roff* language, so it does not represent itself in output. Also see **\e** above.

\[ti] Basic Latin tilde. Some output devices format “**~**” as U+02DC (small tilde).

For maximum portability, avoid escape sequences (including special characters) not listed above.

Hooks

Two macros, both GNU extensions, are called internally by the *groff man* package to format page headers and footers and can be redefined by the administrator in a site’s *man.local* file (see section “Files” below). The presentation of **TH** above describes the default headers and footers. Because these macros are hooks for *groff man* internals, man pages have no reason to call them. Such hook definitions typically consist of “sp” and “tl” requests. **PT** furthermore has the responsibility of emitting a PDF bookmark after writing the first page header in a document. Consult the existing implementations in *an.tmac* when drafting replacements.

.BT Set the page footer text (“bottom trap”).

.PT Set the page header text (“page trap”).

To remove a page header or footer entirely, define the appropriate macro as empty rather than deleting it.

Deprecated features

Use of the following in man pages for public distribution is discouraged.

.AT [*system* [*release*]]

Alter the footer for use with legacy AT&T man pages, overriding any definition of the *footer-inside* argument to **TH**. This macro exists only to render man pages from historical systems.

The inside footer is populated per the value of *system.*

3	7th edition *(default)*
4	System III
5	System V

The optional *release* argument specifies the release number, as in “System V Release 3”.

.DT Reset tab stops to the default (every 0.5i [inches]).

Use of this presentation-oriented macro is deprecated. It translates poorly to HTML, under which exact space control and tabulation are not readily available. Thus, information or distinctions that you use tab stops to express are likely to be lost. If you feel tempted to change the tab stops such that calling this macro later to restore them is desirable, consider composing a table using *tbl*(1) instead.

.OP *option-name* [*option-argument*]

Indicate an optional command parameter called *option-name*, which is set in bold. If the option takes an argument, specify *option-argument* using a noun, abbreviation, or hyphenated noun phrase. If present, *option-argument* is preceded by a space and set in italics. Square brackets in roman surround both arguments.

Use of this quasi-semantic macro, an extension whose name originated in DWB *troff*, is deprecated; *groff*'s interface differs. Neither can easily be used to annotate options that take optional arguments or options whose arguments have internal structure (such as a mixture of literal and variable components). One could work around these limitations with font selection escape sequences, but font style alternation macros are preferable; they are more flexible and perform italic corrections on typesetters.

.PD [*vertical-space*]

Configure the amount of vertical space between paragraphs or (sub)sections. The optional argument *vertical-space* specifies the amount; the default scaling unit is "v". Without an argument, inter-paragraph spacing resets to its default value; see subsection "Horizontal and vertical spacing" above.

Use of this presentation-oriented macro is deprecated. It translates poorly to HTML, under which exact control of inter-paragraph spacing is not readily available. Thus, information or distinctions that you use **PD** to express are likely to be lost.

.SB [*text*]

Set *text* in bold and (on typesetters) one point smaller than the default type size. Given no argument, **SB** plants a one-line input trap; text on the next line, which can be further formatted with a macro, is set smaller and in bold. Use of this macro, an extension originating in SunOS 4.0 *troff*, is deprecated. **SM** without an argument, followed immediately by "**B** *text*", produces the same output more portably. The macros' order is interchangeable; put *text* with the latter.

Note: terminals render *text* in bold at the normal size instead. Do not rely upon **SB** to communicate semantic information distinct from using bold style at normal size; it is hidden from readers using such devices.

.UC [*version*]

Alter the footer for use with legacy BSD man pages, overriding any definition of the *footer-inside* argument to **TH**. This macro exists only to render man pages from historical systems.

The inside footer is populated per the value of *version.*

3	3rd Berkeley Distribution *(default)*
4	4th Berkeley Distribution
5	4.2 Berkeley Distribution
6	4.3 Berkeley Distribution
7	4.4 Berkeley Distribution

History

M. Douglas McIlroy designed, implemented, and documented the AT&T *man* macros for Unix Version 7 (1979) and employed them to edit Volume 1 of its *Programmer's Manual*, a compilation of all man pages supplied by the system. The package supported the macros listed in this page not described as extensions, except **P** and the deprecated **AT** and **UC**. It documented no registers and defined only **R** and **S** strings.

UC appeared in 3BSD (1980). Unix System III (1980) introduced **P** and exposed the registers **IN** and **LL**, which had been internal to Seventh Edition Unix *man*. PWB/Unix 2.0 (1980) added the **Tm** string. 4BSD (1980) added **lq** and **rq** strings. SunOS 2.0 (1985) recognized **C**, **D**, **P**, and **X** registers. 4.3BSD (1986) added **AT** and **P**. Ninth Edition Unix (1986) introduced **EX** and **EE**. SunOS 4.0 (1988) added **SB**. Unix System V (1988) incorporated the **lq** and **rq** strings.

Except for **EX/EE**, James Clark implemented the foregoing features in early versions of *groff*. Later, *groff* 1.20 (2009) resurrected **EX/EE** and originated **SY/YS**, **TQ**, **MT/ME**, and **UR/UE**. Plan 9 from User Space's *troff* introduced **MR** in 2020, and incorporated the **lq** and **rq** strings in 2025.

Options

The following *groff* options set registers (with **−r**) and strings (with **−d**) recognized and used by the *man* macro package. To ensure rendering consistent with output device capabilities and reader preferences, man pages should never manipulate them.

−dAD=*adjustment-mode*
: Set line adjustment to *adjustment-mode,* which is typically "**b**" for adjustment to both margins (the default), or "**l**" for left alignment (ragged right margin). Any valid argument to *groff*'s "ad" request may be used. See *groff*(7) for less-common choices.

−rBP=*base-paragraph-inset*
: Set the inset amount for ordinary paragraphs not within an **RS**/**RE** inset. The default is 5n.

−rcR=1
: Enable continuous rendering. Output is not paginated; instead, one (potentially very long) page is produced. This is the default for terminal and HTML devices. Use **−rcR=0** to disable it on terminals; on HTML devices, it cannot be disabled.

−rC1
: Number output pages consecutively, in strictly increasing sequence, rather than resetting the page number to 1 (or the value of register **P**) with each new *man* document.

−rCHECKSTYLE=*n*
: Report problems with usage of this macro package exhibited by the input at verbosity level *n*, where *n* is an integer in the range 0–3, inclusive; **0** disables the messages and is the default. This feature is a development and debugging aid for man page maintainers; the problems diagnosed, and range and meanings of the supported levels, are subject to change.

−rCS=1
: Set section headings (the argument(s) to **SH**) in full capitals. This transformation is off by default because it discards lettercase distinctions.

−rCT=1
: Set the man page identifier (the first argument to **TH**) in full capitals in headers and footers. This transformation is off by default because it discards lettercase distinctions.

−rD1
: Enable double-sided layout, formatting footers for even and odd pages differently; see the description of **TH** in subsection "Document structure macros" above.

−rFT=*footer-distance*
: Set distance of the footer relative to the bottom of the page to *footer-distance;* this amount is always negative. At one half-inch above this location, the page text is broken before writing the footer. Ignored if continuous rendering is enabled. The default is "−0.5i − 1v".

−dHF=*heading-font*
: Select the font used for section and subsection headings; the default is "**B**" (bold style of the default family). Any valid argument to *groff*'s "ft" request may be used. See *groff*(7).

−rHY=0
: Disable automatic hyphenation. Normally, it is enabled (1). The hyphenation mode is determined by the *groff* locale; see section "Localization" of *groff*(7).

−rIN=*standard-indentation*
: Set the default indentation amount used by **IP**, **TP**, and **HP**, and the inset amount used by **RS**. The default is 7n on terminals and 7.2n on typesetters. Use only integer multiples of unit "n" on terminals for consistent indentation.

−rLL=*line-length*
: Set line length; the default is 80n on terminals and 6.5i on typesetters.

−rLT=*title-length*
: Set the line length for titles. ("Titles" is the *roff* term for headers and footers.) By default, it is set to the line length (see **−rLL** above).

−dMF=*man-page-topic-font*
: Select the font used for man page identifiers in **TH** calls and topics named in **MR** calls; the default is "**I**" (italic style of the default family). Any valid argument to *groff*'s "ft" request may be used. If the **MF** string ends in "I", the package assumes it to be an oblique typeface, and applies italic corrections before and after man page topics and identifiers.

−rP*n*
: Start enumeration of pages at *n*. The default is 1.

−rPO=*page-offset*
: Set page offset; the default is 0 on terminals and 1i on typesetters.

−rS*type-size*
: Use *type-size* for the document's body text; acceptable values are 10, 11, or 12 points. See subsection "Font style macros" above for the default.

−rSN=*subsection-indentation*
: Set indentation of subsection headings to *subsection-indentation.* The default is 3n.

−rTS=*separation*
: Require the given *separation* between a **TP** paragraph's tag and its body. The default is 2n.

−rU0
: Disable generation of URI hyperlinks in output drivers capable of them, making the arguments to **MT** and **UR** calls visible as formatted text. *grohtml*(1), *gropdf*(1), and *grotty*(1) enable hyperlinks by default (the last only if not in its legacy output mode).

−rX*p*
: Number successors of page *p* as *p*a, *p*b, *p*c, and so forth. The register tracking the suffixed page letter uses format "a" (see the "af" request in *groff*(7)). For example, the option **−rX2** produces the following page numbers: 1, 2, 2a, 2b, . . . , 2aa, 2ab, and so on.

Files

/usr/share/groff/1.24.2/tmac/an.tmac
: Most *man* macros are defined in this file. It also loads extensions from *an−ext.tmac* (see below).

/usr/share/groff/1.24.2/tmac/andoc.tmac
: This brief *groff* program detects whether the *man* or *mdoc* macro package is used by a document and loads the correct macro definitions, taking advantage of the fact that pages using them must call **TH** or **Dd**, respectively, before any other macros. A *man* program or a user typing, for example, "**groff −mandoc page.1**", need not know which package the file *page.1* uses. Multiple man pages, in either format, can be handled; *andoc* reloads each macro package as necessary. Page-local redefinitions of names used by the *man* or *mdoc* packages prior to **TH** or **Dd** calls are "clobbered" by the reloading process. If you want to provide your own definition of an extension macro to ensure its availability, the *an−ext.tmac* entry below offers advice.

/usr/share/groff/1.24.2/tmac/an−ext.tmac
: Definitions of macros described above as extensions (and not deprecated) are contained in this file; in some cases, they are simpler versions of definitions appearing in *an.tmac*, and are ignored if the formatter is GNU *troff*. They are written to be compatible with AT&T *troff* and permissively licensed—not copylefted. To reduce the risk of name space collisions, string and register names begin only with "**m**". We encourage man page authors who are concerned about portability to legacy Unix systems to copy these definitions into their pages, and maintainers of *troff* implementations or workalike systems that format man pages to re-use them. To ensure reliable rendering, define them after your page calls **TH**; see the discussion of *andoc.tmac* above. Further, it is wise to define such page-local macros (if at all) after the "Name" section to accommodate timid *makewhatis*(8) or *mandb*(8) implementations that easily give up scanning for indexing material.

/usr/share/groff/1.24.2/tmac/man.tmac
: is a wrapper enabling the package to be loaded with the option "**−m man**".

/usr/share/groff/1.24.2/tmac/mandoc.tmac
: is a wrapper enabling *andoc.tmac* to be loaded with the option "**−m mandoc**".

/etc/groff/site−tmac/man.local
: Put site-local changes and customizations into this file.

```
.\" Put only one space after the end of a sentence.
.ss 12 0 \" See groff(7).
.\" Keep pages narrow even on wide terminals.
.if n .if \n[LL]>80n .nr LL 80n
```

On multi-user systems, it is more considerate to users whose preferences may differ from the administrator's to be less aggressive with such settings, or to permit their override with a user-specific *man.local* file. Place the requests below at the end of the site-local file to manifest courtesy.

```
.soquiet \V[XDG_CONFIG_HOME]/man.local
.soquiet \V[HOME]/.man.local
```

However, a security-sandboxed *man*(1) program may lack permission to open such files.

Notes

Some tips on composing and troubleshooting your man pages follow.

- What's the difference between a man page topic and identifier?

  A single man page may document several related but distinct topics. For example, *printf*(3) and *fprintf*(3) are often presented together. Moreover, multiple programming languages have functions named "printf", and may document these in a man page. The identifier is intended to (with the section) uniquely identify a page on the system; it may furthermore correspond closely to the file name of the document.

  The *man*(1) librarian makes access to man pages convenient by resolving topics to man page identifiers. Thus, you can type "**man fprintf**", and other pages can refer to it, without knowing whether the installed document uses "printf", "fprintf", or even "c_printf" as its identifier.

- Some ASCII characters look funny or copy and paste wrong.

  On devices with large glyph repertoires, like UTF-8-capable terminals and PDF, GNU *troff*, like AT&T *troff* before it, maps several keycaps to code points outside the Unicode basic Latin range (historically "ASCII") because that usually results in better typography in the general case. When documenting GNU/Linux command or C language syntax, however, this translation is sometimes not desirable.

To get a "literal"...	...should be input.
'	\[aq]
−	\-
\	\[rs]
^	\[ha]
`	\[ga]
~	\[ti]

  If a neutral double quote (") is needed in a macro argument, you can use **\[dq]** to get it. Do *not* use **\[aq]** for an ordinary apostrophe (as in "can't") or **\-** for an ordinary hyphen (as in "word-aligned").

- Do I ever need to use an empty macro argument ("")?

  Probably not. When this seems necessary, often a shorter or clearer alternative is available.

Instead of...	**...should be considered.**
`.TP ""`	`.TP`
`.BI "" italic-text bold-text`	`.IB italic-text bold-text`
`.TH foo 1 "" "foo 1.2.3"`	`.TH foo 1 yyyy-mm-dd "foo 1.2.3"`
`.IP "" 4n`	`.IP`

`.IP "" 4n`
paragraph
...
...
	`.RS 4n`
	`.P`
	paragraph
	`.RE`

`.B one two "" three`	`.B one two three`

In the title heading (**TH**), the date of the page's last revision is more important than packaging information; it should not be omitted. Ideally, a page maintainer keeps both up to date.

IP is sometimes ill-understood and misused, especially when no mark argument is supplied—an indentation argument is not required. By setting an explicit indentation, you may be overriding the reader's preference as set with the **-rIN** option. If your page renders adequately without one, use the simpler form. If you need to indent multiple (unmarked) paragraphs, consider setting an inset region with **RS** and **RE** instead.

In the last example, the empty argument does have a subtly different effect than its suggested replacement: the empty argument causes an additional space character to be interpolated between the arguments "two" and "three"—but it is a regular breaking space, so it can be discarded at the end of an output line. It is better not to be subtle, particularly with space, which can be overlooked in source and rendered forms.

- **RS** doesn't indent relative to my indented paragraph.

  The **RS** macro determines the *inset amount,* the position at which an *ordinary* paragraph (**P** and its synonyms) is set; the value of the **IN** register determines its default amount. This register also determines the default indentation used by **IP**, **TP**, and **HP**. To create an inset relative to an indented paragraph, call **RS** repeatedly until an acceptable indentation is achieved, or give **RS** an indentation argument that is at least as much as the paragraph's indentation amount relative to an adjacent ordinary (**P**) paragraph.

  Another approach to tagged paragraphs places an **RS** call immediately after the tag; this also forces a break regardless of the tag's width, which some authors prefer. Follow-up paragraphs under the tag can then be set with **P** instead of **IP**. Remember to use **RE** to end the indented region before starting the next tagged paragraph (at the appropriate nesting level).

- **RE** doesn't move the inset back to the expected level.

  RE takes an inset *level* as an argument, unlike **RS**'s inset *amount.* "**.RE 1**" goes to the level before any **RS** macros were called, **RE 2** goes to the level of the first **RS** call you made, and so forth. If you desire symmetry in your macro calls, simply issue one **RE** without an argument for each **RS** that precedes it.

  The **SH** and **SS** sectioning macros clear relative insets; **RE** calls have no effect until **RS** is used again.

- Do I need to keep typing the indentation in a series of **IP** calls?

  Not if you don't want to change it. Review subsection "Horizontal and vertical spacing" above.

Instead of...	**...should be considered.**

`.IP \[bu] 4n`
paragraph
`.IP \[bu] 4n`
another-paragraph
	`.IP \[bu] 4n`
	paragraph
	`.IP \[bu]`
	another-paragraph

- Why doesn't the package provide a string to insert an ellipsis?

  Examples of ellipsis usage are shown above, in subsection "Synopsis macros". The idiomatic *roff* ellipsis is three dots (periods) with thin space escape sequences \| internally separating them. Since dots both begin control lines and are candidate end-of-sentence characters, however, it is sometimes necessary to prefix

and/or suffix an ellipsis with the dummy character escape sequence **\&**. That fact stands even if a string is defined to contain the sequence; further, if the string ends with **\&**, end-of-sentence detection is defeated when you use the string at the end of an actual sentence. (Ending a sentence with an ellipsis is often poor style, but not always.) A hypothetical string **EL** that contained an ellipsis, but not the trailing dummy character **\&**, would then need to be suffixed with the latter when not ending a sentence.

Instead of...	**...do this.**
`.ds EL \&.|.|.`	`Arguments are`
`Arguments are`	`.IR src-file\~ .|.|.\&`
`.IR src-file\~ \*(EL\&`	`.IR dest-dir .`
`.IR dest-dir .`	

The first column practices a false economy; the savings in typing is offset by the cost of obscuring even the suggestion of an ellipsis to a casual reader of the source document, and reduced portability to non-*roff* man page formatters that cannot handle string definitions.

Unicode defines an ellipsis code point, and some fonts have an ellipsis glyph, which some man pages have accessed non-portably with the font-dependent **\N** escape sequence. We discourage their use; on terminals, they may crowd the dots into a half-width character cell, and do not render at all if the output device lacks the glyph. In synopses, missing ellipses can mislead the reader. Dots and space are universally supported.

- When and how should I use quotation marks?

  As noted above in subsection "Font style macros", apply quotation marks to "brief specimens of literal text, such as article titles, inline examples, mentions of individual characters or short strings, and (sub)section headings in man pages". Multi-word literals, such as Unix commands with arguments, when set inline (as opposed to displayed between **EX** and **EE**), should be quoted to ensure that the boundaries of the literal are clear even when the material is stripped of font styling by, for example, copy-and-paste operations. *groff*, Heirloom Doctools *troff*, *neatroff*, and *mandoc* support all of the special characters **\[oq]**, **\[cq]**, **\[lq]**, **\[rq]**, **\[aq]**, and **\[dq]** described in subsection "Portability" above. DWB, Plan 9, and Solaris *troff*s do not. Interpolating the strings ***(lq** and ***(rq** portably yields directional double quotation marks, if available, in all these formatters (though *neatroff* does not supply a *man* macro package), but they cannot reliably be used in macro arguments.

  Obtaining directional single quotation marks is more of a challenge. Historically, man pages used ` and **'**, which *troff* rendered on typesetters as ‘ and ’, exclusively for them. However, in recent years, some distributors of *groff* have chosen to override the meanings of these characters in man pages, remapping them to their Unicode Basic Latin code points. Unfortunately, ` and ' are the *only* reliable means of obtaining directional single quotation marks in AT&T *troff*; in that implementation, often no special character escape sequences exist to obtain them. Further, AT&T *troff*'s special character identifiers, like its font names, were device-specific. To achieve quotation portably in man pages rendered both by AT&T and more modern *troff*s, consider adding a preamble to your page after the **TH** call as follows.

  ```
  .ie \n(.g \{\
  .  ds oq \[oq]\"
  .  ds cq \[cq]\"
  .\}
  .el \{\
  .  ds oq `\"
  .  ds cq '\"
  .\}
  ```

  You must then use the ***** escape sequence to interpolate the quotation mark strings.

  ```
  The command
  .RB \*(oq "while !\& git pull; do sleep 10; done" \*(cq
  retries an update from the repository until it succeeds.
  ```

  If this procedure seems complex, petition your distributor to revert their remapping of the ` and ' characters.

- Escape sequences of the form **\[***xx***]** don't format correctly.

  The **\[***xx***]** special character escape sequence is a GNU *troff* extension also supported by *mandoc,* Heirloom Doctools *troff,* and *neatroff.* DWB, Plan 9, and Solaris *troff*s don't implement it. If your man page requires portability to these formatters, spell such escape sequences as "**\(***xx*"; no closing parenthesis is used. *xx* must be exactly two characters; *groff_char*(7) lists portable special character identifiers.

Authors

James Clark wrote the initial GNU implementation of the *man* macro package. Later, Werner Lemberg supplied the **S**, **LT**, and **cR** registers, the last a 4.3BSD-Reno *mdoc*(7) feature. Larry Kollar added the **FT**, **HY**, and **SN** registers; the **HF** string; and the **PT** and **BT** macros in *groff* 1.19 (2003). Lemberg and Eric S. Raymond contributed **EX/EE**, **MT/ME**, **UR/UE**, **TQ**, and an early version of the **SY/YS** macros to *groff* 1.20 (2009). G. Branden Robinson implemented the **AD** and **MF** strings; **CS**, **CT**, and **U** registers; and the **MR** macro for *groff* 1.23 (2023), and the **BP**, **PO**, and **TS** registers and a revised implementation of the **SY/YS** macros for *groff* 1.24 (2026).

Susan G. Kleinmann wrote the initial version of this document for the Debian GNU/Linux system. Lemberg imported it to *groff.* He and Robinson revised and updated it. Raymond and Robinson documented the extension macros. Raymond also originated the portability section, to which Ingo Schwarze contributed most of the material on escape sequences.

See also

tbl(1), *eqn*(1), and *refer*(1) are preprocessors used with man pages. *man*(1) describes the man page librarian on your system. *groff_mdoc*(7) details the *groff* version of BSD's alternative macro package for man pages.

groff_man(7), *groff*(7), *groff_char*(7)

Name

groff_mdoc — compose BSD-style manual (man) pages with GNU *roff*

Synopsis

groff **−mdoc** *file* ...

Description

The GNU implementation of the *mdoc* macro package is part of the *groff*(1) document formatting system. *mdoc* is a structurally- and semantically-oriented package for writing Unix manual pages with *troff*(1). Its predecessor, the *man*(7) package, primarily addressed page layout and presentational concerns, leaving the selection of fonts and other typesetting details to the individual author. This discretion has led to divergent styling practices among authors using it.

mdoc organizes its macros into *domains*. The *page structure domain* lays out the page and comprises titles, section headings, displays, and lists. The *general text domain* supplies macros to quote or style text, or to interpolate common noun phrases. The *manual domain* offers semantic macros corresponding to the terminology used by practitioners in discussion of Unix commands, routines, and files. Manual domain macros distinguish command-line arguments and options, function names, function parameters, pathnames, variables, cross references to other manual pages, and so on. These terms are meaningful both to the author and the readers of a manual page. It is hoped that the resulting increased consistency of the man page corpus will enable easier translation to future documentation tools.

Throughout Unix documentation, a manual entry is referred to simply as a "man page", regardless of its length, without gendered implication, and irrespective of the macro package selected for its composition.

Getting started

The *mdoc* package attempts to simplify man page authorship and maintenance without requiring mastery of the *roff* language. This document presents only essential facts about *roff*. For further background, including a discussion of basic typographical concepts like "breaking", "filling", and "adjustment", see *roff*(7). Specialized units of measurement also arise, namely ens, vees, inches, and points, abbreviated "n", "v", "i", and "p", respectively; see section "Measurements" of *groff*(7).

For brief examples, we employ an arrow notation illustrating a transformation of input on the left to rendered output on the right. Consider the .**Dq** macro, which double-quotes its arguments.

.Dq man page → "man page"

Usage

An *mdoc macro* is *called* by placing the *roff* control character, **.** (dot) at the beginning of a line followed by its name. In this document, we often discuss a macro name with this leading dot to identify it clearly, but the dot is *not* part of its name. Space or tab characters can separate the dot from the macro name. Arguments may follow, separated from the macro name and each other by spaces, but *not* tabs. The dot at the beginning of the line prepares the formatter to expect a macro name. A dot followed immediately by a newline is ignored; this is called the *empty request*. To begin an input line with a dot (or a neutral apostrophe **'**) in some context other than a macro call, precede it with the `\&` escape sequence; this is a dummy character, not formatted for output. The backslash is the *roff* escape character; it can appear anywhere and it always followed by at least one more character. If followed by a newline, the backslash escapes the input line break; you can thus keep input lines to a reasonable length without affecting their interpretation.

Macros in GNU *troff* accept an unlimited number of arguments; other *troff*s often can't handle more than nine. In some cases, arguments may continue or extend onto the next input line without resort to the `\newline` escape sequence; see subsection "Extended arguments" below. Neutral double quotes **"** can be used to group multiple words into an argument; see subsection "Passing space characters in an argument" below.

Most of *mdoc*'s general text and manual domain macros *parse* their argument lists for *callable* macro names. This means that an argument in the list matching a general text or manual domain macro name (and defined to be callable) will be called with the remaining arguments when it is encountered. In such cases, the argument, although the name of a macro, is not preceded by a dot. Macro calls can thus be nested. This approach to macro argument processing is a unique characteristic of the *mdoc* package, not a general feature of *roff* syntax.

For example, the option macro, **.Op**, may call the flag and argument macros, **.Fl** and **.Ar**, to specify an optional flag with an argument.

.Op Fl s Ar bytes → [**−s** *bytes*]

To prevent a word from being interpreted as a macro name, precede it with the dummy character.

.Op \&Fl s \&Ar bytes → [Fl s Ar bytes]

In this document, a macro that parses its argument list for other macro names is termed *parsed*; macros that permit other macros to call them thus are described as *callable*. This usage is a technical *faux pas*, since all *mdoc* macros are in fact interpreted (unless prevented with `\&`), but as it is cumbersome to constantly refer to macros as "being able to call other macros", we employ the term "parsed" instead. Except where explicitly stated, all *mdoc* macros are parsed and callable.

In the following, we term an *mdoc* macro that starts a line (with a leading dot) a *command* if a distinction from those appearing as arguments of other macros is necessary.

Passing space characters in an argument

Sometimes it is desirable to give a macro an argument containing one or more space characters, for instance to specify a particular arrangement of arguments demanded by the macro. Additionally, quoting multi-word arguments that are to be treated the same makes *mdoc* work faster; macros that parse arguments do so once (at most) for each. For example, the function command **.Fn** expects its first argument to be the name of a function and any remaining arguments to be function parameters. Because C language standards mandate the inclusion of types *and* identifiers in the parameter lists of function definitions, each `Fn` parameter after the first will be at least two words in length, as in "*int foo*".

There are a few ways to embed a space in a macro argument. One is to use the unadjustable space escape sequence *****space*. The formatter treats this escape sequence as if it were any other printable character, and will not break a line there as it would a word space when the output line is full. This method is useful for macro arguments that are not expected to straddle an output line boundary, but has a drawback: this space does not adjust as others do when the output line is formatted. An alternative is to use the unbreakable space escape sequence, `\~`, which cannot break but does adjust. This *groff* extension is widely but not perfectly portable. Another method is to enclose the string in double quotes.

.Fn fetch char\ *str → **fetch**(*char *str*)
.Fn fetch char\~*str → **fetch**(*char *str*)
.Fn fetch "char *str" → **fetch**(*char *str*)

If the `\` before the space in the first example or the double quotes in the third example were omitted, `.Fn` would see three arguments, and the result would contain an undesired comma.

.Fn fetch char *str → **fetch**(*char* , **str*)

Trailing space characters

It is wise to remove trailing spaces from the ends of input lines. Should the need arise to put a formattable space at the end of a line, do so with the unadjustable or unbreakable space escape sequences.

Formatting the backslash and other glyphs

When you need the *roff* escape character `\` to appear in the output, use `\e` or `\[rs]` instead. Strictly, `\e` formats the current escape character; it works reliably as long as no *roff* request is used to change it, which should never happen in man pages. `\[rs]` is a *groff special character* escape sequence that explicitly formats the "reverse solidus" (backslash) glyph. In general, use special character escape sequences to format characters outside the ISO 646 ("ASCII") or Unicode Basic Latin range, and to format correct glyphs for the characters '"', "", '−', '^', '`', and '~'. *groff_char*(7) presents the formatter's special character identifiers, annotating those that are portable to legacy systems. Section "Predefined strings" below lists characters commonly used by *mdoc* documents.

Other possible pitfalls

groff mdoc warns when an empty input line is found outside of a *display*, a topic presented in subsection "Examples and displays" below. Use empty requests to space the source document for maintenance.

Leading spaces cause a break and are formatted. Avoid this behaviour if possible. Similarly, do not put more than one space between words in an ordinary text line; they are not "normalized" to a single space as other text formatters might do.

Don't try to use the neutral double quote character **"** to represent itself in an argument. Use the special character escape sequence `\[dq]` to format it. Avoid using `\[dq]` for conventional quotation; see subsection "Enclosure and quoting macros" below.

If your document must be portable to legacy systems like 4.4BSD, special character escape sequences shown here in the form **\[***xx***]** must be spelled **\(***xx* instead. No version of *groff* or *mandoc* requires the latter form.

As a typesetting system, *roff* distinguishes hyphens, minus signs, and dashes of various widths. When inputting a hyphen or dash ("hyphen-minus") (Unicode U+002D) that should be usable in URLs or copy-and-paste operations to shell prompts or program code, prefix it with the escape character: `\-`.

The formatter attempts to detect the ends of sentences and by default puts the equivalent of two spaces between sentences on the same output line; see *roff*(7). To defeat this detection in a parsed list of macro arguments, put `\&` before the punctuation mark. Thus,

```
The
.Ql .
character.
.Pp
The
.Ql \&.
character.
.Pp
.No test .
test
.Pp
.No test.
test
```

gives

The . character

The . character.

test. test

test. test

as output. As can be seen in the first and third output lines, *mdoc* handles punctuation characters specially in macro arguments. This will be explained in section "General syntax" below.

A comment in the source file of a man page can begin with **.\"** at the start of an input line, **\"** after other input, or `\#` anywhere (the last is a *groff* extension); the remainder of any such line is ignored.

A man page template

Use *mdoc* to construct a man page from the following template.

```
.\" The following three macro calls are required.
.Dd date
.Dt identifier [section-id [section-keyword-or-title]]
.Os [package-or-operating system [version-or-release]]
.Sh Name
.Nm topic
.Nd summary-description
.\" The next heading is used in sections 2 and 3.
.\" .Sh Library
.\" The next heading is used in sections 1-4, 6, 8, and 9.
.Sh Synopsis
```

```
.Sh Description
.\" Uncomment and populate the following sections as needed.
.\" .Sh "Implementation notes"
.\" The next heading is used in sections 2, 3, and 9.
.\" .Sh "Return values"
.\" The next heading is used in sections 1, 3, 6, and 8.
.\" .Sh Environment
.\" .Sh Files
.\" The next heading is used in sections 1, 6, and 8.
.\" .Sh "Exit status"
.\" .Sh Examples
.\" The next heading is used in sections 1, 4, 6, 8, and 9.
.\" .Sh Diagnostics
.\" .Sh Compatibility
.\" The next heading is used in sections 2, 3, 4, and 9.
.\" .Sh Errors
.\" .Sh "See also"
.\" .Sh Standards
.\" .Sh History
.\" .Sh Authors
.\" .Sh Caveats
.\" .Sh Bugs
```

The first items in the template are the commands .**Dd**, .**Dt**, and .**Os**. They identify the page and are discussed below in section "Title macros".

The remaining items in the template are section headings (.**Sh**); of which "Name" and "Description" are mandatory. These headings are discussed in section "Page structure domain", which follows section "Manual domain". Familiarize yourself with manual domain macros first; we use them to illustrate the use of page structure domain macros.

Conventions

In the descriptions of macros below, square brackets surround optional arguments. An ellipsis ('. . .') represents repetition of the preceding argument zero or more times. Alternative values of a parameter are separated with |. If a mandatory parameter can take one of several alternative values, use braces to enclose the set, with spaces and | separating the items.

ztar {**c** | **x**} [**−w** [**−y** | **−z**]] [**−f** *archive*] *member* . . .

An alternative to using braces is to separately synopsize distinct operation modes, particularly if the list of valid optional arguments is dependent on the user's choice of a mandatory parameter.

ztar c [**−w** [**−y** | **−z**]] [**−f** *archive*] *member* . . .
ztar x [**−w** [**−y** | **−z**]] [**−f** *archive*] *member* . . .

Most macros affect subsequent arguments until another macro or a newline is encountered. For example, '**.Li ls Bq Ar file**' doesn't produce '**ls [file]**', but '**ls** [*file*]'. Consequently, a warning message is emitted for many commands if the first argument is itself a macro, since it cancels the effect of the preceding one. On rare occasions, you might want to format a word along with surrounding brackets as a literal.

.Li "ls [file]" → **ls [file]** *# list any files named e, f, i, or l*

Many macros possess an implicit width, used when they are contained in lists and displays. If you avoid relying on these default measurements, you escape potential conflicts with site-local modifications of the *mdoc* package. Explicit **−width** and **−offset** arguments to the .**Bl** and .**Bd** macros are preferable.

Title macros

We present the **mandatory** title macros first due to their importance even though they formally belong to the page structure domain macros. They designate the page identifier and section of the manual, date of last revision, and the operating system or software project associated with the page. Call each once at the beginning of the document. They populate the page headers and footers, "titles" in *roff* parlance.

.Dd *date*

This first macro of any *mdoc* manual records the last modification date of the document source. Arguments are catenated and separated with space characters.

Historically, *date* was written in U.S. traditional format, "*Month day , year*" where *Month* is the full month name in English, *day* an integer without a leading zero, and *year* the four-digit year. This localism is not enforced, however. *groff* recommends ISO 8601 format, *YYYY-MM-DD*. A *date* of the form '**$Mdocdate:** *Month day year* **$**' is also recognized. It is used in OpenBSD manuals to automatically insert the current date when committing.

This macro is neither callable nor parsed.

.Dt *identifier* [*section-id* [*section-keyword-or-title*]]

identifier and *section-id* together should uniquely indicate the man page on the system. A *section-id* that begins with an integer in the range 1–9 or is one of the words `unass`, `draft`, or `paper` selects a predefined section title. This use of "section" has nothing to do with the section headings otherwise discussed in this page; it arises from the organizational scheme of printed and bound Unix manuals.

In this implementation, the following titles are defined for integral section numbers.

`1`	General Commands Manual
`2`	System Calls Manual
`3`	Library Functions Manual
`4`	Kernel Interfaces Manual
`5`	File Formats Manual
`6`	Games Manual
`7`	Miscellaneous Information Manual
`8`	System Manager's Manual
`9`	Kernel Developer's Manual

A section title may be arbitrary or one of the following abbreviations.

`USD`	User's Supplementary Documents
`PS1`	Programmer's Supplementary Documents
`AMD`	Ancestral Manual Documents
`SMM`	System Manager's Manual
`URM`	User's Reference Manual
`PRM`	Programmer's Manual
`KM`	Kernel Manual
`IND`	Manual Master Index
`LOCAL`	Local Manual
`CON`	Contributed Software Manual

For compatibility, `MMI` can be used for `IND`, and `LOC` for `LOCAL`. Values from the previous table will specify a new section title. If *section-keyword-or-title* designates a computer architecture recognized by *groff mdoc*, its value is prepended to the default section title as specified by the second parameter. By default, the following architecture keywords are defined.

acorn26, acorn32, algor, alpha, amd64, amiga, amigappc, arc, arm, arm26, arm32, armish, atari, aviion, beagle, bebox, cats, cesfic, cobalt, dreamcast, emips, evbarm, evbmips, evbppc, evbsh3, ews4800mips, hp300, hp700, hpcarm, hpcmips, hpcsh, hppa, hppa64, i386, ia64, ibmnws, iyonix, landisk, loongson, luna68k, luna88k, m68k, mac68k, macppc, mips, mips64, mipsco, mmeye, mvme68k, mvme88k, mvmeppc, netwinder, news68k, newsmips, next68k, ofppc, palm, pc532, playstation2, pmax, pmppc, powerpc, prep, rs6000, sandpoint, sbmips, sgi, sgimips, sh3, shark, socppc, solbourne, sparc, sparc64, sun2, sun3, tahoe, vax, x68k, x86_64, xen, zaurus

If a section title is not determined after the above matches have been attempted, *section-keyword-or-title* is used.

The effects of varying `.Dt` arguments on the page header content are shown below. Observe how `\&` prevents the numeral 2 from being used to look up a predefined section title.

`.Dt foo 2`	→	foo(2)	System Calls Manual	foo(2)
`.Dt foo 2 m68k`	→	foo(2)	m68k System Calls Manual	foo(2)
`.Dt foo 2 baz`	→	foo(2)	System Calls Manual	foo(2)
`.Dt foo \&2 baz`	→	foo(2)	baz	foo(2)
`.Dt foo "" baz`	→	foo	baz	foo
`.Dt foo M Z80`	→	foo(M)	Z80	foo(M)

roff strings define section titles and architecture identifiers. Site-specific additions might be found in the file *mdoc.local*; see section "Files" below.

This macro is neither callable nor parsed.

.Os [*operating-system-or-package-name* [*version-or-release*]]

This macro associates the document with a software distribution. When composing a man page to be included in the base installation of an operating system, do not provide an argument; *mdoc* will supply it. In this implementation, that default is "GNU". It may be overridden in the site configuration file, *mdoc.local*; see section "Files" below. A portable software package maintaining its own man pages can supply its name and version number or release identifier as optional arguments. A *version-or-release* argument should use the standard nomenclature for the software specified. In the following table, recognized *version-or-release* arguments for some predefined operating systems are listed. As with **.Dt**, site additions might be defined in *mdoc.local*.

ATT
: 7th, 7, III, 3, V, V.2, V.3, V.4

BSD
: 3, 4, 4.1, 4.2, 4.3, 4.3t, 4.3T, 4.3r, 4.3R, 4.4

NetBSD
: 0.8, 0.8a, 0.9, 0.9a, 1.0, 1.0a, 1.1, 1.2, 1.2a, 1.2b, 1.2c, 1.2d, 1.2e, 1.3, 1.3a, 1.4, 1.4.1, 1.4.2, 1.4.3, 1.5, 1.5.1, 1.5.2, 1.5.3, 1.6, 1.6.1, 1.6.2, 1.6.3, 2.0, 2.0.1, 2.0.2, 2.0.3, 2.1, 3.0, 3.0.1, 3.0.2, 3.0.3, 3.1, 3.1.1, 4.0, 4.0.1, 5.0, 5.0.1, 5.0.2, 5.1, 5.1.2, 5.1.3, 5.1.4, 5.2, 5.2.1, 5.2.2, 6.0, 6.0.1, 6.0.2, 6.0.3, 6.0.4, 6.0.5, 6.0.6, 6.1, 6.1.1, 6.1.2, 6.1.3, 6.1.4, 6.1.5, 7.0, 7.0.1, 7.0.2, 7.1, 7.1.1, 7.1.2, 7.2, 8.0, 8.1

FreeBSD
: 1.0, 1.1, 1.1.5, 1.1.5.1, 2.0, 2.0.5, 2.1, 2.1.5, 2.1.6, 2.1.7, 2.2, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8, 2.2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 4.0, 4.1, 4.1.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.6.2, 4.7, 4.8, 4.9, 4.10, 4.11, 5.0, 5.1, 5.2, 5.2.1, 5.3, 5.4, 5.5, 6.0, 6.1, 6.2, 6.3, 6.4, 7.0, 7.1, 7.2, 7.3, 7.4, 8.0, 8.1, 8.2, 8.3, 8.4, 9.0, 9.1, 9.2, 9.3, 10.0, 10.1, 10.2, 10.3, 10.4, 11.0, 11.1, 11.2, 11.3, 12.0, 12.1

OpenBSD
: 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6

DragonFly
: 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.8.1, 1.9, 1.10, 1.11, 1.12, 1.12.2, 1.13, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.9.1, 2.10, 2.10.1, 2.11, 2.12, 2.13, 3.0, 3.0.1, 3.0.2, 3.1, 3.2, 3.2.1, 3.2.2, 3.3, 3.4, 3.4.1, 3.4.2, 3.4.3, 3.5, 3.6, 3.6.1, 3.6.2, 3.7, 3.8, 3.8.1, 3.8.2, 4.0, 4.0.1, 4.0.2, 4.0.3, 4.0.4, 4.0.5, 4.0.6, 4.1, 4.2, 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.3, 4.4, 4.4.1, 4.4.2, 4.4.3, 4.5, 4.6, 4.6.1, 4.6.2, 4.7, 4.8, 4.8.1, 4.9, 5.0, 5.0.1, 5.0.2, 5.1, 5.2, 5.2.1, 5.2.2, 5.3, 5.4, 5.4.1, 5.4.2, 5.4.3, 5.5, 5.6, 5.6.1, 5.6.2

Darwin
: 8.0.0, 8.1.0, 8.2.0, 8.3.0, 8.4.0, 8.5.0, 8.6.0, 8.7.0, 8.8.0, 8.9.0, 8.10.0, 8.11.0, 9.0.0, 9.1.0, 9.2.0, 9.3.0, 9.4.0, 9.5.0, 9.6.0, 9.7.0, 9.8.0, 10.0.0, 10.1.0, 10.2.0, 10.3.0, 10.4.0, 10.5.0, 10.6.0, 10.7.0, 10.8.0, 11.0.0, 11.1.0, 11.2.0, 11.3.0, 11.4.0, 11.5.0, 12.0.0, 12.1.0, 12.2.0, 13.0.0, 13.1.0, 13.2.0, 13.3.0, 13.4.0, 14.0.0, 14.1.0, 14.2.0, 14.3.0, 14.4.0, 14.5.0, 15.0.0, 15.1.0, 15.2.0, 15.3.0, 15.4.0, 15.5.0, 15.6.0, 16.0.0, 16.1.0, 16.2.0, 16.3.0, 16.4.0, 16.5.0, 16.6.0,

17.0.0, 17.1.0, 17.2.0, 17.3.0, 17.4.0, 17.5.0, 17.6.0, 17.7.0, 18.0.0, 18.1.0, 18.2.0, 18.3.0, 18.4.0, 18.5.0, 18.6.0, 18.7.0, 19.0.0, 19.1.0, 19.2.0

Historically, the first argument used with .**Dt** was **BSD** or **ATT**. An unrecognized version argument after **ATT** is replaced with "Unix"; for other predefined abbreviations, it is ignored and a warning diagnostic emitted. Otherwise, unrecognized arguments are displayed verbatim in the page footer. For instance, this page uses "**.Os groff 1.24.2**" whereas a locally produced page might employ "**.Os "UXYZ CS Department"**", omitting versioning.

This macro is neither callable nor parsed.

Introduction to manual and general text domains

What's in a Name . . .

The manual domain macro names are derived from the day to day informal language used to describe commands, subroutines and related files. Slightly different variations of this language are used to describe the three different aspects of writing a man page. First, there is the description of *mdoc* macro command usage. Second is the description of a Unix command *with mdoc* macros, and third, the description of a command to a user in the verbal sense; that is, discussion of a command in the text of a man page.

In the first case, *troff* macros are themselves a type of command; the general syntax for a *troff* command is:

.Xx argument1 argument2 . . .

`.Xx` is a macro command, and anything following it are arguments to be processed. In the second case, the description of a Unix command using the manual domain macros is a bit more involved; a typical "Synopsis" command line might be displayed as:

filter [**–flag**] ⟨*infile*⟩ ⟨*outfile*⟩

Here, **filter** is the command name and the bracketed string **–flag** is a *flag* argument designated as optional by the option brackets. In *mdoc* terms, ⟨*infile*⟩ and ⟨*outfile*⟩ are called *meta arguments*; in this example, the user has to replace the meta expressions given in angle brackets with real file names. Note that in this document meta arguments are used to describe *mdoc* commands; in most man pages, meta variables are not specifically written with angle brackets. The macros that formatted the above example:

```
.Nm filter
.Op Fl flag
.Ao Ar infile Ac Ao Ar outfile Ac
```

In the third case, discussion of commands and command syntax includes both examples above, but may add more detail. The arguments ⟨*infile*⟩ and ⟨*outfile*⟩ from the example above might be referred to as *operands* or *file arguments*. Some command-line argument lists are quite long:

make [**–eiknqrstv**] [**–D** *variable*] [**–d** *flags*] [**–f** *makefile*] [**–I** *directory*] [**–j** *max_jobs*] [*variable*=*value*] [*target* . . .]

Here one might talk about the command *make* and qualify the argument, *makefile*, as an argument to the flag, **–f**, or discuss the optional file operand *target*. In the verbal context, such detail can prevent confusion, however the *mdoc* package does not have a macro for an argument *to* a flag. Instead the `Ar` argument macro is used for an operand or file argument like *target* as well as an argument to a flag like *variable*. The make command line was produced from:

```
.Nm make
.Op Fl eiknqrstv
.Op Fl D Ar variable
.Op Fl d Ar flags
.Op Fl f Ar makefile
.Op Fl I Ar directory
.Op Fl j Ar max_jobs
.Op Ar variable Ns = Ns Ar value
```

```
.Bk
.Op Ar target ...
.Ek
```

The `.Bk` and `.Ek` macros are explained in "Keeps".

General Syntax

The manual domain and general text domain macros share a similar syntax with a few minor deviations; most notably, `.Ar`, `.Fl`, `.Nm`, and `.Pa` differ only when called without arguments; and `.Fn` and `.Xr` impose an order on their argument lists. All manual domain macros are capable of recognizing and properly handling punctuation, provided each punctuation character is separated by a leading space. If a command is given:

```
.Ar sptr, ptr),
```

The result is:

sptr, ptr),

The punctuation is not recognized and all is output in the font used by `.Ar`. If the punctuation is separated by a leading whitespace:

```
.Ar sptr , ptr ) ,
```

The result is:

sptr , ptr) ,

The punctuation is now recognized and output in the default font distinguishing it from the argument strings. To remove the special meaning from a punctuation character, escape it with `\&`.

The following punctuation characters are recognized by *mdoc*:

. , : ; ( ) [ ] ? !

troff is limited as a macro language, and has difficulty when presented with a string containing certain mathematical, logical, or quotation character sequences:

```
{+,-,/,*,%,<,>,<=,>=,=,==,&,`,',"}
```

The problem is that *troff* may assume it is supposed to actually perform the operation or evaluation suggested by the characters. To prevent the accidental evaluation of these characters, escape them with `\&`. Typical syntax is shown in the first manual domain macro displayed below, `.Ad`.

Manual domain

Addresses

The address macro identifies an address construct.

`Usage: .Ad ⟨address⟩ . . .`

.Ad addr1	*addr1*
.Ad addr1 .	*addr1*.
.Ad addr1 , file2	*addr1*, *file2*
.Ad f1 , f2 , f3 :	*f1*, *f2*, *f3*:
.Ad addr)) ,	*addr*)),

The default width is 12n.

Author Name

The `.An` macro is used to specify the name of the author of the item being documented, or the name of the author of the actual manual page.

`Usage: .An ⟨author name⟩ . . .`

.An "Joe Author"	Joe Author
.An "Joe Author" ,	Joe Author,
.An "Joe Author" Aq nobody@FreeBSD.org	Joe Author <nobody@FreeBSD.org>
.An "Joe Author")) ,	Joe Author)),

The default width is 12n.

In a section titled "Authors", `An` causes a break, allowing each new name to appear on its own line. If this is not desirable,

```
.An -nosplit
```

call will turn this off. To turn splitting back on, write

```
.An -split
```

Arguments

The **.Ar** argument macro may be used whenever an argument is referenced. If called without arguments, '*file* . . .' is output.

`Usage: .Ar [⟨argument⟩] . . .`

.Ar	*file* . . .
.Ar file1	*file1*
.Ar file1 .	*file1*.
.Ar file1 file2	*file1 file2*
.Ar file ... dest-dir	*file* . . . *dest-dir*
.Ar f1 f2 f3 :	*f1 f2 f3*:
.Ar file)) ,	*file*)),

The default width is 12n.

Configuration Declaration (Section Four Only)

The `.Cd` macro is used to demonstrate a *config*(8) declaration for a device interface in a section four manual.

`Usage: .Cd ⟨argument⟩ . . .`

.Cd "device le0 at scode?"	**device le0 at scode?**

In a section titled "Synopsis", `Cd` causes a break before and after its arguments.

The default width is 12n.

Command Modifiers

The command modifier is identical to the `.Fl` (flag) command with the exception that the `.Cm` macro does not assert a dash in front of every argument. Traditionally flags are marked by the preceding dash, however, some commands or subsets of commands do not use them. Command modifiers may also be specified in conjunction with interactive commands such as editor commands. See "Flags".

The default width is 10n.

Defined Variables

A variable (or constant) that is defined in an include file is specified by the macro `.Dv`.

`Usage: .Dv ⟨defined-variable⟩ . . .`

.Dv MAXHOSTNAMELEN	MAXHOSTNAMELEN

.Dv TIOCGPGRP) TIOCGPGRP)

The default width is 12n.

Errnos

The `.Er` errno macro specifies the error return value for section 2, 3, and 9 library routines. The second example below shows `.Er` used with the `.Bq` general text domain macro, as it would be used in a section two manual page.

`Usage: .Er ⟨errno type⟩ . . .`

.Er ENOENT ENOENT
.Er ENOENT) ; ENOENT);
.Bq Er ENOTDIR [ENOTDIR]

The default width is 17n.

Environment Variables

The `.Ev` macro specifies an environment variable.

`Usage: .Ev ⟨argument⟩ . . .`

.Ev DISPLAY *DISPLAY*
.Ev PATH . *PATH*.
.Ev PRINTER)) , *PRINTER*)),

The default width is 15n.

Flags

The `.Fl` macro handles command-line flags. It prepends a dash, −, to the flag. For interactive command flags that are not prepended with a dash, the `.Cm` (command modifier) macro is identical, but without the dash.

`Usage: .Fl ⟨argument⟩ . . .`

.Fl **−**
.Fl cfv **−cfv**
.Fl cfv . **−cfv**.
.Cm cfv . **cfv**.
.Fl s v t **−s −v −t**
.Fl − , **−−**,
.Fl xyz) , **−xyz**),
.Fl | **− |**

The `.Fl` macro without any arguments results in a dash representing stdin/stdout. Note that giving `.Fl` a single dash will result in two dashes.

The default width is 12n.

Function Declarations

The `.Fd` macro is used in the “Synopsis” section with section two or three functions. It is neither callable nor parsed.

`Usage: .Fd ⟨argument⟩ . . .`

.Fd "#include <sys/types.h>" **#include <sys/types.h>**

In a section titled “Synopsis”, `Fd` causes a break if a function has already been presented and a break has not occurred, leaving vertical space between one function declaration and the next.

In a section titled “Synopsis”, the `In` macro represents the **#include** statement, and is the short form of the above example. It specifies the C header file as being included in a C program. It also causes a break.

While not in the "Synopsis" section, it represents the header file enclosed in angle brackets.

```
Usage: .In ⟨header file⟩
```

.In stdio.h <*stdio.h*>
.In stdio.h <*stdio.h*>

Function Types

This macro is intended for the "Synopsis" section. It may be used anywhere else in the man page without problems, but its main purpose is to present the function type (in BSD kernel normal form) for the "Synopsis" of sections two and three. (It causes a break, allowing the function name to appear on the next line.)

```
Usage: .Ft ⟨type⟩ . . .
```

.Ft struct stat *struct stat*

Functions (Library Routines)

The `.Fn` macro is modeled on ANSI C conventions.

```
Usage: .Fn ⟨function⟩ [⟨parameter⟩] . . .
```

.Fn getchar **getchar**()
.Fn strlen) , **strlen**()),
.Fn align "char *ptr" , **align**(*char *ptr*),

Note that any call to another macro signals the end of the `.Fn` call (it will insert a closing parenthesis at that point).

For functions with many parameters (which is rare), the macros `.Fo` (function open) and `.Fc` (function close) may be used with `.Fa` (function argument).

Example:

```
.Ft int
.Fo res_mkquery
.Fa "int op"
.Fa "char *dname"
.Fa "int class"
.Fa "int type"
.Fa "char *data"
.Fa "int datalen"
.Fa "struct rrec *newrr"
.Fa "char *buf"
.Fa "int buflen"
.Fc
```

Produces:

int **res_mkquery**(*int op* , *char *dname* , *int class* , *int type* , *char *data* , *int datalen* , *struct rrec *newrr* , *char *buf* , *int buflen*)

Typically, in a "Synopsis" section, the function declaration will begin the line. If more than one function is presented in the "Synopsis" section and a function type has not been given, a break will occur, leaving vertical space between the current and prior function names.

The default width values of `.Fn` and `.Fo` are 12n and 16n, respectively.

Function Arguments

The `.Fa` macro is used to refer to function arguments (parameters) outside of the "Synopsis" section of the manual or inside the "Synopsis" section if the enclosure macros `.Fo` and `.Fc` instead of `.Fn` are used. `.Fa` may also be used to refer to structure members.

```
Usage: .Fa ⟨function argument⟩ . . .
```

.Fa d_namlen)) ,	*d_namlen*)),
.Fa iov_len	*iov_len*

The default width is 12n.

Return Values

The `.Rv` macro generates text for use in the “Return values” section.

```
Usage: .Rv [-std] [⟨function⟩ . . .]
```

For example, `.Rv -std atexit` produces:

> The **atexit**() function returns the value 0 if successful; otherwise the value −1 is returned and the global variable *errno* is set to indicate the error.

The **−std** option is valid only for manual page sections 2 and 3. Currently, this macro does nothing if used without the **−std** flag.

Exit Status

The `.Ex` macro generates text for use in the “Diagnostics” section.

```
Usage: .Ex [-std] [⟨utility⟩ . . .]
```

For example, `.Ex -std cat` produces:

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

The **−std** option is valid only for manual page sections 1, 6 and 8. Currently, this macro does nothing if used without the **−std** flag.

Interactive Commands

The `.Ic` macro designates an interactive or internal command.

```
Usage: .Ic ⟨argument⟩ . . .
```

.Ic :wq	**:wq**
.Ic "do while {...}"	**do while {...}**
.Ic setenv , unsetenv	**setenv**, **unsetenv**

The default width is 12n.

Library Names

The `.Lb` macro is used to specify the library where a particular function is compiled in.

```
Usage: .Lb ⟨argument⟩ . . .
```

Available arguments to `.Lb` and their results are:

libarchive	Reading and Writing Streaming Archives Library (libarchive, −larchive)
libarm	ARM Architecture Library (libarm, −larm)
libarm32	ARM32 Architecture Library (libarm32, −larm32)
libbluetooth	Bluetooth Library (libbluetooth, −lbluetooth)
libbsm	Basic Security Module Library (libbsm, −lbsm)
libc	Standard C Library (libc, −lc)
libc_r	Reentrant C Library (libc_r, −lc_r)
libcalendar	Calendar Arithmetic Library (libcalendar, −lcalendar)
libcam	Common Access Method User Library (libcam, −lcam)
libcdk	Curses Development Kit Library (libcdk, −lcdk)
libcipher	FreeSec Crypt Library (libcipher, −lcipher)
libcompat	Compatibility Library (libcompat, −lcompat)

libcrypt Crypt Library (libcrypt, −lcrypt)
libcurses Curses Library (libcurses, −lcurses)
libdevinfo Device and Resource Information Utility Library (libdevinfo, −ldevinfo)
libdevstat Device Statistics Library (libdevstat, −ldevstat)
libdisk Interface to Slice and Partition Labels Library (libdisk, −ldisk)
libdwarf DWARF Access Library (libdwarf, −ldwarf)
libedit Command Line Editor Library (libedit, −ledit)
libelf ELF Access Library (libelf, −lelf)
libevent Event Notification Library (libevent, −levent)
libfetch File Transfer Library for URLs (libfetch, −lfetch)
libform Curses Form Library (libform, −lform)
libgeom Userland API Library for kernel GEOM subsystem (libgeom, −lgeom)
libgpib General-Purpose Instrument Bus (GPIB) library (libgpib, −lgpib)
libi386 i386 Architecture Library (libi386, −li386)
libintl Internationalized Message Handling Library (libintl, −lintl)
libipsec IPsec Policy Control Library (libipsec, −lipsec)
libipx IPX Address Conversion Support Library (libipx, −lipx)
libiscsi iSCSI protocol library (libiscsi, −liscsi)
libjail Jail Library (libjail, −ljail)
libkiconv Kernel side iconv library (libkiconv, −lkiconv)
libkse N:M Threading Library (libkse, −lkse)
libkvm Kernel Data Access Library (libkvm, −lkvm)
libm Math Library (libm, −lm)
libm68k m68k Architecture Library (libm68k, −lm68k)
libmagic Magic Number Recognition Library (libmagic, −lmagic)
libmd Message Digest (MD4, MD5, etc.) Support Library (libmd, −lmd)
libmemstat Kernel Memory Allocator Statistics Library (libmemstat, −lmemstat)
libmenu Curses Menu Library (libmenu, −lmenu)
libnetgraph Netgraph User Library (libnetgraph, −lnetgraph)
libnetpgp Netpgp signing, verification, encryption and decryption (libnetpgp, −lnetpgp)
libossaudio OSS Audio Emulation Library (libossaudio, −lossaudio)
libpam Pluggable Authentication Module Library (libpam, −lpam)
libpcap Packet Capture Library (libpcap, −lpcap)
libpci PCI Bus Access Library (libpci, −lpci)
libpmc Performance Counters Library (libpmc, −lpmc)
libposix POSIX Compatibility Library (libposix, −lposix)
libprop Property Container Object Library (libprop, −lprop)
libpthread POSIX Threads Library (libpthread, −lpthread)
libpuffs puffs Convenience Library (libpuffs, −lpuffs)
librefuse File System in Userspace Convenience Library (librefuse, −lrefuse)
libresolv DNS Resolver Library (libresolv, −lresolv)
librpcsec_gss RPC GSS-API Authentication Library (librpcsec_gss, −lrpcsec_gss)
librpcsvc RPC Service Library (librpcsvc, −lrpcsvc)
librt POSIX Real-time Library (librt, −lrt)
libsdp Bluetooth Service Discovery Protocol User Library (libsdp, −lsdp)
libssp Buffer Overflow Protection Library (libssp, −lssp)
libSystem System Library (libSystem, −lSystem)
libtermcap Termcap Access Library (libtermcap, −ltermcap)
libterminfo Terminal Information Library (libterminfo, −lterminfo)
libthr 1:1 Threading Library (libthr, −lthr)

libufs	UFS File System Access Library (libufs, −lufs)
libugidfw	File System Firewall Interface Library (libugidfw, −lugidfw)
libulog	User Login Record Library (libulog, −lulog)
libusbhid	USB Human Interface Devices Library (libusbhid, −lusbhid)
libutil	System Utilities Library (libutil, −lutil)
libvgl	Video Graphics Library (libvgl, −lvgl)
libx86_64	x86_64 Architecture Library (libx86_64, −lx86_64)
libz	Compression Library (libz, −lz)

Site-specific additions might be found in the file *mdoc.local*; see section “Files” below.

In a section titled “Library”, `Lb` causes a break before and after its arguments.

Literals

The `Li` literal macro may be used for special characters, symbolic constants, and other syntactical items that should be typed exactly as displayed.

```
Usage: .Li ⟨argument⟩ ...
```

.Li \en	**\n**
.Li M1 M2 M3 ;	**M1 M2 M3**;
.Li cntrl−D) ,	**cntrl-D**),
.Li 1024 ...	**1024** ...

The default width is 16n.

Names

The `Nm` macro identifies the document’s central topic. Upon its first call, it has the peculiarity of remembering its argument. When subsequently called without arguments, `Nm` regurgitates this initial name for the sole purpose of making less work for the document’s author. Use of `Nm` is also appropriate when presenting a command synopsis for the topic of a man page in section 1, 6, or 8. Its behavior changes when presented with arguments of various forms.

.Nm groff_mdoc	**groff_mdoc**
.Nm	**groff_mdoc**
.Nm \−mdoc	**−mdoc**
.Nm foo)) ,	**foo**)),
.Nm :	**groff_mdoc**:

By default, the topic is set in boldface to reflect its prime importance in the discussion. Cross references to other man page topics should use `Xr`; including a second argument for the section number enables them to be hyperlinked. By default, cross-referenced topics are set in italics to avoid cluttering the page with boldface.

The default width is 10n.

Options

The `.Op` macro places option brackets around any remaining arguments on the command line, and places any trailing punctuation outside the brackets. The macros `.Oo` and `.Oc` (which produce an opening and a closing option bracket, respectively) may be used across one or more lines or to specify the exact position of the closing parenthesis.

```
Usage: .Op [⟨option⟩] ...
```

.Op	[]
.Op Fl k	[**−k**]
.Op Fl k) .	[**−k**]).
.Op Fl k Ar kookfile	[**−k** *kookfile*]

.Op Fl k Ar kookfile ,	[**−k** *kookfile*],
.Op Ar objfil Op Ar corfil	[*objfil* [*corfil*]]
.Op Fl c Ar objfil Op Ar corfil ,	[**−c** *objfil* [*corfil*]],
.Op word1 word2	[word1 word2]
.Li .Op Oo Ao option Ac Oc ...	**.Op** [⟨option⟩] ...

Here a typical example of the `.Oo` and `.Oc` macros:

```
.Oo
.Op Fl k Ar kilobytes
.Op Fl i Ar interval
.Op Fl c Ar count
.Oc
```

Produces:

[[**−k** *kilobytes*] [**−i** *interval*] [**−c** *count*]]

The default width values of `.Op` and `.Oo` are 14n and 10n, respectively.

Pathnames

The `.Pa` macro formats file specifications. If called without arguments, '~' (recognized by many shells) is output, representing the user's home directory.

```
Usage: .Pa [⟨pathname⟩] ...
```

.Pa	~
.Pa /usr/share	*/usr/share*
.Pa /tmp/fooXXXXX) .	*/tmp/fooXXXXX*).

The default width is 32n.

Standards

The `.St` macro replaces standard abbreviations with their formal names.

```
Usage: .St ⟨abbreviation⟩ ...
```

Available pairs for "Abbreviation/Formal Name" are:

ANSI/ISO C

−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")
−isoC−99	ISO/IEC 9899:1999 ("ISO C99")
−isoC−2011	ISO/IEC 9899:2011 ("ISO C11")
−isoC−2023	ISO/IEC 9899:2024 ("ISO C23")

POSIX Part 1: System API

−iso9945−1−90	ISO/IEC 9945-1:1990 ("POSIX.1")
−iso9945−1−96	ISO/IEC 9945-1:1996 ("POSIX.1")
−p1003.1	IEEE Std 1003.1 ("POSIX.1")
−p1003.1−88	IEEE Std 1003.1-1988 ("POSIX.1")
−p1003.1−90	ISO/IEC 9945-1:1990 ("POSIX.1")
−p1003.1−96	ISO/IEC 9945-1:1996 ("POSIX.1")
−p1003.1b−93	IEEE Std 1003.1b-1993 ("POSIX.1")
−p1003.1c−95	IEEE Std 1003.1c-1995 ("POSIX.1")

–p1003.1g–2000	IEEE Std 1003.1g-2000 (“POSIX.1”)
–p1003.1i–95	IEEE Std 1003.1i-1995 (“POSIX.1”)
–p1003.1–2001	IEEE Std 1003.1-2001 (“POSIX.1”)
–p1003.1–2004	IEEE Std 1003.1-2004 (“POSIX.1”)
–p1003.1–2008	IEEE Std 1003.1-2008 (“POSIX.1”)
–p1003.1–2024	IEEE Std 1003.1-2024 (“POSIX.1”)

POSIX Part 2: Shell and Utilities

–iso9945–2–93	ISO/IEC 9945-2:1993 (“POSIX.2”)
–p1003.2	IEEE Std 1003.2 (“POSIX.2”)
–p1003.2–92	IEEE Std 1003.2-1992 (“POSIX.2”)
–p1003.2a–92	IEEE Std 1003.2a-1992 (“POSIX.2”)

X/Open

–susv1	Version 1 of the Single UNIX Specification (“SUSv1”)
–susv2	Version 2 of the Single UNIX Specification (“SUSv2”)
–susv3	Version 3 of the Single UNIX Specification (“SUSv3”)
–susv4	Version 4 of the Single UNIX Specification (“SUSv4”)
–svid4	System V Interface Definition, Fourth Edition (“SVID4”)
–xbd5	X/Open Base Definitions Issue 5 (“XBD5”)
–xcu5	X/Open Commands and Utilities Issue 5 (“XCU5”)
–xcurses4.2	X/Open Curses Issue 4, Version 2 (“XCURSES4.2”)
–xns5	X/Open Networking Services Issue 5 (“XNS5”)
–xns5.2	X/Open Networking Services Issue 5.2 (“XNS5.2”)
–xpg3	X/Open Portability Guide Issue 3 (“XPG3”)
–xpg4	X/Open Portability Guide Issue 4 (“XPG4”)
–xpg4.2	X/Open Portability Guide Issue 4, Version 2 (“XPG4.2”)
–xsh5	X/Open System Interfaces and Headers Issue 5 (“XSH5”)

Miscellaneous

–ieee754	IEEE Std 754-1985
–iso8601	ISO 8601
–iso8802–3	ISO/IEC 8802-3:1989

Variable Types

The `.Vt` macro may be used whenever a type is referenced. In a section titled “Synopsis”, `Vt` causes a break (useful for old-style C variable declarations).

`Usage: .Vt ⟨type⟩ . . .`

.Vt extern char *optarg ;	*extern char *optarg*;
.Vt FILE *	*FILE **

Variables

Generic variable reference.

`Usage: .Va ⟨variable⟩ . . .`

.Va count	*count*
.Va settimer ,	*settimer*,
.Va "int *prt") :	*int *prt*):
.Va "char s"])) ,	*char s*])),

The default width is 12n.

Manual Page Cross References

The `.Xr` macro expects the first argument to be a manual page name. The optional second argument, if a string (defining the manual section), is put into parentheses.

```
Usage: .Xr ⟨man page name⟩ [⟨section⟩] ...
```

.Xr mdoc *mdoc*
.Xr mdoc , *mdoc*,
.Xr mdoc 7 *mdoc*(7)
.Xr xinit 1x ; *xinit*(1x);

The default width is 10n.

General text domain

AT&T Macro

```
Usage: .At [⟨version⟩] ...
```

.At AT&T UNIX
.At v6 . Version 6 AT&T UNIX.

The following values for ⟨version⟩ are possible:

```
32v, v1, v2, v3, v4, v5, v6, v7, III, V, V.1, V.2, V.3, V.4
```

BSD Macro

```
Usage: .Bx {-alpha | -beta | -devel} ...
       .Bx [⟨version⟩ [⟨release⟩]] ...
```

.Bx BSD
.Bx 4.3 . 4.3BSD.
.Bx –devel
BSD (currently under development)

⟨version⟩ will be prepended to the string 'BSD'. The following values for ⟨release⟩ are possible:

```
Reno, reno, Tahoe, tahoe, Lite, lite, Lite2, lite2
```

NetBSD Macro

```
Usage: .Nx [⟨version⟩] ...
```

.Nx NetBSD
.Nx 1.4 . NetBSD 1.4.

For possible values of ⟨version⟩ see the description of the `.Os` command above in section "Title macros".

FreeBSD Macro

```
Usage: .Fx [⟨version⟩] ...
```

.Fx FreeBSD
.Fx 2.2 . FreeBSD 2.2.

For possible values of ⟨version⟩ see the description of the `.Os` command above in section "Title macros".

DragonFly Macro

```
Usage: .Dx [⟨version⟩] ...
```

.Dx DragonFly
.Dx 1.4 . DragonFly 1.4.

For possible values of ⟨version⟩ see the description of the `.Os` command above in section "Title macros".

OpenBSD Macro

```
Usage: .Ox [⟨version⟩] ...
```

.Ox 1.0 OpenBSD 1.0

BSD/OS Macro

Usage: `.Bsx [⟨version⟩] . . .`

.Bsx 1.0 BSD/OS 1.0

Unix Macro

Usage: `.Ux . . .`

.Ux Unix

Emphasis Macro

Text may be stressed or emphasized with the `.Em` macro. The usual font for emphasis is italic.

Usage: `.Em ⟨argument⟩ . . .`

.Em does not	*does not*
.Em exceed 1024 .	*exceed 1024*.
.Em vide infra)) ,	*vide infra*)),

The default width is 10n.

Font Mode

The `.Bf` font mode must be ended with the `.Ef` macro (the latter takes no arguments). Font modes may be nested within other font modes.

`.Bf` has the following syntax:

```
.Bf ⟨font mode⟩
```

⟨font mode⟩ must be one of the following three types:

Em | −emphasis	Same as if the `.Em` macro was used for the entire block of text.
Li | −literal	Same as if the `.Li` macro was used for the entire block of text.
Sy | −symbolic	Same as if the `.Sy` macro was used for the entire block of text.

Both macros are neither callable nor parsed.

Enclosure and Quoting Macros

The concept of enclosure is similar to quoting. The object being to enclose one or more strings between a pair of characters like quotes or parentheses. The terms quoting and enclosure are used interchangeably throughout this document. Most of the one-line enclosure macros end in small letter `q` to give a hint of quoting, but there are a few irregularities. For each enclosure macro, there is a pair of opening and closing macros that end with the lowercase letters `o` and `c` respectively.

Quote	Open	Close	Function	Result
.Aq	.Ao	.Ac	Angle Bracket Enclosure	<string>
.Bq	.Bo	.Bc	Bracket Enclosure	[string]
.Brq	.Bro	.Brc	Brace Enclosure	{string}
.Dq	.Do	.Dc	Double Quote	“string”
.Eq	.Eo	.Ec	Enclose String (in XY)	XstringY
.Pq	.Po	.Pc	Parenthesis Enclosure	(string)
.Ql			Quoted or Literal	“string” or `string`
.Qq	.Qo	.Qc	Straight Double Quote	"string"
.Sq	.So	.Sc	Single Quote	‘string’

All macros ending with ‘q’ and ‘o’ have a default width value of 12n.

.Eo, **.Ec** These macros expect the first argument to be the opening and closing strings, respectively.

.Es, **.En** To work around the nine-argument limit in the original *troff* program, *mdoc* supports two other macros that are now obsolete. `.Es` uses its first and second parameters as opening and closing marks which are then used to enclose the arguments of `.En`. The default width value is 12n for both macros.

.Eq The first and second arguments of this macro are the opening and closing strings respectively, followed by the arguments to be enclosed.

.Ql The quoted-or-literal macro behaves differently in *troff* and *nroff* modes. When formatting with *nroff*(1), *mdoc* quotes the arguments. With *troff*, *mdoc* sets them in a constant-width font.

The default width is 16n.

.Pf The prefix macro suppresses the whitespace between its first and second argument:

.Pf (Fa name2 (*name2*

The default width is 12n.

The `.Ns` macro (see below) performs the analogous suffix function.

.Ap The `.Ap` macro inserts an apostrophe and exits any special text modes, continuing in `.No` mode.

Examples of quoting:

.Aq	⟨⟩
.Aq Pa ctype.h) ,	⟨*ctype.h*⟩),
.Bq	[]
.Bq Em Greek , French .	[*Greek*, *French*].
.Dq	“”
.Dq string abc .	“string abc”.
.Dq '\[ha][A-Z]'	“'^[A-Z]'”
.Ql man mdoc	`man mdoc`
.Qq	""
.Qq string) ,	"string"),
.Qq string Ns),	"string),"
.Sq	‘’
.Sq string	‘string’
.Em or Ap ing	*or*’ing

For a good example of nested enclosure macros, see the `.Op` option macro. It was created from the same underlying enclosure macros as those presented in the list above. The `.Xo` and `.Xc` extended argument list macros are discussed below.

Normal text macro

`No` formats subsequent argument(s) normally, ending the effect of `Em` and similar. Parsing is *not* suppressed, so you must prefix words like `No` with `\&` to avoid their interpretation as *mdoc* macros.

```
Usage: .No argument ...
```

.Em Use caution No here .	→ *Use caution* here.
.Em No dogs allowed .	→ *No* dogs allowed.
.Em \&No dogs allowed .	→ *No dogs allowed*.

The default width is 12n.

No-Space Macro

The `.Ns` macro suppresses insertion of a space between the current position and its first parameter. For example, it is useful for old style argument lists where there is no space between the flag and argument:

```
Usage: ... ⟨argument⟩ Ns [⟨argument⟩] ...
       .Ns ⟨argument⟩ ...
```

.Op Fl I Ns Ar directory [**–I***directory*]

Note: The `.Ns` macro always invokes the `.No` macro after eliminating the space unless another macro name follows it. If used as a command (i.e., the second form above in the 'Usage' line), `.Ns` is identical to `.No`.

(Sub)section cross references

Use the `.Sx` macro to cite a (sub)section heading within the given document.

```
Usage: .Sx ⟨section-reference⟩ ...
```

.Sx Files → "Files"

The default width is 16n.

Symbolics

The symbolic emphasis macro is generally a boldface macro in either the symbolic sense or the traditional English usage.

```
Usage: .Sy ⟨symbol⟩ ...
```

.Sy Important Notice → **Important Notice**

The default width is 6n.

Mathematical Symbols

Use this macro for mathematical symbols and similar things.

```
Usage: .Ms ⟨math symbol⟩ ...
```

.Ms sigma → **sigma**

The default width is 6n.

References and Citations

The following macros make a modest attempt to handle references. At best, the macros make it convenient to manually drop in a subset of *refer*(1) style references.

.Rs	Reference start (does not take arguments). In a section titled "See also", it causes a break and begins collection of reference information until the reference end macro is read.
.Re	Reference end (does not take arguments). The reference is printed.
.%A	Reference author name; one name per invocation.
.%B	Book title.
.%C	City/place.
.%D	Date.
.%I	Issuer/publisher name.
.%J	Journal name.
.%N	Issue number.
.%O	Optional information.
.%P	Page number.
.%Q	Corporate or foreign author.
.%R	Report name.
.%T	Title of article.
.%U	Optional hypertext reference.
.%V	Volume.

Macros beginning with `%` are not callable but accept multiple arguments in the usual way. Only the `.Tn` macro is handled properly as a parameter; other macros will cause strange output. `.%B` and `.%T` can be used outside of the `.Rs`/`.Re` environment.

Example:

```
.Rs
.%A "Matthew Bar"
.%A "John Foo"
```

```
.%T "Implementation Notes on foobar(1)"
.%R "Technical Report ABC-DE-12-345"
.%Q "Drofnats College"
.%C "Nowhere"
.%D "April 1991"
.Re
```

produces

Matthew Bar and John Foo, *Implementation Notes on foobar(1)*, Technical Report ABC-DE-12-345, Drofnats College, Nowhere, April 1991.

Trade Names or Acronyms

The trade name macro prints its arguments at a smaller type size. It is intended to imitate a small caps fonts for fully capitalized acronyms.

`Usage: .Tn ⟨symbol⟩ ...`

.Tn DEC DEC
.Tn ASCII ASCII

The default width is 10n.

Extended Arguments

The **.Xo** and **.Xc** macros allow one to extend an argument list on a macro boundary for the `.It` macro (see below). Note that **.Xo** and **.Xc** are implemented similarly to all other macros opening and closing an enclosure (without inserting characters, of course). This means that the following is true for those macros also.

Here is an example of `.Xo` using the space mode macro to turn spacing off:

```
.Bd -literal -offset indent
.Sm off
.It Xo Sy I Ar operation
.No \en Ar count No \en
.Xc
.Sm on
.Ed
```

produces

I*operation*\n*count*\n

Another one:

```
.Bd -literal -offset indent
.Sm off
.It Cm S No / Ar old_pattern Xo
.No / Ar new_pattern
.No / Op Cm g
.Xc
.Sm on
.Ed
```

produces

S/*old_pattern*/*new_pattern*/[**g**]

Another example of `.Xo` and enclosure macros: Test the value of a variable.

```
.Bd -literal -offset indent
.It Xo
.Ic .ifndef
.Oo \&! Oc Ns Ar variable Oo
```

```
.Ar operator variable No ...
.Oc Xc
.Ed
```

produces

.ifndef [!]*variable* [*operator variable* ...]

Page structure domain

Section headings

The following `.Sh` section heading macros are required in every man page. The remaining section headings are recommended at the discretion of the author writing the manual page. The `.Sh` macro is parsed but not generally callable. It can be used as an argument in a call to `.Sh` only; it then reactivates the default font for `.Sh`.

The default width is 8n.

.Sh Name The `.Sh Name` macro is mandatory. If not specified, headers, footers, and page layout defaults will not be set and things will be rather unpleasant. The *Name* section consists of at least three items. The first is the `.Nm` name macro naming the subject of the man page. The second is the name description macro, `.Nd`, which separates the subject name from the third item, which is the description. The description should be the most terse and lucid possible, as the space available is small.

`.Nd` first prints −, then all its arguments.

.Sh Library This section is for section two and three function calls. It should consist of a single `.Lb` macro call; see "Library Names".

.Sh Synopsis The "Synopsis" section describes the typical usage of the subject of a man page. The macros required are either `.Nm`, `.Cd`, or `.Fn` (and possibly `.Fo`, `.Fc`, `.Fd`, and `.Ft`). The function name macro `.Fn` is required for manual page sections 2 and 3; the command and general name macro `.Nm` is required for sections 1, 5, 6, 7, and 8. Section 4 manuals require a `.Nm`, `.Fd` or a `.Cd` configuration device usage macro. Several other macros may be necessary to produce the synopsis line as shown below:

cat [**−benstuv**] [**−**] *file* ...

The following macros were used:

```
.Nm cat
.Op Fl benstuv
.Op Fl
.Ar file No ...
```

.Sh Description In most cases the first text in the "Description" section is a brief paragraph on the command, function or file, followed by a lexical list of options and respective explanations. To create such a list, the `.Bl` (begin list), `.It` (list item) and `.El` (end list) macros are used (see "Lists and Columns" below).

.Sh Implementation notes
Implementation specific information should be placed here.

.Sh Return values Sections 2, 3 and 9 function return values should go here. The `.Rv` macro may be used to generate text for use in the "Return values" section for most section 2 and 3 library functions; see "Return Values".

The following `.Sh` section headings are part of the preferred manual page layout and must be used appropriately to maintain consistency. They are listed in the order in which they would be used.

.Sh Environment The *Environment* section should reveal any related environment variables and clues to their behavior and/or usage.

.Sh Files Files which are used or created by the man page subject should be listed via the `.Pa` macro in the "Files" section.

.Sh Examples There are several ways to create examples. See subsection "Examples and Displays" below for details.

.Sh Diagnostics Diagnostic messages from a command should be placed in this section. The `.Ex` macro may be used to generate text for use in the "Diagnostics" section for most section 1, 6 and 8 commands; see "Exit Status".

.Sh Compatibility Known compatibility issues (e.g. deprecated options or parameters) should be listed here.

.Sh Errors Specific error handling, especially from library functions (man page sections 2, 3, and 9) should go here. The `.Er` macro is used to specify an error (errno).

.Sh See also References to other material on the man page topic and cross references to other relevant man pages should be placed in the "See also" section. Cross references are specified using the `.Xr` macro. Currently *refer*(1) style references are not accommodated.

It is recommended that the cross references be sorted by section number, then alphabetically by name within each section, then separated by commas. Example:

ls(1), *ps*(1), *group*(5), *passwd*(5)

.Sh Standards If the command, library function, or file adheres to a specific implementation such as IEEE Std 1003.2 ("POSIX.2") or ANSI X3.159-1989 ("ANSI C89") this should be noted here. If the command does not adhere to any standard, its history should be noted in the *History* section.

.Sh History Any command which does not adhere to any specific standards should be outlined historically in this section.

.Sh Authors Credits should be placed here. Use the `.An` macro for names and the `.Aq` macro for email addresses within optional contact information. Explicitly indicate whether the person authored the initial manual page or the software or whatever the person is being credited for.

.Sh Bugs Blatant problems with the topic go here.

User-specified `.Sh` sections may be added; for example, this section was set with:

```
.Sh "Page structure domain"
```

Subsection headings

Subsection headings have exactly the same syntax as section headings: `.Ss` is parsed but not generally callable. It can be used as an argument in a call to `.Ss` only; it then reactivates the default font for `.Ss`.

The default width is 8n.

Paragraphs and Line Spacing

.Pp The `.Pp` paragraph command may be used to specify a line space where necessary. The macro is not necessary after a `.Sh` or `.Ss` macro or before a `.Bl` or `.Bd` macro (which both assert a vertical distance unless the **−compact** flag is given).

The macro is neither callable nor parsed and takes no arguments; an alternative name is `.Lp`.

Keeps

The only keep that is implemented at this time is for words. The macros are `.Bk` (begin keep) and `.Ek` (end keep). The only option that `.Bk` currently accepts is **−words** (also the default); this prevents breaks in the middle of options. In the example for **make** command-line arguments (see "What's in a Name"), the keep

prevents *nroff* from placing the flag and the argument on separate lines.

Neither macro is callable or parsed.

More work needs to be done on the keep macros; specifically, a **−line** option should be added.

Examples and Displays

There are seven types of displays.

.D1 (This is D-one.) Display one line of indented text. This macro is parsed but not callable.

−ldghfstru

The above was produced by: **.D1 Fl ldghfstru**.

.Dl (This is D-ell.) Display one line of indented *literal* text. The `.Dl` example macro has been used throughout this file. It allows the indentation (display) of one line of text. Its default font is set to constant width (literal). `.Dl` is parsed but not callable.

```
% ls −ldg /usr/local/bin
```

The above was produced by: **.Dl % ls \−ldg /usr/local/bin**.

.Bd Begin display. The `.Bd` display must be ended with the `.Ed` macro. It has the following syntax:

.Bd {−literal | −filled | −unfilled | −ragged | −centered} [−offset ⟨string⟩] [−file ⟨file name⟩] [−compact]

−ragged Fill, but do not adjust (leave the right margin ragged).

−centered Center lines between the current left and right margin. Note that each single line is centered.

−unfilled Do not fill; break lines where their input lines are broken. This can produce overlong lines without warning messages.

−filled Display a filled block. Text is filled and adjusted; the left and right margins are straight.

−literal Display block with literal font (usually fixed-width). Useful for source code or simple tabbed or spaced text.

−file ⟨*file name*⟩ The file whose name follows the **−file** flag is read and displayed before any data enclosed with `.Bd` and `.Ed`, using the selected display type. Any *troff/mdoc* commands in the file will be processed.

−offset ⟨*string*⟩ If **−offset** is specified with one of the following strings, the string is interpreted to indicate the level of indentation for the forthcoming block of text:

left Align block on the current left margin; this is the default mode of `.Bd`.

center Supposedly center the block. At this time unfortunately, the block merely gets left aligned about an imaginary center margin.

indent Indent by one default indent value or tab. The default indent value is also used for the `.D1` and `.Dl` macros, so one is guaranteed the two types of displays will line up. The indentation value is normally set to 6n or about two thirds of an inch (six constant width characters).

indent−two Indent two times the default indent value.

right This *left* aligns the block about two inches from the right side of the page. This macro needs work and perhaps may never do the right thing within *troff*.

If ⟨string⟩ is a valid numeric expression instead (*with a scaling indicator other than* '*u*'), use that value for indentation. The most useful scaling indicators are 'm' and 'n', specifying the so-called *Em* and *En square*. This is approximately the width of the letters 'm' and 'n' respectively of the current font (for *nroff* output, both scaling indicators give the same values). If ⟨string⟩ isn't a numeric expression, it is tested whether it is an *mdoc* macro name, and the default offset value associated with this macro is used.

Finally, if all tests fail, the width of ⟨string⟩ (typeset with a fixed-width font) is taken as the offset.

−compact Suppress insertion of vertical space before begin of display.

.Ed End display (takes no arguments).

Lists and Columns

There are several types of lists which may be initiated with the `.Bl` begin-list macro. Items within the list are specified with the `.It` item macro, and each list must end with the `.El` macro. Lists may be nested within themselves and within displays. The use of columns inside of lists or lists inside of columns is untested.

In addition, several list attributes may be specified such as the width of a tag, the list offset, and compactness (blank lines between items allowed or disallowed). Most of this document has been formatted with a tag style list (**−tag**).

It has the following syntax forms:

.Bl {−hang | −ohang | −tag | −diag | −inset} [−width ⟨string⟩] [−offset ⟨string⟩] [−compact]
.Bl −column [−offset ⟨string⟩] ⟨string1⟩ ⟨string2⟩ ...
.Bl {−item | −enum [−nested] | −bullet | −hyphen | −dash} [−offset ⟨string⟩] [−compact]

And now a detailed description of the list types.

−bullet A bullet list.

```
.Bl -bullet -offset indent -compact
.It
Bullet one goes here.
.It
Bullet two here.
.El
```

Produces:

- Bullet one goes here.
- Bullet two here.

−dash (or **−hyphen**)
A dash list.

```
.Bl -dash -offset indent -compact
.It
Dash one goes here.
.It
Dash two here.
.El
```

Produces:

- Dash one goes here.
- Dash two here.

−enum An enumerated list.

```
.Bl -enum -offset indent -compact
.It
Item one goes here.
.It
And item two here.
.El
```

The result:

1. Item one goes here.
2. And item two here.

If you want to nest enumerated lists, use the **–nested** flag (starting with the second-level list):

```
.Bl -enum -offset indent -compact
.It
Item one goes here
.Bl -enum -nested -compact
.It
Item two goes here.
.It
And item three here.
.El
.It
And item four here.
.El
```

Result:

1. Item one goes here.
   1.1. Item two goes here.
   1.2. And item three here.
2. And item four here.

–item A list of type **–item** without list markers.

```
.Bl -item -offset indent
.It
Item one goes here.
Item one goes here.
Item one goes here.
.It
Item two here.
Item two here.
Item two here.
.El
```

Produces:

Item one goes here. Item one goes here. Item one goes here.

Item two here. Item two here. Item two here.

–tag A list with tags. Use **–width** to specify the tag width.

SL sleep time of the process (seconds blocked)

PAGEIN
: number of disk I/O operations resulting from references by the process to pages not loaded in core.

UID numerical user-id of process owner

PPID numerical id of parent of process priority (non-positive when in non-interruptible wait)

The raw text:

```
.Bl -tag -width "PPID" -compact -offset indent
.It SL
sleep time of the process (seconds blocked)
```

```
.It PAGEIN
number of disk I/O operations resulting from references
by the process to pages not loaded in core.
.It UID
numerical user-id of process owner
.It PPID
numerical id of parent of process priority
(non-positive when in non-interruptible wait)
.El
```

−diag Diag lists create section four diagnostic lists and are similar to inset lists except callable macros are ignored. The **−width** flag is not meaningful in this context.

Example:

```
.Bl -diag
.It You can't use Sy here.
The message says all.
.El
```

produces

You can't use Sy here. The message says all.

−hang A list with hanging tags.

Hanged labels appear similar to tagged lists when the label is smaller than the label width.

Longer hanged list labels blend into the paragraph unlike tagged paragraph labels.

And the unformatted text which created it:

```
.Bl -hang -offset indent
.It Em Hanged
labels appear similar to tagged lists when the
label is smaller than the label width.
.It Em Longer hanged list labels
blend into the paragraph unlike
tagged paragraph labels.
.El
```

−ohang Lists with overhanging tags do not use indentation for the items; tags are written to a separate line.

SL
sleep time of the process (seconds blocked)

PAGEIN
number of disk I/O operations resulting from references by the process to pages not loaded in core.

UID
numerical user-id of process owner

PPID
numerical id of parent of process priority (non-positive when in non-interruptible wait)

The raw text:

```
.Bl -ohang -offset indent
.It Sy SL
sleep time of the process (seconds blocked)
.It Sy PAGEIN
```

```
number of disk I/O operations resulting from references
by the process to pages not loaded in core.
.It Sy UID
numerical user-id of process owner
.It Sy PPID
numerical id of parent of process priority
(non-positive when in non-interruptible wait)
.El
```

−inset Here is an example of inset labels:

Tag The tagged list (also called a tagged paragraph) is the most common type of list used in the Berkeley manuals. Use a **−width** attribute as described below.

Diag Diag lists create section four diagnostic lists and are similar to inset lists except callable macros are ignored.

Hang Hanged labels are a matter of taste.

Ohang Overhanging labels are nice when space is constrained.

Inset Inset labels are useful for controlling blocks of paragraphs and are valuable for converting *mdoc* manuals to other formats.

Here is the source text which produced the above example:

```
.Bl -inset -offset indent
.It Em Tag
The tagged list (also called a tagged paragraph)
is the most common type of list used in the
Berkeley manuals.
.It Em Diag
Diag lists create section four diagnostic lists
and are similar to inset lists except callable
macros are ignored.
.It Em Hang
Hanged labels are a matter of taste.
.It Em Ohang
Overhanging labels are nice when space is constrained.
.It Em Inset
Inset labels are useful for controlling blocks of
paragraphs and are valuable for converting
.Xr mdoc
manuals to other formats.
.El
```

−column This list type generates multiple columns. The number of columns and the width of each column is determined by the arguments to the **−column** list, ⟨*string1*⟩, ⟨*string2*⟩, etc. If ⟨*stringN*⟩ starts with a `.` (dot) immediately followed by a valid *mdoc* macro name, interpret ⟨*stringN*⟩ and use the width of the result. Otherwise, the width of ⟨*stringN*⟩ (typeset with a fixed-width font) is taken as the *N*th column width.

Each `.It` argument is parsed to make a row, each column within the row is a separate argument separated by a tab or the `.Ta` macro.

The table:

String	Nroff	Troff
<=	<=	≤
>=	>=	≥

was produced by:

```
.Bl -column -offset indent ".Sy String" ".Sy Nroff" ".Sy Troff"
.It Sy String Ta Sy Nroff Ta Sy Troff
.It Li <= Ta <= Ta \*(<=
.It Li >= Ta >= Ta \*(>=
.El
```

Don't abuse this list type! For more complicated cases it might be far better and easier to use *tbl*(1), the table preprocessor.

Other keywords:

-width ⟨*string*⟩ If ⟨*string*⟩ starts with a `.` (dot) immediately followed by a valid *mdoc* macro name, interpret ⟨*string*⟩ and use the width of the result. Almost all lists in this document use this option.

Example:

```
.Bl -tag -width ".Fl test Ao Ar string Ac"
.It Fl test Ao Ar string Ac
This is a longer sentence to show how the
.Fl width
flag works in combination with a tag list.
.El
```

gives:

-test ⟨*string*⟩ This is a longer sentence to show how the **-width** flag works in combination with a tag list.

(Note that the current state of *mdoc* is saved before ⟨*string*⟩ is interpreted; afterward, all variables are restored again. However, boxes (used for enclosures) can't be saved in GNU *troff*(1); as a consequence, arguments must always be *balanced* to avoid nasty errors. For example, do not write `.Ao Ar string` but `.Ao Ar string Xc` instead if you really need only an opening angle bracket.)

Otherwise, if ⟨*string*⟩ is a valid numeric expression (*with a scaling indicator other than* '*u*'), use that value for indentation. The most useful scaling indicators are 'm' and 'n', specifying the so-called *Em* and *En square*. This is approximately the width of the letters 'm' and 'n' respectively of the current font (for *nroff* output, both scaling indicators give the same values). If ⟨*string*⟩ isn't a numeric expression, it is tested whether it is an *mdoc* macro name, and the default width value associated with this macro is used. Finally, if all tests fail, the width of ⟨*string*⟩ (typeset with a fixed-width font) is taken as the width.

If a width is not specified for the tag list type, '6n' is used.

-offset ⟨*string*⟩ If ⟨*string*⟩ is *indent*, a default indent value (normally set to 6n, similar to the value used in `.Dl` or `.Bd`) is used. If ⟨*string*⟩ is a valid numeric expression instead (*with a scaling indicator other than* '*u*'), use that value for indentation. The most useful scaling indicators are 'm' and 'n', specifying the so-called *Em* and *En square*. This is approximately the width of the letters 'm' and 'n' respectively of the current font (for *nroff* output, both scaling indicators give the same values). If ⟨*string*⟩ isn't a numeric expression, it is tested whether it is an *mdoc* macro name, and the default offset value associated with this macro is used. Finally, if all tests fail, the width of ⟨*string*⟩ (typeset with a fixed-width font) is taken as the offset.

–compact Suppress insertion of vertical space before the list and between list items.

Miscellaneous macros

A double handful of macros fit only uncomfortably into one of the above sections. Of these, we couldn't find attested examples for `Me` or `Ot`. They are documented here for completeness—if you know their proper usage, please send a mail to groff@gnu.org and include a specimen with its provenance.

.Bt formats boilerplate text.

.Bt → is currently in beta test.

It is neither callable nor parsed and takes no arguments. Its default width is 6n.

.Fr is an obsolete means of specifying a function return value.

Usage: .**Fr** *return-value* ...

`Fr` allows a break right before the return value (usually a single digit) which is bad typographical behaviour. Instead, set the return value with the rest of the code, using `\~` to tie the return value to the previous word.

Its default width is 12n.

.Hf Inlines the contents of a (header) file into the document.

Usage: .**Hf** *file*

It first prints `File:` followed by the file name, then the contents of *file*. It is neither callable nor parsed.

.Lk Embed hyperlink. This is a GNU extension introduced in *groff* 1.17 (April 2001).

Usage: .**Lk** *uri* [*link-text*]

Its default width is 6n.

.Me Usage unknown. The *mdoc* sources describe it as a macro for "menu entries".

Its default width is 6n.

.Mt Embed email address. This is a GNU extension introduced in *groff* 1.17 (April 2001).

Usage: .**Mt** *email-address*

Its default width is 6n.

.Ot Usage unknown. The *mdoc* sources describe it as "old function type (fortran)".

.Sm Manipulate or toggle argument-spacing mode.

Usage: .**Sm** [**on** | **off**] ...

If argument-spacing mode is off, no spaces between macro arguments are inserted. If called without a parameter (or if the next parameter is neither `on` nor `off`), `Sm` toggles argument-spacing mode.

Its default width is 8n.

.Ud formats boilerplate text.

.Ud → currently under development.

It is neither callable nor parsed and takes no arguments. Its default width is 8n.

Predefined strings

The following strings are predefined for compatibility with legacy *mdoc* documents. Contemporary ones should use the alternatives shown in the "Prefer" column below. See *groff_char*(7) for a full discussion of these special character escape sequences.

String	7-bit	8-bit	UCS	Prefer	Meaning
*(<=	<=	<=	≤	\[<=]	less than or equal to
*(>=	>=	>=	≥	\[>=]	greater than or equal to
*(Rq	"	"	”	\[rq]	right double quote
*(Lq	"	"	“	\[lq]	left double quote
*(ua	^	^	↑	\[ua]	vertical arrow up
*(aa	'	´	´	\[aa]	acute accent
*(ga	`	`	`	\[ga]	grave accent
*(q	"	"	"	\[dq]	neutral double quote
*(Pi	pi	pi	π	\[*p]	lowercase pi
*(Ne	!=	!=	≠	\[!=]	not equals
*(Le	<=	<=	≤	\[<=]	less than or equal to
*(Ge	>=	>=	≥	\[>=]	greater than or equal to
*(Lt	<	<	<	<	less than
*(Gt	>	>	>	>	greater than
*(Pm	+−	±	±	\[+-]	plus or minus
*(If	infinity	infinity	∞	\[if]	infinity
*(Am	&	&	&	&	ampersand
*(Na	*NaN*	*NaN*	*NaN*	.Em NaN	not a number (slanted)
*(Ba	|	|	|	|	bar (upright)

Some column headings are shorthand for standardized character encodings; “7-bit” for ISO 646:1991 IRV (US-ASCII), “8-bit” for ISO Latin-1 (8859-1), and “UCS” for ISO 10646 (Unicode character set). Historically, *mdoc* configured the string definitions to fit the capabilities expected of the output device. Old typesetters lacked directional double quotes, producing repeated directional single quotes ‘‘like this’’; early versions of *mdoc* in fact defined the `Lq` and `Rq` strings this way. Nowadays, output drivers take on the responsibility of glyph substitution, as they possess relevant knowledge of their available repertoires. The `Ba` and `Na` strings imply a mandatory typeface: upright (roman) in the former, and slanted (italics) with the latter.

Diagnostics

The debugging macro `.Db` offered by previous versions of *mdoc* is unavailable in GNU *troff*(1) since the latter provides better facilities to check parameters; additionally, *groff mdoc* implements many error and warning messages, making the package more robust and more verbose.

The remaining debugging macro is `.Rd`, which dumps the package’s global register and string contents to the standard error stream. A normal user will never need it.

Options

The following *groff* options set registers (with **−r**) and strings (with **−d**) recognized and used by the *mdoc* macro package. To ensure rendering consistent with output device capabilities and reader preferences, man pages should never manipulate them.

Setting string `AD` configures the adjustment mode for most formatted text. Typical values are `b` for adjustment to both margins (the default), or `l` for left alignment (ragged right margin). Any valid argument to *groff*’s `ad` request may be used. See *groff*(7) for less-common choices.

```
groff -Tutf8 -dAD=l -mdoc groff_mdoc.7 | less -R
```

Setting register `C` to 1 numbers output pages consecutively, rather than resetting the page number to 1 (or the value of register `P`) with each new *mdoc* document.

By default, the package inhibits page breaks, headers, and footers in the midst of the document text if it is being displayed with a terminal device such as ‘latin1’ or ‘utf8’, to enable more efficient viewing of the page. This behavior can be changed to format the page as if for 66-line Teletype output by setting the continuous rendering register `cR` to zero while calling *groff*(1).

```
groff -Tlatin1 -rcR=0 -mdoc foo.man > foo.txt
```

On HTML devices, it cannot be disabled.

Section headings (defined with `.Sh`) and page titles in headers (defined with `.Dt`) can be presented in full capitals by setting the registers `CS` and `CT`, respectively, to 1. These transformations are off by default because they discard case distinction information.

Setting register `D` to 1 enables double-sided page layout, which is only distinct when not continuously rendering. It places the page number at the bottom right on odd-numbered (recto) pages, and at the bottom left on even-numbered (verso) pages, swapping places with the arguments to `.Os`.

```
groff -Tps -rD1 -mdoc foo.man > foo.ps
```

The value of the `FT` register determines the footer's distance from the page bottom; this amount is always negative and should specify a scaling unit. At one half-inch above this location, the page text is broken before writing the footer. It is ignored if continuous rendering is enabled. The default is "−0.5i − 1v".

The `HF` string sets the font used for section and subsection headings; the default is `B` (bold style of the default family). Any valid argument to *groff*'s `ft` request may be used.

Normally, automatic hyphenation is enabled using a mode appropriate to the *groff* locale; see section "Localization" of *groff*(7). It can be disabled by setting the `HY` register to zero.

```
groff -Tutf8 -rHY=0 -mdoc foo.man | less -R
```

The paragraph and subsection heading indentation amounts can be changed by setting the registers `BP` and `SN`.

```
groff -Tutf8 -rBP=4n -rSN=2n -mdoc foo.man | less -R
```

The default paragraph and subsection heading indentation amounts are 5n and 3n, respectively. Section headings are set with an indentation of zero.

The line and title lengths can be changed by setting the registers `LL` and `LT`, respectively:

```
groff -Tutf8 -rLL=100n -rLT=100n -mdoc foo.man | less -R
```

If not set, both registers default to 80n for terminal devices and 6.5i otherwise.

The `MF` string sets the font used for man page identifiers in document headers and footers and in the formatted output of the `Xr` macro. The default is `I` (italic style of the default family). Any valid argument to *groff*'s `ft` request may be used.

Setting the `P` register starts enumeration of pages at its value. The default is 1.

The `PO` register configures the page offset. The default is device-dependent; typically 0 for terminal devices and 1i otherwise.

To change the document font size to 11p or 12p, set register `S` accordingly:

```
groff -Tdvi -rS11 -mdoc foo.man > foo.dvi
```

Register `S` is ignored when formatting for terminal devices.

By default, *groff* hyperlinks the formatted text of `Lk`, `Mt`, and `Xr` calls on output devices that support hyperlinking ("html", "pdf", and terminal devices). Set the `U` register to 0 to disable this feature.

Setting the `X` register to a page number *p* numbers its successors as *p***a**, *p***b**, *p***c**, and so forth. The register tracking the suffixed page letter uses format `a` (see the `af` request in *groff*(7)).

Files

/usr/share/groff/1.24.2/tmac/andoc.tmac

This brief *groff* program detects whether the *man* or *mdoc* macro package is being used by a document and loads the correct macro definitions, taking advantage of the fact that pages using them must call **TH** or **Dd**, respectively, before any other macros. A user typing, for example,

```
groff -mandoc page.1
```

need not know which package the file *page.1* uses. Multiple man pages, in either format, can be handled; *andoc.tmac* reloads each macro package as necessary.

/usr/share/groff/1.24.2/tmac/doc.tmac

implements the bulk of the *groff mdoc* package and loads further components as needed from the *mdoc* subdirectory.

/usr/share/groff/1.24.2/tmac/doc-old.tmac
: implements a legacy version of the *mdoc* package; it was superseded in 4.4BSD (1994) and is needed only to render certain man pages from 4.3BSD-Reno.

/usr/share/groff/1.24.2/tmac/mdoc.tmac
: is a wrapper enabling the package to be loaded with the option "**-m mdoc**".

/usr/share/groff/1.24.2/tmac/mdoc/doc-common
: defines macros, registers, and strings concerned with the production of formatted output. It includes strings of the form `doc-volume-ds-`*X* and `doc-volume-as-`*X* for manual section titles and architecture identifiers, respectively, where *X* is an argument recognized by .**Dt**.

/usr/share/groff/1.24.2/tmac/mdoc/doc-nroff
: defines parameters appropriate for rendering to terminal devices.

/usr/share/groff/1.24.2/tmac/mdoc/doc-ditroff
: defines parameters appropriate for rendering to typesetter devices.

/usr/share/groff/1.24.2/tmac/mdoc/doc-syms
: defines many strings and macros that interpolate formatted text, such as names of operating system releases, ∗BSD libraries, and standards documents. The string names are of the form `doc-str-`*O*`-`*V*, `doc-str-St--`*S*`-`*I* (observe the double dashes), or `doc-str-Lb-`*L*, where *O* is one of the operating system macros from section "General text domain" above, *V* is an encoding of an operating system release (sometimes omitted along with the - preceding it), *S* an identifier for a standards body or committee, *I* one for an issue of a standard promulgated by *S*, and *L* a keyword identifying a ∗BSD library.

/etc/groff/site-tmac/mdoc.local
: This file houses local additions and customizations to the package. It can be empty.

See also

The mandoc project maintains an independent implementation of the *mdoc* language and a renderer that directly parses its markup as well as that of *man*.

groff(1), *man*(1), *troff*(1), *groff_man*(7), *mdoc*(7)

Bugs

Section 3f has not been added to the header routines.

`.Fn` needs to have a check to prevent splitting up the line if its length is too short. Occasionally it separates the last parenthesis, and sometimes looks ridiculous if output lines are being filled.

The list and display macros do not do any keeps and certainly should be able to.

As of *groff* 1.23, `Tn` no longer changes the type size; this functionality may return in a future release.

Name

groff_me – "me" macro package for formatting *roff* documents

Synopsis

groff –me [*option* ...] [*file* ...]
groff –m me [*option* ...] [*file* ...]

Description

The GNU implementation of the *me* macro package is part of the *groff* document formatting system. The *me* package of macro definitions for the *roff* language provides a convenient facility for preparing technical papers in various formats. This version is based on the *me* distributed with 4.4BSD and can be used with the GNU *troff* formatter as well as those descended from AT&T *troff*.

Some formatter requests affect page layout unpredictably when used in conjunction with this package; however, the following may be used with impunity after the first call to a paragraphing macro like **lp** or **pp**. Some arguments are optional; see *groff*(7) for details, particularly of requests whose argument list is designated with an ellipsis. An asterisk ∗ marks *groff* extensions.

ad	*c*	set text adjustment mode to *c*
af	*r f*	assign format *f* to register *r*
am	*m e*	append to macro *m* until *e* called
as	*s t*	append rest of line *t* to string *s*
bp	*n*	begin new page numbered *n*
br		break output line
ce	*n*	center next *n* productive input lines
cp	*n*	en-/disable AT&T *troff* compatibility mode∗
de	*m e*	define macro *m* until *e* called
do	*t*	interpret input *t* with compatibility mode off∗
ds	*s t*	define rest of line *t* as string *s*
el	*t*	interpret *t* if corresponding **ie**'s *p* false
fc	*c d*	set field delimiter *c* and padding glyph *d*
fi		enable filling
hc	*c*	set hyphenation character to *c*
hy	*m*	set automatic hyphenation mode to *m*
ie	*p t*	as **if**, but enable interpretation of later **el**
if	*p t*	if condition *p*, interpret rest of line *t*
in	*h*	set indentation to distance *h*
lc	*c*	set leader repetition glyph to *c*
ls	*n*	set line spacing to *n*
mc	*c h*	set (right) margin glyph to *c* at distance *h*
mk	*r*	mark vertical position in register *r*
na		disable adjustment of text
ne	*v*	need vertical space of distance *v*
nf		disable filling
nh		disable automatic hyphenation
nr	*r n i*	assign register *r* value *n* with auto-increment *i*
ns		begin no-space mode
pl	*v*	set page length to *v*
pn	*n*	set next page number to *n*
po	*h*	set page offset to *h*
rj	*n*	right-align next *n* productive input lines∗
rm	*m*	remove macro, string, or request *m*
rn	*m n*	rename macro, string, or request *m* to *n*
rr	*r*	remove register *r*

rs resume spacing (end no-space mode)
rt *v* return to vertical position set by **mk**, or *v*
so *f* source (interpolate) input file *f*
sp *n* insert *n* lines of vertical space
ta ... set tab stops
tc *c* set tab repetition glyph to *c*
ti *h* set temporary indentation (next line only) to *h*
tl ... output three-part title
tr ... translate characters
ul *n* underline next *n* productive input lines

Except on title pages (produced by calling **tp**), *me* suppresses the output of vertical space at the tops of pages (after the output of any page header); the **sp** request thus does not work there. You can instead call **bl** or enclose the desired spacing request in a diversion, for instance by calling **(b** and **)b**. *me* also intercepts the **ll** request; see the "*me* Reference Manual" for details.

Name space

Objects in *me* follow a rigid naming convention. To avoid conflict, any user-defined register, string, or macro names should be single numerals or uppercase letters, or any longer sequence of letters and numerals with at least one uppercase letter. (For portability between BSD and *groff me*, limit names to two characters, and avoid the name **[** (left square bracket).) The names employed by any preprocessors in use should also not be repurposed.

Macros

$0 post-section heading hook
$1 pre-section depth 1 hook
$2 pre-section depth 2 hook
$3 pre-section depth 3 hook
$4 pre-section depth 4 hook
$5 pre-section depth 5 hook
$6 pre-section depth 6 hook
$C post-chapter title hook
$H page/column heading hook
$c output chapter number and title
$f output footer
$h output header
$p output section heading
$s output footnote area separator
(b begin block
(c begin centered block
(d begin delayed text
(f begin footnote
(l begin list
(q begin long quotation
(x begin index entry
(z begin floating keep
)b end block
)c end centered block
)d end delayed text
)f end footnote
)l end list
)q end long quotation
)x end index entry
)z end floating keep

++ set document segment type
+c begin chapter
1c end multi-column layout
2c begin multi-column layout
EN end *eqn* equation
EQ begin *eqn* equation
GE end *grn* picture with drawing position at bottom
GF end *grn* picture with drawing position at top
GS start *grn* picture
IE end *ideal* picture with drawing position at bottom
IF end *ideal* picture with drawing position at top
IS start *ideal* picture
PE end *pic* picture with drawing position at bottom
PF end *pic* picture with drawing position at top
PS start *pic* picture
TE end *tbl* table
TH end heading for multi-page *tbl* table
TS start *tbl* table
b embolden argument
ba set base indentation
bc begin new column
bi embolden and italicize argument
bx box argument
ef set even-numbered page footer
eh set even-numbered page header
ep end page
fo set footer
he set header
hl draw horizontal line
hx suppress next page's headers/footers
i italicize argument
ip begin indented paragraph
ld reset localization and date registers and strings∗
ll set line length
lp begin fully left-aligned paragraph
np begin numbered paragraph
of set odd-numbered page footer
oh set odd-numbered page header
pd output delayed text
pp begin first-line indented paragraph
q quote argument
r set argument in roman
re reset tab stops
sh begin numbered section
sm set argument at smaller type size
sx change section depth
sz set type size and vertical spacing
tp begin title page
u underline argument
uh begin unnumbered section
xl set line length (current environment only)
xp output index

Some macros are provided for "old" *roff* (1) compatibility. The "*me* Reference Manual" describes alternatives for modern documents.

ar use Arabic numerals for page numbers
bl insert space (even at page top; cf. **sp**)
ix set indentation without break
m1 set page top to header distance
m2 set header to text distance
m3 set text to footer distance
m4 set footer to page bottom distance
n1 begin output line numbering
n2 end or alter output line numbering
pa begin page
ro use Roman numerals for page numbers
sk skip next page

Registers

$0 section depth
$1 first section number component
$2 second section number component
$3 third section number component
$4 fourth section number component
$5 fifth section number component
$6 sixth section number component
$c current column number
$d delayed text number
$f footnote number
$i paragraph base indentation
$l column width
$m number of available columns
$p numbered paragraph number
$s column spacing (indentation)
bi display (block) indentation
bm distance from text area to page bottom
bs display (block) pre/post space
bt block threshold for keeps
ch current chapter number
df display font
dv vertical spacing of displayed text (as percentage)*
es equation pre/post space
ff footnote font
fi footnote indentation (first line only)
fm footer margin
fp footnote type size in points
fs footnote prespace
fu footnote undent (right indentation)
hm header margin
ii indented paragraph indentation
no line numbering offset*
pf paragraph font
pi paragraph indentation
po page offset
pp paragraph type size in points

ps paragraph prespace
qi long quotation left/right indentation
qp long quotation type size in points
qs long quotation pre/post space
sf section title font
si section indentation per level of depth
so additional section title offset
sp section title type size in points
ss section prespace
sx super/subscript line height increase*
tf title font
tm distance from page top to text area
tp title type size in points
tv vertical spacing of text (as percentage)*
xs index entry prespace
xu index undent (right indentation)
y2 year of the century*
y4 year*
yr year minus 1900
zs floating keep pre/post space

Strings

delayed text marker
$n catenated section number
* footnote marker
– em dash
< begin subscripting
> end subscripting
dw weekday name
lq left double quotation mark
mo month name
rq right double quotation mark
td date
wa term for "appendix" used by **.$c***
wc term for "chapter" used by **.$c***
{ begin superscripting
} end superscripting

Files

/usr/share/groff/1.24.2/tmac/e.tmac
implements the package.

/usr/share/groff/1.24.2/tmac/refer−me.tmac
implements *refer*(1) support for *me*.

/usr/share/groff/1.24.2/tmac/me.tmac
is a wrapper enabling the package to be loaded with the option "**−m me**".

Notes

Some early *roff* macro packages limited their names to a single letter, which followed the formatter's **m** option, resulting in "names" like *−mm*, *−ms*, *−mv*, and *−mn*. *me*'s "e" stands for "Eric P. Allman", the author of the original macro package and the technical papers documenting it.

See also

Two manuals are available in source and rendered form. On your system, they may be compressed and/or available in additional formats.

/usr/share/doc/groff/meintro.me
/usr/share/doc/groff/meintro.ps
: is "Writing Papers with *Groff* Using −*me*", by Eric P. Allman, adapted for *groff* by James Clark.

/usr/share/doc/groff/meref.me
/usr/share/doc/groff/meref.ps
: is the "*me* Reference Manual", by Eric P. Allman, adapted for *groff* by James Clark and G. Branden Robinson.

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

For preprocessors supported by *me*, see *eqn*(1), *grn*(1), *pic*(1), *refer*(1), and *tbl*(1).

groff (1), *troff* (1), *groff* (7)

Name

groff_mm – memorandum macros for GNU *roff*

Synopsis

groff –mm [*option* ...] [*file* ...]
groff –m mm [*option* ...] [*file* ...]

Description

The *mm* macro package distributed with the *groff* document formatting system is suitable for the composition of letters, memoranda, reports, and books.

Call an *mm* macro at the beginning of a document to initialize the package. A simple *mm* document might use only **P** for paragraphing. Set numbered and unnumbered section headings with **H** and **HU**, respectively. Change the style of the typeface with **B**, **I**, and **R**; you can alternate styles within a word with **BI**, **BR**, **IB**, **IR**, **RB**, and **RI**. Several nestable list types are available via **AL**, **BL**, **BVL**, **DL**, **ML**, **RL**, and **VL**; each of these begins a list, to which **LI** adds an item and **LE** ends the (nested) list. **LB** begins a list with customizable layout parameters. **DS** and **DF** start static and floating displays, respectively; **DE** terminates either.

groff mm is intended to be compatible with the *mm* implementation found in the AT&T Documenter's Workbench 3.3 ("DWB"), with the following limitations and changes.

- Omitted features include the logo and company name strings, **}Z** and **]S**, respectively; the encoded company site location addresses recognized as the third argument to the **AU** macro; the **Pv** ("private" heading) register; and the **OK** (other keywords) and **PM** (proprietary markings) macros.
- *groff mm* implements the **CS** (output cover page or "sheet") macro only for memorandum type 4.
- The *grap* preprocessor is not explicitly supported; *groff mm* defines no **G1** or **G2** macros.
- Registers **A**, **C**, **T**, and **U**, set from the *troff* or *nroff* command lines with DWB, are not recognized.
- When setting the registers **L** or **W** from the command line, use an explicit scaling unit to avoid surprises.
- The **Le** register defaults to **1**, consistently with **Lf**, **Lt**, and **Lx** in DWB; equations captioned with **EC** appear in tables of contents produced by **TC**.
- The **Li** register configures the text indentation of **RL** list items; DWB used a hard-coded value of 6 ens.
- *groff mm* uses the same adjustment and font style defaults in *nroff* and *troff* modes.
- Cut marks are not supported.

DWB supported only seven heading levels. As a compatible extension, *groff mm* supports fourteen, introducing new registers **H8** through **H14**, and affecting the interpretation of the **HF** and **HP** strings.

groff mm features a citation (or "bookmark") system, permitting the document to make (customizable) internal references like "See chapter 5, page 128.". Forward references require two-pass formatting. See the **INITR** macro description below and *mmroff* (1).

Macro, register, and string descriptions in this page frequently mention each other; most references are to macros. Where a register or string is referenced, its type is explicitly identified. *mm*'s macro names are usually in full capitals; registers and strings tend to have mixed-case names.

Except where noted, *mm* assumes that horizontal measurements are reckoned in ens (scaling unit **n**) and vertical ones in vees (scaling unit **v**). *groff mm* permits use of non-integral *typographical points* (scaling unit **z**). Use explicit scaling units for clarity and predictable behavior.

Document styles

groff mm offers three frameworks for document organization. **COVER/COVEND** is a flexible means of preparing any document requiring a cover page. **LT/LO** aids preparation of typical Anglophone correspondence (business letters, for example). The **MT** memorandum type mechanism implements a group of formal styles historically used by AT&T Bell Laboratories. Your document can select at most one of these approaches; when used, each disables the others.

Localization

groff mm is designed to be easily localized. For languages other than English, strings that can appear in output are collected in the file */usr/share/groff/1.24.2/tmac/*xx*.tmac*, where *xx* is an ISO 639 two-letter language identifier. Localization packages should be loaded after *mm*; for example, you might format a Swedish *mm* document with the command "**groff −mm −msv**".

This package can also be localized by site or territory; for example, */usr/share/groff/1.24.2/tmac/mse.tmac* illustrates how to adapt the output to a national standard using its ISO 3166 territory code. Such a package can define a string that causes a macro file */usr/share/groff/1.24.2/tmac/mm/*territory*_locale* to be loaded at package initialization. If this mechanism is not used, */usr/share/groff/1.24.2/tmac/mm/locale* is loaded instead. No diagnostic is produced if these files do not exist.

Registers and strings

Much *mm* behavior can be configured by registers and strings. The **nr** request assigns a value to a register.

.nr *ident* [±]*n* [*i*]

ident is the name of the register, and *n* is the value to be assigned. Prefixing *n* with a plus or minus sign increments or decrements (respectively) its existing value. If assignment of a (possibly) negative *n* is required, further prefix it with a zero or enclose it in parentheses. If *i* is specified, the register's value automatically changes by *i* prior to interpolation if a plus or minus sign is included in the escape sequence as follows.

\n[±]**[***ident***]**

i can be negative; it combines algebraically with the sign in the interpolation escape sequence. The brackets around *ident* are literal.

Many of the registers *mm* provides are as Boolean-valued, meaning that they are considered "true" (on, enabled) when they have a positive value, and "false" (off, disabled) otherwise.

Define strings with the **ds** request.

.ds *ident contents*

contents consumes everything up to the end of the line, including trailing spaces. It is a good practice to end *contents* with a comment escape sequence (**\"**) so that extraneous spaces do not intrude during document maintenance. To include leading spaces in *contents*, prefix it with a double quote. Interpolate strings with the * escape sequence.

*****[*ident*]

Register and string name spaces are distinct, but strings and macros share a name space. Defining a string with the same name as an *mm* macro is not supported and may cause incorrect rendering, the emission of diagnostic messages, and an error exit status from *troff*.

Register format

A register is interpolated using Arabic numerals if no other format has been assigned to it. Assign a format to a register with the **af** request.

.af *R c*

R is the name of the register, and *c* is the format. If *c* is a sequence of Arabic numerals, their quantity defines a zero-padded minimum width for the interpolated register value.

Form	**Sequence**
1	0, 1, 2, 3, . . ., 10, . . .
001	000, 001, 002, 003, . . ., 1000, . . .
i	0, i, ii, iii, iv, . . .
I	0, I, II, III, IV, . . .
a	0, a, b, c, . . ., z, aa, ab, . . .
A	0, A, B, C, . . ., Z, AA, AB, . . .

Fonts

mm assumes that the font styles **R** (roman), **I** (italic), and **B** (bold) are mounted at font positions **1**, **2**, and **3**, respectively. Use the **fp** request to mount substitute fonts at these positions as desired. The default font family is "**T**" (Times). To select a different one, invoke *groff*'s **fam** request or use its **–f** command-line option.

Macros

Double-quote macro arguments that contain space characters. An explicitly empty argument may be specified with an empty pair of double quotes; for example, the following macro call has three arguments.

```
.XX "foo bar" "" baz
```

Some macros are documented as causing a page break; this does not occur if such a macro is called when the drawing position is already at the top of a page (after a page heading, if any).

Hook macros are undefined by default; *mm* calls them to enable customization of its behavior. (DWB termed these "exits".)

Macro names longer than two characters are extensions; some shorter names were not part of DWB's published interface but are documented aspects of *groff mm*.

1C [**1**]
: Format page text in one column (the default layout). The page is broken. A **1** argument suppresses this break; its use may cause body text and a pending footnote to overprint. See **2C**, **MC**, and **NCOL**.

2C
: Begin two-column formatting. This is a special case of **MC**. See **1C** and **NCOL**.

AE
: Abstract end; stop collecting abstract text. See **AS**.

AF [*org-name*]
: Specify a memorandum's organizational affiliation. At most one can be declared; *org-name* is used by **MT** memoranda and available to cover pages. See **COVER**.

AFX
: Define this hook macro to assume responsibility for formatting the affiliated firm name defined by **AF** in memorandum types 0 and 4 and documents using the *ms* cover page style. If not defined (the default), internally defined macros handle this task; see subsection "Internals" and section "Files" below. Applications include setting the firm name in a different font family or at a larger type size, drawing a rule across the page, and including a logo image using *groff*'s **PDFPIC** or **PSPIC** macros. See **MT**, **COVER**, and *groff_tmac*(5).

AL [*number-format* [*text-indent* [**1**]]]
: Begin an auto-incrementing numbered list, where item numbers start at one and are followed by a dot. *number-format* assigns the register format (see above) *mm* uses for the list item enumerators. The default is **0**. A *text-indent* argument overrides register **Li**. A third argument suppresses the vertical space that normally precedes each list item; see register **Lsp**. Vertical space in the amount of **Lsp** precedes the list itself, but does not accumulate with pre-item space when this list is nested in another. **AL** calls **LB**; use **LI** to declare list items, and **LE** to end the list.

APP [*sequence-number* [*title*]]
: Begin an appendix. If *sequence-number* is omitted, it increments (or is initialized to **1**, if necessary). The register format used for *sequence-number* is "A". The page is broken. The register **Aph** determines whether an appendix heading is then formatted. This heading uses the string **App** followed by *sequence-number*. Appendices appear in any table of contents (see **TC**). The string **Apptxt** is set to *title* if the latter is present, and made empty otherwise.

APPSK *sequence-number n* [*title*]
: As **APP**, but increment the page number by *n*. Use this macro to "skip pages" when diagrams or other materials not formatted by *troff* are included in appendices.

AS [*placement* [*indentation*]]
: Abstract start; begin collecting abstract. Input up to the next **AE** call is included in the abstract. *placement* influences the location of the abstract on the cover page of a memorandum (see **MT**).

COVER, by contrast, ignores *placement* by default, but can be customized to interpret it.

placement	**Effect**
0	The abstract appears on page 1 and cover page if the document is a "released paper" memorandum ("**.MT 4**"); otherwise, it appears on page 1 without a cover page.
1	The abstract appears only on the cover page ("**.MT 4**" only).

An abstract does not appear at all in external letters ("**.MT 5**"). A *placement* of **2** was supported by DWB, but is not by *groff mm*.

A second argument increases the indentation by *indentation* and reduces the line length by twice this amount. The default is **0**.

AT *title* ...

Specify author's title(s). If present, **AT** must appear just after the corresponding author's **AU**. Each *title* occupies an output line beneath the author's name in the signature block used by **LT** letters (see **SG**) and in **MT** memoranda. The *ms* cover page style also formats these data.

AU [*name* [*initials* [*loc* [*dept* [*ext* [*room* [*arg1* [*arg2* [*arg3*]]]]]]]]]

Specify author. **AU** terminates a document title started with **TL**, and can be called without arguments for that purpose. Author information is used by cover pages, **MT** memoranda, and **SG**. Further arguments comprise initials, location, department, telephone extension, room number or name, and up to three additional items. Repeat **AU** to identify multiple authors.

Use **WA/WE** instead to identify the author for documents employing **LT**.

AV [*name* [**1**]]

Format approval lines for a handwritten signature and date. Two horizontal rules are drawn, with the specified *name* and the text of the string **Letdate**, respectively, beneath them. Above these rules, *mm* formats the text in the string **Letapp**; a second argument replaces this text with one vee of vertical space. See **LT**.

AVL [*name*]

As **AV**, but omitting the approval notation (the **Letapp** string), the date rule, and the label below the date rule (the **Letdate** string).

B [*bold-text* [*previous-font-text*]] ...

Join *bold-text* in boldface with *previous-font-text* in the previous font, without space between the arguments. If no arguments, switch font to bold style.

B1 Begin boxed static display. The text is indented by **1n**, and the line length reduced by **2n**. See **DS**. This is a *groff mm* extension.

B2 End boxed static display. See **B1**. This is a *groff mm* extension.

BE End bottom block; see **BS**.

BI [*bold-text* [*italic-text*]] ...

Join *bold-text* in boldface with *italic-text* in italics, without space between the arguments.

BL [*text-indent* [**1**]]

Begin bulleted list. *mm* marks each item with the string **BU**. A *text-indent* argument overrides register **Pi**. A second argument suppresses the vertical space that normally precedes each list item; see register **Lsp**. Vertical space in the amount of **Lsp** precedes the list itself, but does not accumulate with pre-item space when this list is nested in another. **BL** calls **LB**; use **LI** to declare list items, and **LE** to end the list.

BR [*bold-text* [*roman-text*]] ...

Join *bold-text* in boldface with *roman-text* in roman style, without space between the arguments.

BS Begin bottom block. *mm* collects input into the bottom block until the document calls **BE**, and outputs it between the footnote area and footer of each page.

BVL [*text-indent* [*mark-indent* [**1**]]]
: Begin broken variable-item (or "tagged") list. Each item should supply its own mark. The line is always broken after the mark; contrast **VL**. A *text-indent* argument overrides register **Pi**; *mark-indent* sets the distance from the indentation of the current list to the mark. A third argument suppresses the vertical space that normally precedes each list item; see register **Lsp**. Vertical space in the amount of **Lsp** precedes the list itself, but does not accumulate with pre-item space when this list is nested in another. **BVL** calls **LB**; use **LI** to declare list items, and **LE** to end the list.

COVER [*style*]
: Begin a cover page description. **COVER** must appear before the body text (or main matter) of a document. The argument *style* is used to construct the file name */usr/share/groff/1.24.2/tmac/mm/*style*.cov* and load it with the **mso** request. The default *style* is "**ms**"; the *ms.cov* file prepares a cover page resembling that of the *ms* package. A *.cov* file must define a **COVEND** macro, which a document must call at the end of the cover description. Use cover description macros in the following order; only **TL** and **AU** are required.

```
.COVER
.AF
.TL
.AU
.AT
.\" Add additional AU and AT calls as needed.
.AS
.AE
.COVEND
```

COVEND
: End the cover description.

DE
: End static or floating display begun with **DS** or **DF**.

DF [*format* [*fill* [*right-indentation*]]]
: Begin floating display. A floating display is saved in a queue and output in the order entered. Arguments are handled as in **DS**. Floating displays cannot be nested. Placement of floating displays is controlled by the registers **De** and **Df**.

DL [*text-indent* [**1**]]
: Begin dashed list. *mm* marks each item with the string **EM**. A *text-indent* argument overrides register **Pi**. A second argument suppresses the vertical space that normally precedes each list item; see register **Lsp**. Vertical space in the amount of **Lsp** precedes the list itself, but does not accumulate with pre-item space when this list is nested in another. **DL** calls **LB**; use **LI** to declare list items, and **LE** to end the list.

DS [*format* [*fill* [*right-indentation*]]]
: Begin static display. *mm* collects input into a display until the document calls **DE**. The display is output on a single page unless it is taller than the height of the page. **DS** can be nested (contrast with **DF**).

format	**Effect**
none	Do not indent the display.
L	Do not indent the display.
I	Indent text by **\n[Si]**.
C	Center each line.
CB	Center the whole display as a block.
R	Right-align each line.
RB	Right-align the whole display as a block.

The values "L", "I", "C", and "CB" can also be specified as "0", "1", "2", and "3", respectively, for compatibility with DWB.

fill	**Effect**
none	Disable filling.
N	Disable filling.
F	Enable filling.

"N" and "F" can also be specified as "0" and "1", respectively, for compatibility with DWB.

A third argument reduces the line length by *right-indentation.*

mm normally places vertical space before and after the display. Set register **Ds** to "0" to suppress it.

EC [*title* [*override* [*flag* [*ref-name*]]]]

Caption an equation. A caption comprises the string **Capec** followed by an automatically incrementing counter stored in the register **Ec**, punctuation configured by the register **Of**, then *title* (if any). Use the **af** request to configure **Ec**'s number format. *override* and *flag* alter the equation number as follows. Omitting *flag* and specifying **0** in its place are equivalent.

flag	**Effect**
0	Prefix number with *override*.
1	Suffix number with *override*.
2	Replace number with *override*.

mm centers equation captions irrespective of the alignment of any enclosing display.

Specifying *ref-name* stores the equation number as if by "**.SETR** *ref-name* **\n[Ec]**". Recognition of this argument is a *groff mm* extension.

The package lists captioned equations in a table of contents (see **TC**) if the Boolean register **Le** is true. The string **Le** captions this list.

EF ["' '*left*'*center*'*right*' "']

Define the even-page footer, which is formatted just above the normal page footer on even-numbered pages. See **PF**. **EF** defines the string **EOPef**.

EH ["' '*left*'*center*'*right*' "']

Define the even-page header, which is formatted just below the normal page header on even-numbered pages. See **PH**. **EH** defines the string **TPeh**.

EN End mathematical expression input preprocessed by *eqn*(1); see **EQ**.

EOP *groff mm* calls this hook macro in lieu of normal page footer layout. The package uses a separate environment to format headers and footers. See **TP**.

Strings available to EOP	
EOPf	argument to **PF**
EOPef	argument to **EF**
EOPof	argument to **OF**

EPIC [**–L**] *width height* [*name*]

Draw a box with the given *width* and *height*. The text *name* (or default text) is formatted inside the box; no attempt is made to size the box to fit the text. An application of this macro is to indicate the placement of an image to be determined later, or externally composited in postprocessing. **–L** as the first argument left-aligns the box; the default is to center it. See **PIC**.

EQ [*label*]

Start mathematical expression input preprocessed by *eqn*(1). **EQ** and **EN** macro calls bracket an equation region. Such regions must be contained in displays (**DS**/**DE**), except when the region is used only to configure *eqn* and not to produce output. If present, *label* appears aligned to the right and centered vertically within the display; see register **Eq**. If multiple *eqn* regions occur within a

display, only the last *label* (if any) is used.

EX [*title* [*override* [*flag* [*ref-name*]]]]
Caption an exhibit. Arguments are handled analogously to **EC**. The register **Ex** is the exhibit counter. The string **Capex** precedes the exhibit number and any *title*. *mm* centers exhibit captions irrespective of the alignment of any enclosing display.

The package lists captioned exhibits in a table of contents (see **TC**) if the Boolean register **Lx** is true. The string **Lx** captions this list.

FC [*closing-text*]
Output the string **Letfc**, or the specified *closing-text,* as the formal closing of a letter or memorandum. See **LT** and **MT**.

FD [*arg* [**1**]]
Configure display of footnotes. The first argument encodes enablement of automatic hyphenation, adjustment to both margins, indentation of footnote text, and left- vs. right-alignment of the footnote label within the space allocated for it.

arg	**Hyphenate?**	**Adjust?**	**Indent?**	**Label alignment**
0	no	yes	yes	left
1	yes	yes	yes	left
2	no	no	yes	left
3	yes	no	yes	left
4	no	yes	no	left
5	yes	yes	no	left
6	no	no	no	left
7	yes	no	no	left
8	no	yes	yes	right
9	yes	yes	yes	right
10	no	no	yes	right
11	yes	no	yes	right

An *arg* greater than 11 is treated as **0**. *mm*'s default is **0**.

A second argument resets footnote numbering when a first-level heading is encountered. See **FS**.

FE End footnote; see **FS**.

FG [*title* [*override* [*flag* [*ref-name*]]]]
Caption a figure. Arguments are handled analogously to **EC**. The register **Fg** is the figure counter. The string **Capfg** precedes the figure number and any *title*. *mm* centers figure captions irrespective of the alignment of any enclosing display.

The package lists captioned figures in a table of contents (see **TC**) if the Boolean register **Lf** is true. The string **Lf** captions this list.

FS [*mark*]
Start footnote. *mm* collects input into a footnote until the document calls **FE**, outputting it when the vertical drawing position nears the page bottom. By default, *mm* automatically numbers footnotes starting at 1; the number is available in register **Ftnum** and, with a trailing period, in string **F**. If desired, interpolate this string in the body text to format the footnote mark. *mm* interpolates string **F** prior to the footnote text. Footnotes are vertically separated by the product of registers **Fs** and **Lsp**. In *groff mm*, footnotes may be used in displays.

A *mark* argument, which need not be numeric, replaces the default footnote mark, leaving the string **F** unchanged. In that event, you must explicitly write any footnote mark to appear in the footnote itself.

GETHN *ref-name* [*string*]
Interpolate *ref-name*'s heading mark, or, if *string* is specified, store it there. In neither case does **GETHN** suffix the interpolation with space. See **INITR**.

GETPN *ref-name* [*string*]
Interpolate *ref-name*'s page number, or, if *string* is specified, store it there. In neither case does **GETPN** suffix the interpolation with space. See **INITR**.

GETR *ref-name*
Retrieve location data for reference *ref-name*, call **GETHN** and **GETPN** to populate strings **Qrfh** and **Qrfn**, respectively, and interpolate string **Qrf** as an internal reference to it. See **INITR**.

GETST *ref-name* [*string*]
Interpolate *ref-name*'s auxiliary reference datum, or, if *string* is specified, store it there. In neither case does **GETST** suffix the interpolation with space. See **INITR**.

H *level* [*title* [*suffix*]]
Set a numbered section heading at *level*. *mm* calls **LC** to clear any lists, then produces numbered *heading marks* of the form $a.b.c\ldots$, with up to fourteen levels of nesting. Each level's number increments automatically with each **H** call and is reset to zero when a more significant *level* is specified. "**1**" is the most significant or coarsest division of the document. Text after an **H** call is formatted as a paragraph; calling **P** is unnecessary.

The optional *title* must be double-quoted if it contains spaces. *mm* appends *suffix* to *title* in the body of the document, but omits it from any table of contents (see **TC**). This facility can be used to annotate the heading title with a footnote. *suffix* should not interpolate the **F** string; specify a footnote mark explicitly. See **FS**.

Heading behavior is configurable. Several registers set *thresholds,* where heading levels at or below a threshold value are handled in one way, and those above it another. For example, *mm* populates a table of contents (see **TC**) with the title of a heading if its level is within the **Cl** register threshold.

Heading layout. Register **Ej** sets a threshold for page breaking (ejection) prior to a heading. If not preceded by a page break, a heading level within the threshold in register **Hps** is preceded by the amount of vertical space in register **Hps1**, and by the amount in **Hps2** otherwise. The **Hb** register sets a threshold at which a break occurs after the heading, and register **Hs** sets a threshold within which vertical space follows it. If the heading level is above both of these, and the paragraph is not numbered, *mm* produces a *run-in heading*; paragraph text follows on the same output line. Otherwise, register **Hi** configures the indentation of text after headings. Threshold register **Hc** permits centering; *mm* centers a heading with a level within both of the **Hb** and **Hc** thresholds.

Heading typeface and size. The **HF** string configures the fonts used for heading marks and titles at each level. The string **HP** likewise assigns a type size to each heading level. The hook macros **HX** and/or **HZ** can control the vertical spacing used by headings.

Heading number format. Registers named **H1** through **H14** store counters for each heading level. Their values are printed using Arabic numerals by default; see **HM**. The heading levels are catenated with dots for formatting; to typeset only the deepest, set the **Ht** register. Heading numbers are not suffixed with a trailing dot except when only the first level is output; to omit a dot in this case as well, clear the **H1dot** register.

Customizing heading behavior. **H** calls **HX**, **HY**, and **HZ** hook macros to further customize headings. These can change the heading's *mark* (the numbered portion before any heading title), its vertical spacing, and its vertical space requirements (for instance, to require a minimum quantity of subsequent output lines on the page, breaking the page otherwise). Define hook macros in expectation of the following parameters. The argument *declared-level* is the *level* argument to **H**, or **0** for unnumbered headings (see **HU**). *actual-level* is the same as *declared-level* for numbered headings, and the value of register **Hu** for unnumbered headings. *title-suffix* is the catenation of the corresponding arguments to **H** or **HU**.

HX *declared-level actual-level title-suffix*
mm calls **HX** before setting the heading. Your definition may alter **}0**, **}2**, and **;3**.

}0 (string)
contains the heading mark plus two spaces if *declared-level* is non-zero, and otherwise is empty.

;0 (register)
encodes a position for the text after the heading. 0 means that the heading is to be run in, 1 means that a break is to occur before the text, and 2 means that vertical space is to separate heading and text.

}2 (string)
is the suffix that separates a run-in heading from the text. It contains two spaces if register **;0** is 0, and otherwise is empty.

;3 (register)
contains the vertical space required for the heading to be typeset. If that amount is not available, the page is broken prior to the heading. The default is **2v**.

HY *declared-level actual-level title-suffix*
mm calls **HY** after determining the heading typeface and size. It could be used to change indentation.

HZ *declared-level actual-level title-suffix*
mm calls **HZ** after formatting the heading, just before **H** or **HU** returns. It could be used to change the page header to include a section heading.

HC [*hyphenation-character*]
Set hyphenation character, selecting the default, "\%", if called without argument.

HM [*arg1* [*arg2* [. . . [*arg14*]]]]
Set the heading mark style. Each argument assigns the specified register format (see above) to the corresponding heading level. The default is **1** for all levels. An explicitly empty argument also indicates the default.

HU *title* [*suffix*]
Set an unnumbered section heading with *title* and, as a *groff mm* extension, an optional *suffix*. The heading is treated as a numbered heading of the level stored in register **Hu**, but no heading mark is output; see **H**.

I [*italic-text* [*previous-font-text*]] . . .
Join *italic-text* in italics with *previous-font-text* in the previous font, without space between the arguments. If no arguments, switch font to italic style.

IA [*recipient-name* [*title*]]
Specify the inside address in a letter. The arguments give each recipient a name and title. *mm* collects input into the inside address until the document calls **IE**, then outputs it. You can specify multiple recipients with empty **IA/IE** pairs; *mm* uses only the last address. See **LT**.

IB [*italic-text* [*bold-text*]] . . .
Join *italic-text* in italics with *bold-text* in boldface, without space between the arguments.

IE End the inside address begun with **IA**.

IND *argument* . . .
If the Boolean register **Ref** is true, write an index entry as a specially prepared *roff* comment to the standard error stream, with each *argument* separated from its predecessor by a tab character. The entry's location information is arranged as configured by the most recent **INITI** call.

INDP Output the index set up by **INITI** and populated by **IND** calls. By default, **INDP** calls **SK** and writes a centered caption interpolating the string **Index**. It then disables filling and calls **2C**; afterward, it restores filling and calls **1C**.

Define macros to customize this behavior. **INDP** calls **TXIND** before the caption, **TYIND** *instead* of writing the caption, and **TZIND** after formatting the index.

INITI *location-type file-name* [*macro*]
Initialize *groff mm*'s indexing system. Argument *location-type* selects how the location of each index entry is reported. *file-name* populates an internal string used later by **INDP**.

location-type	**Entry format**
N	page number
H	heading mark
B	page number, tab character, heading mark

If *macro* is specified, *groff mm* calls it for each index entry with the arguments given to **IND**.

INITR *file-name-prefix*
Initialize the internal reference macros. Internal references are written to the standard error stream, which should be redirected into a file named file-name-prefix.*qrf*. *mmroff*(1) handles this and the two formatting passes it requires. The first pass identifies internal references; the second includes them. See **SETR**, **GETHN**, **GETPN**, and **GETST**.

IR [*italic-text* [*roman-text*]] . . .
Join *italic-text* in italics with *roman-text* in roman style, without space between the arguments.

LB *text-indent mark-indent pad type* [*mark-or-format* [*pre-item-space* [*pre-list-space*]]]
Begin list. The macros **AL**, **BL**, **BVL**, **DL**, **ML**, **RL**, and **VL** call **LB** in various ways; they are simpler to use and may be preferred if they suit the desired purpose.

mm tracks the nesting level of lists; the outermost is 0. It indents each list item (text formatted after an **LI** call) by *text-indent*. The mark is left-aligned at *mark-indent*, and padded on the left with *pad* ens of space. *type* determines the mark's format.

type	**Output for a mark "x"**
0	x
1	x.
2	x)
3	(x)
4	[x]
5	<x>
6	{x}

If *type* is **0**, *mark-or-format* specifies each item's mark (which can be overridden by an argument to **LI**). If *mark-or-format* is empty (or the dummy character "**\&**") *mm* sets items at *mark-indent* with a hanging indent at *text-indent*.

If *type* is greater than zero, *mm* supplies an automatically incrementing numeric mark, and interprets *mark-or-format* as a register format.

A *type* of **−1** causes *groff mm* to break the line after the mark even if it fits within *text-indent;* this is a *groff mm* extension.

The last two arguments manage vertical space. Unless a list's nesting level is greater than the value of register **Ls**, its items are preceded by *pre-item-space* multiplied by the register **Lsp**; the default is **1**. **LB** precedes the list by *pre-list-space* multiplied by the register **Lsp**; the default is **0**.

LC [*list-level*]
Clear list state. Active lists are terminated as if with **LE**, either all (the default) or only those from the current level down to *list-level* if specified.

LE [**1**] End list. The current list is terminated. An argument of **1** causes vertical space in the amount of register **Lsp** to follow the list.

LI [*mark* [*pad-prefix*]]
Begin a list item. *mm* collects input into a list item until the document terminates the current list or calls **LI** again. By default, the item's text is preceded by any mark configured by the current list. If *mark* is the only argument, it replaces the mark configured in the corresponding **LB** call. If the width of the mark plus any *pad* specified in the **LB** call exceeds the text indentation, *groff mm* warns and uses one en of padding between the mark and the text. The presence of a second argument prefixes *mark* to the **LB**-configured mark and, as a *groff mm* extension, conditionally puts an unbreakable space between the prefix and mark per the argument's Boolean value.

LO *option* [*value*]
Specify letter options; see **LT**. By default, *mm* recognizes the following *option* values. See **IA** regarding the inside address and string **DT** regarding the date.

option	**Meaning and Effect**
AT	Attention; put contents of string **LetAT** and *value* left-aligned after the inside address.
CN	Confidential; put *value,* or contents of string **LetCN**, left-aligned after the date.
RN	Reference; put contents of string **LetRN** and *value* after the confidential notation (if any) and the date, aligned with the latter.
SA	Salutation; put *value,* or contents of string **LetSA**, left-aligned after the inside address and the confidential notation (if any).
SJ	Subject; put contents of string **LetSJ** and *value* left-aligned after the inside address and the attention and salutation notations (if any). In letter type "SP", **LetSJ** is ignored and *value* is set in full capitals.

LT [*style*]
Format a letter in the designated *style,* defaulting to **BL** (see below). A letter begins with the writer's address (**WA/WE**), followed by the date (**ND**), the inside address (**IA/IE**), the body of the letter (**P** and other general-purpose *mm* macros), the formal closing (**FC**), the signature (**SG**), and notations (**NS/NE**). Any of these except the body text may be omitted. Letter options specified with **LO** add further annotations, which are extensible; see subsection "Internals" below.

style	**Description**
BL	Blocked: the writer's address, date, formal closing, and signature are indented to the center of the line. Everything else is left-aligned.
SB	Semi-blocked: as **BL**, but the first line of each paragraph is indented by **\n[Pi]** ens.
FB	Fully blocked: everything is left-aligned.
SP	Simplified: as **FB**, but a formal closing is omitted, and the signature is set in full capitals.

MC *column-width* [*gutter-width*]
Begin multi-column layout. *groff mm* creates as many columns of *column-width* as the line length will permit. *gutter-width* is the interior spacing between columns. It defaults to *column-width*/15. **NCOL** moves to the next column, and **1C** returns to single-column layout. **MC** is a *groff mm* extension, generalizing **2C**. See **MULB** for an alternative.

ML *mark* [*text-indent* [**1**]]
Start a *marked list;* the *mark* argument precedes each item. *text-indent* overrides the default indentation of the list items, which is the width of *mark* plus one en. A third argument suppresses the vertical space that normally precedes each list item; see register **Lsp**. Vertical space in the amount of **Lsp** precedes the list itself, but does not accumulate with pre-item space when this list is nested in another. **ML** calls **LB**; use **LI** to declare list items, and **LE** to end the list.

MT [*type* [*addressee*]]
Select memorandum type. These correspond to formats used by AT&T Bell Laboratories, where the *mm* package was initially developed, affecting the document layout. Some of these included a cover page with a caption categorizing the document. *groff mm* uses *type* to construct the file name */usr/*

*share/groff/1.24.2/tmac/mm/*type.*MT* and load it with the **mso** request. Memorandum types 0 to 5 are supported; any other value of *type* is mapped to type 6. Omitting *type* implies **1**. *addressee* sets a string analogous to one used by AT&T cover page macros not implemented in *groff mm*.

type	**Description**
0	normal memorandum; no caption
1	captioned "TECHNICAL MEMORANDUM"
2	captioned "INTERNAL MEMORANDUM"
3	captioned "ADMINISTRATIVE MEMORANDUM"
4	released paper
5	external letter
string	captioned *string*

See **COVER** for a more flexible cover page mechanism.

MOVE *y-pos* [*x-pos* [*line-length*]]
Move to a position, setting page offset to *x-pos*. If *line-length* is not given, the difference between current and new page offset is used. Use **PGFORM** without arguments to return to normal.

MULB *cw1 sp1* [*cw2 sp2*] . . . *cwn*
Begin alternative multi-column mode. All column widths *cw*i must be specified, as must the amount of space *sp*i between each column pair. **MULB** uses a diversion and operates in a separate environment.

MULE End alternative multi-column mode and emit the columns.

MULN
Start next column in alternative column mode.

NCOL Start next column (only when using **2C** or **MC**). Contrast with **MULN**.

ND [*arg*]
Set the document's date. *mm* does not interpret *arg*; it can be a revision identifier (or empty). *mm* calls **ND** itself when initializing.

NE End notation begun with **NS**; filling is enabled.

nP [*type*]
As **P**, but set a paragraph number in the form *x*.*yy*, where *x* is the number of the second heading level, and *yy* increments with each **nP** call; its value is not reset when the second-level heading number changes. *mm* uses a register format of "00" for *yy*.

NS [*code* [**1**]]
Declare notations, typically for letters or memoranda, of the type specified by *code*. The text corresponding to *code* is output, and filling is disabled until the document calls **NE**. Typically, a list of names or attachments lies within **NS/NE**. If *code* is absent or does not match one of the values listed under the **Letns** string description below, each line of notations is formatted as "Copy (*line*) to". If a second argument, conventionally **1**, is given, *code* becomes the entire notation and **NE** is not necessary. In *groff mm*, you can set up further notations to be recognized by **NS**; see the strings **Letns** and **Letnsdef** below.

OF ["'*left*'*center*'*right*'"]
Define the odd-page footer, which is formatted just above the normal page footer on odd-numbered pages. See **PF**. **OF** defines the string **EOPof**.

OH ["'*left*'*center*'*right*'"]
Define the odd-page header, which is formatted just below the normal page header on odd-numbered pages. See **PH**. **OH** defines the string **TPoh**.

OP Ensure that subsequent text is formatted at the top of an odd-numbered page; no page break is performed if the drawing position is already there.

P [*type*]
Begin new paragraph. If *type* is missing or **0**, **P** sets the paragraph fully left−aligned. A *type* of **1** indents the first line by **\[Pi]** ens. Set the register **Pt** to select a default paragraph indentation style. The register **Ps** determines the amount of vertical space between paragraphs.

To set a paragraph with a hanging indent, use **VL** with the desired indentation as the argument, **LI** (with no argument), and **LE**.

PE End picture input preprocessed by *pic*(1); see **PS**.

PF ["'*left*'*center*'*right*'"]
Define the page footer. The footer is formatted at the bottom of each page; the argument is otherwise as described in **PH**. **PF** defines the string **EOPf**. See **EF**, **OF**, and **EOP**.

PGFORM [*line-length* [*page-length* [*page-offset* [**1**]]]]
Set line length, page length, and/or page offset. This macro can be used for letterheads and similar. It is normally the first macro call in a file, though it is not necessary. **PGFORM** can be used without arguments to reset everything after a **MOVE** call. A line break is done unless the fourth argument is given. This can be used to avoid the page number on the first page while setting new width and length. (It seems as if this macro sometimes doesn't work too well. Use the command-line arguments to change line length, page length, and page offset instead.)

PGNH Suppress header on the next page. To suppress a header on the first page, call this macro prior to formatting any text in the document. Also see register **N**.

PH ["'*left*'*center*'*right*'"]
Define the page header, formatted at the top of each page, as the argument, where *left*, *center*, and *right* are aligned to the respective locations on the line. A "**%**" character in *arg* is replaced by the page number. If the argument is absent, no page header is set. The default page header is

```
"''- % -''"
```

which centers the page number between hyphens and formats nothing at the upper left and right. Header macros call **PX** (if defined) after formatting the header. **PH** defines the string **TPh**. See **EH**, **OH**, and **TP**.

PIC [**−B**] [**−C**|**−I** *n*|**−L**|**−R**] *file* [*width* [*height*]]
Include PostScript document *file*. The optional **−B** argument draws a box around the picture. The optional **−L**, **−C**, **−R**, and **−I** *n* arguments align the picture or indent it by *n*. By default, the picture is left-aligned. Optional *width* and *height* arguments resize the picture. Use of this macro requires two-pass processing; see **INITR** and *mmroff*(1).

PS Start picture input preprocessed by *pic*(1).

PY As **PE**, but with "flyback", returning the drawing position to where it was prior to the picture. This is a *groff mm* extension.

R [*roman-text* [*previous-font-text*]] ...
Join *roman-text* in roman style with *previous-font-text* in the previous font, without space between the arguments. If no arguments, switch font to roman style.

RB [*roman-text* [*bold-text*]] ...
Join *roman-text* in roman style with *bold-text* in boldface, without space between the arguments.

RD [*prompt* [*div* [*str*]]]
Write *prompt* (if specified, a tab and colon ":" to the standard error stream and read response from standard input stream, optionally into a diversion *div* and/or string *str*. Interpolate the saved text by calling its name like a macro. If *string* is present, the input is also stored in a string of that name.

RF End a bibliographic reference citiation started with **RS**. See **RP**.

RI [*roman-text* [*italic-text*]] . . .
Join *roman-text* in roman style with *italic-text* in italics, without space between the arguments.

RL [*text-indent* [**1**]]
Begin an auto-incrementing numbered list, where item numbers start at one and set between square brackets. A *text-indent* argument overrides register **Li**. A second argument suppresses the vertical space that normally precedes each list item; see register **Lsp**. Vertical space in the amount of **Lsp** precedes the list itself, but does not accumulate with pre-item space when this list is nested in another. **RL** calls **LB**; use **LI** to declare list items, and **LE** to end the list. "RL" is a mnemonic for "reference list", but this macro has no relationship with *mm*'s bibliographic reference list facilities.

RP [*suppress-counter-reset* [*page-ejection-policy*]]
Format a reference list, listing items accumulated within **RS/RF** pairs. The reference counter is reset unless the first argument is **1**. Normally, page breaks occur before and after the references are output; the register **Rpej** configures this behavior, and a second argument overrides its value. **TC** calls **RP** automatically if references have accumulated.

groff mm calls **LB** to format the reference list; the **Rpfmt** string configures the list's appearance. Setting register **Ls** to **0** suppresses spacing after the list. The string **Rp** contains the reference list caption.

RPX This hook macro assumes responsibility for formatting the heading of the reference page that **RP** produces. If not defined (the default), *mm* sets the reference list caption in the string **Rp** after two vees of space, centered in italics, followed by another vee of space.

RS [*reference-string*]
Begin citation of an automatically numbered bibliographic reference entry. *mm* collects input into a reference entry until the document calls **RF**. A subsequent **RP** call emits the accumulated entries. References are numbered starting at 1; the register **:R** stores the most recently assigned number. Interpolate the string **Rf** where the reference mark should be, then place the entry between **RS** and **RF** calls on text lines after the reference mark. The **Rfstyle** register determines whether the mark is bracketed, superscripted, or both. If *reference-string* is specified, *mm* also stores the reference mark in a string of that name.

S [*type-size* [*vertical-spacing*]]
Set type size and vertical spacing. Each argument is a *groff* measurement, using an appropriate scaling unit and an optional **+** or **−** prefix to increment or decrement the current value. An argument of **P** restores the previous value, **C** retains the current value, and **D** requests the default. *mm* treats an empty or omitted argument as **P**.

SA [*mode*]
Set or restore the default enablement of adjustment. Specify **0** or **1** as *mode* to set a document's default explicitly; *groff mm* assumes **1**. Invoke the **na** request as desired to temporarily suspend adjustment. **H** and **HU**, and **SA** when called without a *mode* argument, restore the default adjustment.

SETR *ref-name* [*string*]
Create reference *ref-name*, storing its heading and page numbers. If *string* is present, *groff mm* saves it as an auxiliary datum for retrieval by **GETST**. See **INITR**.

SG [*arg* [**1**]]
Output author signature block(s) in **LT** letters and **MT** memoranda. The format of a signature block depends on the letter or memorandum type. See subsection "Internals" below.

LT letters emit a signature for one author at most (see **WA**). The author's title, if any, is set on the next line, indented to align with the name, except in letter type "SP", where it is set in full capitals (like the name) after a comma and space.

Memorandum type 4 uses no signature block. In other memoranda, each of an author's titles is set on a subsequent line, indented to align with the name. They furthermore encode a secretarial annotation including the location, department, and initials specified in each author's **AU** call, followed by any *arg*, writing it at the left margin preceding the last author's name, or preceding the first if a second **SG** argument is present.

SK [*n*] Skip *n* pages. If *n* is **0** or omitted, *mm* breaks the page if necessary. Otherwise, *mm* prints *n* pages, blank except for any headers and footers.

SM *text* [*post*]
SM *pre text post*
Format *text* at a smaller type size, joined with any specified *pre* and *post* at normal size, without space between the arguments.

SP [*distance*]
Space downward by *distance.* Multiple **SP** calls in sequence produce only the largest of the specified *distances.* **SP** accepts boundary-relative motions specified with a prefixed **|** operator, but not negative ones.

SP has no effect when the drawing position is at the top of the page. Put the dummy character escape sequence **\&** (followed by **\c** if desired to prevent a break) on a text line prior to an **SP** call to overcome this restriction.

TAB Reset tab stops to every 5 ens. The **ta** request customizes them.

TB [*title* [*override* [*flag* [*ref-name*]]]]
Caption a table. Arguments are handled analogously to **EC**. The register **Tb** is the table counter. The string **Captb** precedes the table number and any *title.* *mm* centers table captions irrespective of the alignment of any enclosing display.

The package lists captioned tables in a table of contents (see **TC**) if the Boolean register **Lt** is true. The string **Lt** captions this list.

TC [*slevel* [*spacing* [*tlevel* [*tab* [*h1* [*h2* [*h3* [*h4* [*h5*]]]]]]]]]
Output table of contents. This macro is normally the last called in the document. It flushes any pending displays and, if any references are pending (see **RS**), calls **RP**. It then begins a new page with the contents caption, stored in the string **Captc**, centered at the top. The entries follow after three vees of space. Each entry is a saved section (number and) heading title (see the **Cl** register), along with its associated page number. By default, *mm* indents an entry by an amount corresponding to its heading level and the maximum heading length encountered at that heading level; if defined, the string **Ci** overrides these indentations. Entries at heading levels up to and including *slevel* are preceded by *spacing* vees of space. Entries at heading levels up to and including *tlevel* are followed by a leader and a right-aligned page number. If the Boolean-valued *tab* argument is true, the leader is replaced with horizontal motion in the same amount. For entries above heading level *tlevel*, the page number follows the heading text after a word space. Each argument *h1*...*h5* appears in order on its own line, centered, above the contents caption. Page numbering restarts at 1, in register format "i". If the **Oc** register is true, numbering of these pages is suppressed.

If the document calls **TC** with at most four arguments, *mm* calls the hook macro **TX** prior to formatting the contents caption, and **TY** (if defined) *instead* of formatting the contents caption.

mm then presents lists of figures, tables, equations, and exhibits, in that order. A list appears only if at least one such captioned item is present, and if its corresponding register (**Le**, **Lf**, **Lt**, or **Lx**) is set. **TX***xx* and **TY***xx* macros, where *xx* is "FG", "TB", "EC", or "EX", and strings **Capfg**, **Captb**, **Capec**, and **Capex** analogously configure the output of each captioned list.

TE End *tbl*(1) table. See **TS**.

TH Mark the end of a **TS/TE** table heading. When a table spans multiple pages, the heading repeats at the top of each page. *groff mm* does not implement the **N** argument supported by DWB.

TL [*charging-case-number* [*filing-case-number*]]
Begin document title. *mm* collects input into the title until a subsequent **AU** call and formats it as directed by the **MT** memorandum type or cover page macros. *mm* saves *charging-case-number* and *filing-case-number* for use in all memorandum types except 4.

TM *number* ...
Declare technical memorandum number(s) used with **MT** memoranda.

TP *mm* calls this hook macro in lieu of normal page header layout. The package uses a separate environment to format headers and footers. See **EOP**.

Strings available to TP	
TPh	argument to **PH**
TPeh	argument to **EH**
TPoh	argument to **OH**

TS [**H**] Start *tbl*(1) table. Argument "H" tells *mm* that the table's heading should repeat after page breaks. See **TE**, **TH**, and *tbl*(1).

VERBON [*format* [*type-size* [*font*]]]
Begin display of verbatim content. *format* controls several parameters. Add up the values of desired features; the default is **0**. Further arguments configure the *type-size* in typographical points (on typesetting devices), and the face (*font*). On typesetting devices, the default face is **CR** (Courier roman), a monospaced font: all glyphs are equally wide.

Value	**Effect**
1	Disable the formatter's escape character (\).
2	Vertically space before the display.
4	Vertically space after the display.
8	Number output lines; call formatter's **nm** request with arguments in string **Verbnm**.
16	Indent by the amount stored in register **Verbin**.

VERBOFF
End verbatim display.

VL [*text-indent* [*mark-indent* [**1**]]]
Begin variable-item (or "tagged") list. Each item should supply its own mark, or tag. A *text-indent* argument overrides register **Pi**; *mark-indent* sets the distance from the indentation of the current list to the mark. A third argument suppresses the vertical space that normally precedes each list item; see register **Lsp**. Vertical space in the amount of **Lsp** precedes the list itself, but does not accumulate with pre-item space when this list is nested in another. **RL** calls **LB**; use **LI** to declare list items, and **LE** to end the list.

VM [**−T**] [*top* [*bottom*]]
Configure vertical margins.

Without a **−T** argument, increment the top and bottom margins by *top* and *bottom*, respectively. If absent, set the corresponding margin to zero.

With the **−T** argument, set the top and bottom margins to *top* and *bottom*, respectively. If absent, set the respective margin to the default (top: **7v**, bottom: **6v**).

We recommend defining the hook macros **TP** and/or **EOP** if using **−T** and setting top and/or bottom margin to values less than the default. *groff mm* exposes this undocumented DWB macro to enable greater user control of page layout.

WA [*writer's-name* [*title*]]
: Specify the writer(s) of an **LT** letter. *mm* collects input into the writer's address until the document calls **WE**, then outputs it. You can specify multiple writers with empty **WA/WE** pairs; only the last address is used. The arguments give each writer a name and title.

WC [*format* ...]
: Control width of footnotes and displays.

format	**Effect**
N	equivalent to "**−WF −FF −WD**" (default)
WF	set footnotes at full line length, even in two-column mode
−WF	set footnotes using column line length
FF	apply width of first footnote to encountered to subsequent ones
−FF	footnote width determined by **WF** and **−WF**
WD	set displays at full line length, even in two-column mode
−WD	set displays using column line length

WE
: End the writer's address begun with **WA**.

Strings

Many *mm* strings interpolate predefined, localizable text. We present their contents in quotation marks.

Abstract
: "ABSTRACT"

App
: "APPENDIX"

Apptxt
: stores the *title* argument to the most recent **APP** call.

Aumt
: lists space-separated indices of positional arguments to the **AU** macro that should be suppressed from appearance in the document heading on the first page of **MT** memorandum types 0–3 and 6. By default, it is undefined; all arguments are formatted except the second (author initials).

BU
: interpolates a bullet mark, **\[bu]**.

Capec
: "Equation"

Capex
: "Exhibit"

Capfg
: "Figure"

Captb
: "Table"

Captc
: "CONTENTS"

Ci
: assigns indentation amounts used by each heading level listed in a table of contents, in one-to-one correspondence, overriding those computed by **TC**. Each word must be a horizontal measurement (like "**1i**").

DT
: stores the date or other identifier set by **ND**, if called, and otherwise one constructed using the formatter's date registers; see *groff*(1). The *groff* locale determines its format, but register **Isodate** may override it.

EM
: interpolates an em dash, **\[em]**.

F
: interpolates an automatically numbered footnote mark; the number is used by the next **FS** call without an argument. In *troff* mode, the mark is superscripted; in *nroff* mode, it is surrounded by square brackets.

H1txt
: stores the text of the current first-level heading; **H** and **HU** update it, as does **TC** when processing table of contents entries and captions of displayed figures, tables, equations, and exhibits.

HF
: assigns font identifiers, separated by spaces, to heading levels in one-to-one correspondence. Each identifier may be a font mounting position, font name, or style name. Omitted values are assumed to be 1. The default is "**2 2 2 2 2 2 2 2 2 2 2 2 2 2**", which places all headings in italics. DWB's default was "**3 3 2 2 2 2 2**".

HP assigns type sizes, separated by spaces, to heading levels in one-to-one correspondence. *groff mm* interprets each size in typographical points, translating zero values to **10**. Omitted values are assumed to be 0 (and are translated accordingly). The default is "**0 0 0 0 0 0 0 0 0 0 0 0 0 0**".

Index "INDEX"

Le "LIST OF EQUATIONS"

Letfc "Yours very truly," (see **FC**)

Letapp "APPROVED:" (see **AV**)

LetAT "ATTENTION:" (see **LO**)

LetCN "CONFIDENTIAL" (see **LO**)

Letdate "Date" (see **AV**)

Letns is a group of strings structuring the notations produced by **NS**. If the *code* argument to **NS** has no corresponding string, the notation is included between parentheses, prefixed with **Letns!copy**, and suffixed with **Letns!to**. Observe the spaces after "Copy" and before "to".

NS code	**String**	**Contents**
0	Letns!0	Copy to
1	Letns!1	Copy (with att.) to
2	Letns!2	Copy (without att.) to
3	Letns!3	Att.
4	Letns!4	Atts.
5	Letns!5	Enc.
6	Letns!6	Encs.
7	Letns!7	Under separate cover
8	Letns!8	Letter to
9	Letns!9	Memorandum to
10	Letns!10	Copy (with atts.) to
11	Letns!11	Copy (without atts.) to
12	Letns!12	Abstract Only to
13	Letns!13	Complete Memorandum to
14	Letns!14	CC
—	Letns!copy	Copy *(with trailing space)*
—	Letns!to	to *(note leading space)*

Letnsdef selects the notation format when **NS** is given no argument. The default is **0**.

LetRN "In reference to:" (see **LO**)

LetSA "To Whom It May Concern:" (see **LO**)

LetSJ "SUBJECT:" (see **LO**)

Lf "LIST OF FIGURES"

Lt "LIST OF TABLES"

Lx "LIST OF EXHIBITS"

MO1
. . .
MO12 "January" through "December"

Qrf "See chapter \E*[Qrfh], page \En[Qrfp]." If your document uses multiple heading levels, you might redefine this string to use the word "section" instead of "chapter".

Qrfh stores the heading mark corresponding to the most recent **GETR** call.

Qrfp stores the page number corresponding to the most recent **GETR** call.

Rf interpolates an automatically numbered reference mark; the number is used by the next **RS** call.

Rp "REFERENCES"

Rpfmt specifies the **LB** arguments that *groff mm* uses to format the items in an **RP** reference list. The default is "**\\n[Li] 0 1 0 \& 0**".

Rg interpolates the registered (trade) mark sign.

Sm interpolates the service mark sign.

Tcstatus interpolates an indicator of the **TC** macro's processing status. If **TC** is not operating, it is empty. User-defined **TP** or **EOP** macros might condition page headers or footers on its contents.

Value	**Meaning**
co	Table of contents
fg	List of figures
tb	List of tables
ec	List of equations
ex	List of exhibits
ap	Appendix

Tm interpolates ™, the trade mark sign.

Verbnm supplies argument(s) to the **nm** request employed by the **VERBON** macro. The default is **1**.

Registers

Default register values, where meaningful, are shown in parentheses.

.mgm indicates that *groff mm* is in use (Boolean-valued; **1**).

Aph formats an appendix heading (and title, if supplied); see **APP** (Boolean-valued; **1**).

Au includes supplemental author information (the third and subsequent arguments to **AU**) in memorandum "from" information; see **COVER** and **MT**, and the **Aumt** string (Boolean-valued; **1**).

Cl sets the threshold for inclusion of headings in a table of contents. Headings at levels above this value are excluded; see **H** and **TC**. The **Cl** register controls whether a heading is *saved* for output in the table of contents at the time **H** or **HU** is called; if you change **Cl**'s value immediately prior to calling **TC**, you are unlikely to get the result you want (**2**).

Cp suppresses page breaks before lists of captioned equations, exhibits, figures, and tables, and before an index; see **EC**, **EX**, **FG**, **TB**, and **INDP** (Boolean-valued; **0**).

D produces debugging information for the *mm* package on the standard error stream. A value of 0 outputs nothing; 1 reports formatting progress. Higher values communicate internal state information of increasing verbosity (**0**).

De causes a page break after a floating display is output; see **DF** (Boolean-valued; **0**).

Df configures the behavior of **DF**. The following values are recognized; 4 and 5 do not override the **De** register (**5**).

Value	**Effect**
0	Flush pending displays at the end of each section when section-page numbering is active, otherwise at the end of the document.
1	Flush a pending display on the current page or column if there is enough space, otherwise at the end of the document.
2	Flush one pending display at the top of each page or column.

3	Flush a pending display on the current page or column if there is enough space, otherwise at the top of the next.
4	Flush as many pending displays as possible in a new page or column.
5	Fill columns or pages with flushed displays until none remain.

Ds puts vertical space in the amount of register **Dsp** (if defined) or **Lsp** before and after each static display; see **DS** (Boolean-valued; **1**).

Dsp configures the amount of vertical space placed before and after static displays; see **DS** and register **Ds** (*undefined*).

Ec is an auto-incrementing equation counter; see **EC**.

E determines the font style used by **MT** memoranda, **LT** letters, and the *ms.cov* cover page style to set the date and certain other document data; **0** selects roman, and **1** bold (Boolean-valued; **1**).

Ej sets the threshold for page breaks (ejection) prior to the format of headings. Headings at levels above this value are set on the same page and column as any preceding text if possible; see **H** (**0**).

Eq aligns an equation label to the left of a display instead of the right (Boolean-valued; **0**).

Ex is an auto-incrementing exhibit counter; see **EX**.

Fg is an auto-incrementing figure counter; see **FG**.

Fs is multiplied by register **Lsp** to vertically separate footnotes; see **FS** (**1**).

Ftnum is an auto-incrementing footnote counter; see **FS**. *groff mm* recognizes **:p** as an alias of **Ftnum** for DWB compatibility.

H1
. . .
H14 are auto-incrementing counters corresponding to each heading level; see **H**.

H1dot appends a period to the number of a level one heading; see **H** (Boolean-valued; **1**).

H1h is a copy of register **H1**, but it is incremented just before a page break. This can be useful in hook macros; see **H** and **HX**.

Hb sets the threshold for breaking the line after formatting a heading. Text after headings at levels above this value is set on the same output line if possible. Paragraphs numbered via **nP** or the **Np** register cause a break regardless; see **H** (**2**).

Hc sets the threshold for centering a heading. Headings at levels above this value use the prevailing alignment (that is, they are not centered); see **H** (**0**).

Hi selects an indentation policy for text formatted after headings. It does not affect "run-in" headings. The following values are recognized; see **H** and **P** (**1**).

Value	**Effect**
0	no indentation
1	indent per the paragraph type
2	indent to align with heading title

Hps sets the heading level threshold for application of preceding vertical space; see **H**. Headings at levels above the value in register **Hps** use the amount of space in register **Hps1**; otherwise that in **Hps2**. The value of **Hps** should be strictly greater than that of **Ej** (**1**).

Hps1 configures the amount of vertical space preceding a heading above the **Hps** threshold; see **H** (*troff* devices: **0.5v**; *nroff* devices: **1v**).

Hps2 configures the amount of vertical space preceding a heading within the **Hps** threshold; see **H** (*troff* devices: **1v**; *nroff* devices: **2v**).

Hs sets the heading level threshold for application of succeeding vertical space. If the heading level is greater than **Hs**, the heading is followed by vertical space in the amount of register **Hss**; see **H** (**2**).

Hss is multiplied by register **Lsp** to produce vertical space after headings above the threshold in register **Hs**; see **H** (**1**).

Ht suppresses output of heading level counters above the deepest when a heading mark is formatted; see **H** (Boolean-valued; **0**).

Hu sets the heading level used by unnumbered headings; see **HU** (**2**).

Hy enables automatic hyphenation of words (Boolean-valued; **0**).

Isodate configures the use of ISO 8601 date format; that is, *YYYY-MM-DD* instead of the format specified by the localization file. Call **ND** without arguments after updating its value (Boolean-valued; **0**).

L defines the page length for the document, and must be set from the command line. A scaling unit should be appended. The default is that of the selected *groff* output device.

Le
Lf
Lt
Lx configure the report of lists of equation, figure, table, and exhibit captions, respectively, after a table of contents; see **TC** (all: Boolean-valued; **1**).

Letwam sets the maximum number of input lines permitted in a writer's address; see **WA** and **WE** (**14**).

Li configures the text indentation (in ens) applied to enumerated list types; that is, to items in **AL** and **RL** lists, and in **LB** lists whose *type* argument is greater than zero (**5**).

Ls sets a threshold for placement of vertical space before list items. If the list nesting level is greater than this value, no such spacing occurs; see **LI** (**\n[.R]**).

Lsp configures the base amount of vertical space used for separation in the document. *mm* applies this spacing to many contexts, sometimes with multipliers; see **DS**, **FS**, **H**, **LI**, and **P** (*troff* devices: **0.5v**; *nroff* devices: **1v**).

N configures the header and footer placements used by **PH**. The default footer is empty. If "section-page" numbering is selected, the default header becomes empty and the default footer becomes "*x-y*", where *x* is is the section number (the number of the current first-level heading) and *y* the page number within the section. The following values are recognized; for finer control, see **PH**, **PF**, **EH**, **EF**, **OH**, and **OF**, and registers **Sectf** and **Sectp**. Value 5 is a *groff mm* extension (**0**).

Value	Effect
0	Set header on all pages.
1	Move header to footer on page 1.
2	Omit header on page 1.
3	Use "section-page" numbering style on all pages.
4	Omit header on all pages.
5	Use "section-page" and "section-figure" numbering style on all pages.

Np numbers paragraphs with a leading mark in the format *s.p*, where *s* is is first-level heading number and *p* is the paragraph number, starting at 1; see **H** and **P** (Boolean-valued; **0**).

O defines the page offset of the document, and must be set from the command line. A scaling unit should be appended. The default is **.75i** on terminal devices. On typesetters, it is **.963i** or set to **1i** by the *papersize.tmac* package; see *groff_tmac*(5).

Oc suppresses the appearance of page numbers in the table of contents; see **TC** (Boolean-valued; **0**).

Of selects a separator format within equation, exhibit, figure, and table captions; see **EC**, **EX**, **FG**, and **TB**. The punctuation in the separator is either a dot or the **EM** string. The following values are recognized; the spaces shown are unpaddable (**0**).

Value	Effect
0	". "
1	" — "

P interpolates the current page number; it is the same as register **%** except when "section-page" numbering is enabled.

Pi configures the indentation (in ens) applied to the first line of an unnumbered paragraph that has such (see **P**); and the text indentation of non-enumerated list items; that is, to items in **BL**, **BVL**, **DL**, and **VL** lists (but not **ML**), and in **LB** lists whose *type* argument is zero or less (**5**).

Pgps causes the type size and vertical spacing set by **S** to apply to headers and footers, overriding the **HP** string. If not set, **S** calls affect headers and footers only when followed by **PH**, **PF**, **OH**, **EH**, **OF**, or **EF** calls (Boolean-valued; **1**).

Ps is multiplied by register **Lsp** to vertically separate paragraphs; see **P** (**1**).

Pt determines when a first-line indentation is applied to a paragraph; see **P** (**0**).

Value	Effect
0	never
1	always
2	always, except immediately after **H**, **HU**, **DE**, or **LE**

Ref is used internally to control *mmroff* (1)'s two-pass approach to index and reference management; see **IND**, **INDP**, **INITI**, **INITR**, and **PIC** (Boolean-valued; **0**).

Rfnum is an auto-incrementing reference counter; see **RS**. *groff mm* recognizes **:R** as an alias of **Rfnum** for DWB compatibility.

Rfstyle determines the format of the reference number in the **Rf** string. *groff mm* maps **0** to **1** in *nroff* mode and **2** in *troff* mode (**0**).

Value	Meaning
0	automatic
1	bracketed
2	superscripted
3	bracketed and superscripted

Changes to **Rfstyle**'s value may not take effect until the next heading or paragraphing macro call.

Rpej configures the default page ejection policy for reference pages; see **RP** (**0**).

Value	Effect
0	Break the page before and after the reference list.
1	Suppress page break after the list.
2	Suppress page break before the list.
3	Suppress page breaks before and after the list.

S defines the type size for the document, and must be set from the command line. A scaling unit should be appended; **p** is typical (**10p**).

Sectf selects the "section-figure" numbering style. Its default is **0** unless register **N** is set to **5** at the command line (Boolean-valued).

Sectp selects the "section-page" numbering style. Its default is **0** unless register **N** is set to **3** or **5** at the command line (Boolean-valued).

Si configures the amount of display indentation in ens; see **DS** (**5**).

Tb is an auto-incrementing table counter; see **TB**.

V defines the vertical spacing for the document, and must be set from the command line. A scaling unit should be appended; **p** is typical. The default vertical spacing is 120% of the type size. This register is a *groff mm* extension.

Verbin configures the amount of indentation for verbatim displays when indentation is selected; see **VERBON** (**5n**).

W defines the "width" of the document (that is, the length of an output line with no indentation); it must be set from the command line. A scaling unit should be appended. The default is **6i** or assigned by the *papersize.tmac* package; see *groff_tmac*(5).

Internals

The argument (if any) to a **COVER** or **MT** call determines the file that *groff mm* loads to configure the document's layout; see section "Files" below.

LT's behavior depends on the *style* argument given to it; it calls macros with names suffixed accordingly. It is therefore possible to define additional letter types, either in the territory-specific macro file, or as local additions. **LT** sets the registers **Pt** and **Pi** to 0 and 5, respectively. (DWB used a value of 3 for **Pi**.) It then calls an initialization macro corresponding to the type. Define the following macros to support a new letter type.

let@init_*type*
initializes any registers and other data needed by the letter type.

let@head_*type*
formats the letterhead; it is called instead of the usual page header macro. Its definition should remove the alias **let@header** unless the letterhead is desired on subsequent pages.

let@sg_*type name title n is-final* [*SG-arg* . . .]
is called by **SG** only for **LT** letters; **MT** memoranda implement their own signature processing. *name* and *title* are specified through **WA/WE**. *n* is the index of the *n*th writer, and *is-final* is true for the last writer to be listed. Further **SG** arguments are appended to the signature line.

let@fc_*type closing*
is called by **FC**, and takes the formal closing as its argument.

LO implements letter options. It requires that a string named **Let***type* be defined, where *type* is the letter type. **LO** then assigns its second argument (*value*) to the string **let∗lo−***type*.

Files

/usr/share/groff/1.24.2/tmac/m.tmac
is the *groff* implementation of the memorandum macros.

/usr/share/groff/1.24.2/tmac/mm.tmac
is a wrapper enabling the package to be loaded with the option "**−m mm**".

/usr/share/groff/1.24.2/tmac/refer−mm.tmac
implements *refer*(1) support for *mm*.

/usr/share/groff/1.24.2/tmac/mm/ms.cov
implements an *ms*-like cover page.

/usr/share/groff/1.24.2/tmac/mm/0.MT
implements memorandum types 0–3 and 6.

/usr/share/groff/1.24.2/tmac/mm/4.MT
implements memorandum type 4.

/usr/share/groff/1.24.2/tmac/mm/5.MT
implements memorandum type 5.

/usr/share/groff/1.24.2/tmac/mm/locale
: performs any (further) desired localization if present.

/usr/share/doc/groff/examples/letter.mm
: illustrates features of the **LT** letter formats.

/usr/share/doc/groff/examples/memorandum.mm
: illustrates features of the **MT** memorandum formats.

/usr/share/doc/groff/examples/story.mm
: illustrates use of *mm* for literary purposes.

Authors

Jörgen Hägg of Lund, Sweden, wrote the version of the *mm* macro package distributed with *groff* based on an early version of *groff ms* by James Clark. Werner Lemberg and G. Branden Robinson revised and updated it.

See also

MM − A Macro Package for Generating Documents, the DWB 3.3 *mm* manual, introduces the package but does not document *groff mm* extensions.

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

groff(1), *troff*(1), *tbl*(1), *pic*(1), *eqn*(1), *refer*(1), *groff_mmse*(7)

Namn

groff_mmse – svenska ”memorandum” makro för GNU *roff*

Syntax

groff –mmse [*flaggor* . . .] [*filer* . . .]
groff –m mmse [*flaggor* . . .] [*filer* . . .]

Beskrivning

mmse Ã¤r en svensk variant av *mm*. Alla texter Ã¤r Ã¶versatta. En A4 sida fÃ¥r text som Ã¤r 13 cm bred, 3,5 cm indragning samt Ã¤r 28,5 cm hÃ¶g. Det finns stÃ¶d fÃ¶r brevuppstÃ¤llning enligt svensk standard fÃ¶r vÃ¤nster och hÃ¶gerjusterad text.

COVER kan anvÃ¤nda *se_ms* som argument. Detta ger ett svenskt fÃ¶rsÃ¤ttsblad. Se *groff_mm*(7) fÃ¶r Ã¶vriga detaljer.

Brev

TillgÃ¤ngliga brevtyper:

.LT SVV
VÃ¤nsterstÃ¤lld lÃ¶ptext med adressat i position T0 (vÃ¤nsterstÃ¤llt).

.LT SVH
HÃ¶gerstÃ¤lld lÃ¶ptext med adressat i position T4 (passar fÃ¶nsterkuvert).

FÃ¶ljande extra LO-variabler anvÃ¤nds.

.LO DNAMN *namn*
Anger dokumentets namn.

.LO MDAT *datum*
Mottagarens datum, anges under **Ert datum:** (**LetMDAT**).

.LO BIL *strÃ¤ng*
Anger bilaga, nummer eller strÃ¤ng med **Bilaga** (**LetBIL**) som prefix.

.LO KOMP *text*
Anger kompletteringsuppgift.

.LO DBET *beteckning*
Anger dokumentbeteckning eller dokumentnummer.

.LO BET *beteckning*
Anger beteckning (Ã¤rendebeteckning i form av diarienummer eller liknande).

.LO SIDOR *antal*
Anger totala antalet sidor och skrivs ut efter sidnumret inom parenteser.

Om makrot **.TP** Ã¤r definierat anropas det efter utskrift av brevhuvudet. DÃ¤r lÃ¤gger man lÃ¤mpligen in postadress och annat som brevfot.

Skrivet av

JÃ¶rgen HÃ¤gg, Lund, Sweden

Filer

/usr/share/groff/1.24.2/tmac/mse.tmac

/usr/share/groff/1.24.2/tmac/mm/se_*.cov

Se ocksÃ¥

groff_mm(7)

Name

groff_mom − modern macros for document composition with GNU *roff*

Synopsis

groff −mom [*option* ...] [*file* ...]
groff −m mom [*option* ...] [*file* ...]

Description

mom is a macro set for *groff*, designed primarily to prepare documents for PDF and PostScript output. *mom* provides macros in two categories: typesetting and document processing. The former provide access to *groff*'s typesetting capabilities in ways that are simpler to master than *groff*'s requests and escape sequences. The latter provide highly customizable markup tags that allow the user to design and output professional-looking documents with a minimum of typesetting intervention.

Files processed with *pdfmom*(1) produce PDF documents. The documents include a PDF outline that appears in the navigation pane panel of document viewers, and may contain clickable internal and external links.

Normally, *groff*'s native PDF driver, *gropdf*(1), is used to generate the output. When *pdfmom* is given the "**−T ps**" option, it still produces PDF, but processing is delegated to *pdfroff*, which uses *groff*'s PostScript driver, *grops*(1). Not all PDF features are available when **−T ps** is given; its primary use is to allow processing of files with embedded PostScript images.

Files processed with **groff −mom** (or **−m mom**) format for the device specified with the **−T** option. (In this installation, **ps** is the default output device.)

mom comes with her own comprehensive documentation in HTML. A PDF manual, "Producing PDFs with *groff* and *mom*", discusses preparation of PDF documents with *mom* in detail.

Files

/usr/share/groff/1.24.2/tmac/mom.tmac
is a wrapper enabling the package to be loaded with the option "**−m mom**".

/usr/share/groff/1.24.2/tmac/om.tmac
implements the package.

/usr/share/doc/groff/html/mom/toc.html
is the entry point to the HTML documentation.

/usr/share/doc/groff/pdf/mom−pdf.pdf
is "Producing PDFs with *groff* and *mom*", by Deri James and Peter Schaffter.

/usr/share/doc/groff/examples/mom/.mom*
are examples of *mom* usage.

Reference

Escape sequences

***[<*colorname*>]**
begin using an initialized colour inline

[BCK** *n]**
move backward in a line

***[BOLDER]**
invoke pseudo bold inline (related to macro **.SETBOLDER**)

***[BOLDERX]**
off pseudo bold inline (related to macro **.SETBOLDER**)

[BU** *n]**
move characters pairs closer together inline (related to macro **.KERN**)

***[COND]**
invoke pseudo condensing inline (related to macro **.CONDENSE**)

***[CONDX]**
off pseudo condensing inline (related to macro **.CONDENSE**)

[CONDSUP]**...[CONDSUPX]**
pseudo-condensed superscript

[DOWN** *n]**
temporarily move downward in a line

***[EN–MARK]**
mark initial line of a range of line numbers (for use with line numbered endnotes)

***[EXT]**
invoke pseudo extending inline (related to macro **.EXTEND**)

***[EXTX]**
off pseudo condensing inline (related to macro **.EXTEND**)

[EXTSUP]**...[EXTSUPX]**
pseudo extended superscript

[FU** *n]**
move characters pairs further apart inline (related to macro **.KERN**)

[FWD** *n]**
move forward in a line

***[LEADER]**
insert leaders at the end of a line

***[RULE]**
draw a full measure rule

[SIZE** *n]**
change the point size inline (related to macro **.PT_SIZE**)

***[SLANT]**
invoke pseudo italic inline (related to macro **.SETSLANT**)

***[SLANTX]**
off pseudo italic inline (related to macro **.SETSLANT**)

[ST<n***>]**...***[ST<***n***>X]**
string tabs (mark tab positions inline)

[SUP]**...[SUPX]**
superscript

***[TB+]**
inline escape for **.TN** (*Tab Next*)

[UL]**...[ULX]**
invoke underlining inline (fixed width fonts only)

[UP** *n]**
temporarily move upward in a line

Macros

.AUTOLEAD
set the linespacing relative to the point size

.B_MARGIN
set a bottom margin

.BR break a justified line

.CENTER
set line-by-line quad centre

.CONDENSE
set the amount to pseudo condense

.EL break a line without advancing on the page

.EXTEND
set the amount to pseudo extend

.FALLBACK_FONT
establish a fallback font (for missing fonts)

.FAM alias to **.FAMILY**

.FAMILY *<family>*
set the *family type*

.FT set the font style (roman, italic, etc.)

.HI [*<measure>* **]**
hanging indent

.HY automatic hyphenation on/off

.HY_SET
set automatic hyphenation parameters

.IB [*<left measure>* *<right measure>* **]**
indent both

.IBX [CLEAR]
exit indent both

.IL [*<measure>* **]**
indent left

.ILX [CLEAR]
exit indent left

.IQ [CLEAR]
quit any/all indents

.IR [*<measure>* **]**
indent right

.IRX [CLEAR]
exit indent right

.JUSTIFY
justify text to both margins

.KERN
automatic character pair kerning on/off

.L_MARGIN
set a left margin (page offset)

.LEFT
set line-by-line quad left

.LL set a line length

.LS set a linespacing (leading)

.PAGE
set explicit page dimensions and margins

.PAGEWIDTH
set a custom page width

.PAGELENGTH
set a custom page length

.PAPER *<paper_type>*
set common paper sizes (letter, A4, etc)

.PT_SIZE
set the point size

.QUAD
"justify" text left, centre, or right

.R_MARGIN
set a right margin

.RIGHT
set line-by-line quad right

.SETBOLDER
set the amount of emboldening

.SETSLANT
set the degree of slant

.SPREAD
force justify a line

.SS set the sentence space size

.T_MARGIN
set a top margin

.TI [*<measure>* **]**
temporary left indent

.WS set the minimum word space size

Documentation of details

Details of inline escape sequences in alphabetical order

***[**<*colorname*>**]**
begin using an initialized colour inline

[BCK** *n]**
move backward in a line

***[BOLDER]**
***[BOLDERX]**
Emboldening on/off

***[BOLDER]** begins emboldening type. ***[BOLDERX]** turns the feature off. Both are inline escape sequences; therefore, they should not appear as separate lines, but rather be embedded in text lines, like this:

```
Not \*[BOLDER]everything\*[BOLDERX] is as it seems.
```

Alternatively, if you wanted the whole line emboldened, you should do

```
\*[BOLDER]Not everything is as it seems.\*[BOLDERX]
```

Once ***[BOLDER]** is invoked, it remains in effect until turned off.

Note: If you're using the document processing macros with **.PRINTSTYLE TYPEWRITE**, *mom* ignores ***[BOLDER]** requests.

[BU** *n]**
move characters pairs closer together inline (related to macro **.KERN**)

***[COND]**
***[CONDX]**
Pseudo-condensing on/off

***[COND]** begins pseudo-condensing type. ***[CONDX]** turns the feature off. Both are inline escape sequences; therefore, they should not appear as separate lines, but rather be embedded in text lines, like this:

[COND]Not everything is as it seems.****[CONDX]**

***[COND]** remains in effect until you turn it off with ***[CONDX]**.

IMPORTANT: You must turn ***[COND]** off before making any changes to the point size of your type, either via the **.PT_SIZE** macro or with the **\s** inline escape sequence. If you wish the new point size to be pseudo-condensed, simply reinvoke ***[COND]** afterward. Equally, ***[COND]** must be turned off before changing the condense percentage with **.CONDENSE**.

Note: If you're using the document processing macros with **.PRINTSTYLE TYPEWRITE**, *mom* ignores ***[COND]** requests.

[CONDSUP]**...[CONDSUPX]**
pseudo-condensed superscript

[DOWN** *n]**
temporarily move downward in a line

***[EN–MARK]**
mark initial line of a range of line numbers (for use with line numbered endnotes)

***[EXT]**
***[EXTX]**
Pseudo-extending on/off

***[EXT]** begins pseudo-extending type. ***[EXTX]** turns the feature off. Both are inline escape sequences; therefore, they should not appear as separate lines, but rather be embedded in text lines, like this:

[EXT]Not everything is as it seems.****[EXTX]**

***[EXT]** remains in effect until you turn it off with ***[EXTX]**.

IMPORTANT: You must turn ***[EXT]** off before making any changes to the point size of your type, either via the **.PT_SIZE** macro or with the **\s** inline escape sequence. If you wish the new point size to be *pseudo-extended*, simply reinvoke ***[EXT]** afterward. Equally, ***[EXT]** must be turned off before changing the extend percentage with **.EXTEND**.

Note: If you are using the document processing macros with **.PRINTSTYLE TYPEWRITE**, *mom* ignores ***[EXT]** requests.

[EXTSUP]**...[EXTSUPX]**
pseudo extended superscript

[FU** *n]**
move characters pairs further apart inline (related to macro **.KERN**)

[FWD** *n]**
move forward in a line

***[LEADER]**
insert leaders at the end of a line

***[RULE]**
draw a full measure rule

[SIZE** *n]**
change the point size inline (related to macro **.PT_SIZE**)

***[SLANT]**
***[SLANTX]**
Pseudo italic on/off

***[SLANT]** begins *pseudo-italicizing type*. ***[SLANTX]** turns the feature off. Both are inline escape sequences; therefore, they should not appear as separate lines, but rather be embedded in text lines, like this:

```
Not \*[SLANT]everything\*[SLANTX] is as it seems.
```

Alternatively, if you wanted the whole line *pseudo-italicized*, you'd do

```
\*[SLANT]Not everything is as it seems.\*[SLANTX]
```

Once ***[SLANT]** is invoked, it remains in effect until turned off.

Note: If you're using the document processing macros with **.PRINTSTYLE TYPEWRITE**, *mom* underlines pseudo-italics by default. To change this behaviour, use the special macro **.SLANT_MEANS_SLANT**.

[ST<number***>]**...***[ST<***number***>X]**
Mark positions of string tabs

The *quad* direction must be **LEFT** or **JUSTIFY** (see **.QUAD** and **.JUSTIFY**) or the *no-fill mode* set to **LEFT** in order for these inlines to function properly. Please see *IMPORTANT*, below.

String tabs need to be marked off with inline escape sequences before being set up with the **.ST** macro. Any input line may contain string tab markers. *<number>*, above, means the numeric identifier of the tab.

The following shows a sample input line with string tab markers.

```
\*[ST1]De minimus\*[ST1X]non curat\*[ST2]lex\*[ST2X].
```

String *tab 1* begins at the start of the line and ends after the word *time*. String *tab 2* starts at *good* and ends after *men*. *Inline escape sequences* (e.g., *font* or *point size changes*, or horizontal movements, including padding) are taken into account when *mom* determines the *position* and *length* of *string tabs*.

Up to nineteen string tabs may be marked (not necessarily all on the same line, of course), and they must be numbered between 1 and 19.

Once string tabs have been marked in input lines, they have to be *set* with **.ST**, after which they may be called, by number, with **.TAB**.

Note: Lines with string tabs marked off in them are normal input lines, i.e. they get printed, just like any input line. If you want to set up string tabs without the line printing, use the **.SILENT** macro.

IMPORTANT: Owing to the way *groff* processes input lines and turns them into output lines, it is not possible for *mom* to *guess* the correct starting position of string tabs marked off in lines that are centered or set flush right.

Equally, she cannot guess the starting position if a line is fully justified and broken with **.SPREAD**.

In other words, in order to use string tabs, **LEFT** must be active, or, if **.QUAD LEFT** or **JUSTIFY** are active, the line on which the *string tabs* are marked must be broken *manually* with **.BR** (but not **.SPREAD**).

To circumvent this behaviour, I recommend using the **PAD** to set up string tabs in centered or flush right lines. Say, for example, you want to use a *string tab* to *underscore* the text of a centered line with a rule. Rather than this,

```
.CENTER
\*[ST1]A line of text\ *[ST1X]\c
.EL
.ST 1
.TAB 1
.PT_SIZE 24
.ALD 3p
\*[RULE]
.RLD 3p
.TQ
```

you should do:

```
.QUAD CENTER
.PAD "#\*[ST1]A line of text\ *[ST1X]#"
.EL
.ST 1
.TAB 1
.PT_SIZE 24
.ALD 3p
\" You can't use \*[UP] or \ *[DOWN] with \ *[RULE].
.RLD 3p
.TQ
```

[SUP]**...[SUPX]**
superscript

***[TB+]**
Inline escape for **.TN** (*Tab Next*)

[UL]**...[ULX]**
invoke underlining inline (fixed width fonts only)

[UP** *n]**
temporarily move upward in a line

Details of macros in alphabetical order

.AUTOLEAD
set the linespacing relative to the point size

.B_MARGIN <*bottom margin*>
Bottom Margin

Requires a unit of measure

.B_MARGIN sets a nominal position at the bottom of the page beyond which you don't want your type to go. When the bottom margin is reached, *mom* starts a new page. **.B_MARGIN requires a unit of measure.** Decimal fractions are allowed. To set a nominal bottom margin of 3/4 inch, enter

```
.B_MARGIN .75i
```

Obviously, if you haven't spaced the type on your pages so that the last lines fall perfectly at the bottom margin, the margin will vary from page to page. Usually, but not always, the last line of type that fits on a page before the bottom margin causes mom to start a new page.

Occasionally, owing to a peculiarity in *groff*, an extra line will fall below the nominal bottom margin. If you're using the document processing macros, this is unlikely to happen; the document processing macros are very hard-nosed about aligning bottom margins.

Note: The meaning of **.B_MARGIN** is slightly different when you're using the document processing

macros.

.FALLBACK_FONT <*fallback font*> **[ABORT | WARN]**
Fallback Font

In the event that you pass an invalid argument to **.FAMILY** (i.e. a non-existent *family*), *mom*, by default, uses the *fallback font*, **Courier Medium Roman** (**CR**), in order to continue processing your file.

If you'd prefer another *fallback font*, pass **.FALLBACK_FONT** the full *family+font name* of the *font* you'd like. For example, if you'd rather the *fallback font* were **Times Roman Medium Roman**,

```
.FALLBACK_FONT TR
```

would do the trick.

Mom issues a warning whenever a *font style set* with **.FT** does not exist, either because you haven't registered the style or because the *font style* does not exist in the current *family set* with **.FAMILY**. By default, **mom** then aborts, which allows you to correct the problem.

If you'd prefer that **mom** not abort on non-existent *fonts*, but rather continue processing using a *fallback font*, you can pass **.FALLBACK_FONT** the argument **WARN**, either by itself, or in conjunction with your chosen *fallback font***.**

Some examples of invoking **.FALLBACK_FONT**:

.FALLBACK_FONT WARN
mom will issue a warning whenever you try to access a non-existent *font* but will continue processing your file with the default *fallback font*, **Courier Medium Roman**.

.FALLBACK_FONT TR WARN
mom will issue a warning whenever you try to access a non-existent *font* but will continue processing your file with a *fallback font* of **Times Roman Medium Roman**; additionally, **TR** will be the *fallback font* whenever you try to access a *family* that does not exist.

.FALLBACK_FONT TR ABORT
mom will abort whenever you try to access a non-existent **font**, and will use the *fallback font* **TR** whenever you try to access a *family* that does not exist. If, for some reason, you want to revert to **ABORT**, just enter **".FALLBACK_FONT ABORT"** and *mom* will once again abort on *font errors*.

.FAM <*family*>
Type Family, alias of **.FAMILY**

.FAMILY <*family*>
Type Family, alias of **.FAM**

.FAMILY takes one argument: the name of the *family* you want. *Groff* comes with a small set of basic families, each identified by a 1-, 2- or 3-letter mnemonic. The standard families are:

```
A   = Avant Garde
BM  = Bookman
H   = Helvetica
HN  = Helvetica Narrow
N   = New Century Schoolbook
P   = Palatino
T   = Times Roman
ZCM = Zapf Chancery
```

The argument you pass to **.FAMILY** is the identifier at left, above. For example, if you want **Helvetica**, enter

```
.FAMILY H
```

Note: The font macro (**.FT**) lets you specify both the type *family* and the desired font with a single macro. While this saves a few keystrokes, I recommend using **.FAMILY for** *family*, and **.FT for** *font*, except where doing so is genuinely inconvenient. **ZCM**, for example, only exists in one style: **Italic** (**I**).

Therefore,

```
.FT ZCMI
```

makes more sense than setting the *family* to **ZCM**, then setting the *font* to *I*.

Additional note: If you are running a *groff* version prior to 1.19.2, you must follow all **.FAMILY** requests with a **.FT** request, otherwise *mom* will set all type up to the next **.FT** request in the fallback font.

When running *groff* 1.19.2 or later, and you call the **.FAMILY** macro, *mom remembers* the font style (such as roman or italic) currently in use (if the font style exists in the new *family*) and continues to use the same font style in the new family. For example:

```
.FAMILY BM \" Bookman family
.FT I \" Medium Italic
<some text> \" Bookman Medium Italic
.FAMILY H \" Helvetica family
<more text> \" Helvetica Medium Italic
```

However, if the font style does not exist in the new family, *mom* sets all subsequent type in the fallback font (by default, **Courier Medium Roman**) until she encounters a **.FT** call that's valid for the *family*.

For example, assuming you don't have the font **Medium Condensed Roman** (*mom* extension *CD*) in the *Helvetica family*:

```
.FAMILY UN \" Univers family
.FT CD \" Medium Condensed
<some text> \" Univers Medium Condensed
.FAMILY H \" Helvetica family
<more text> \" Courier Medium Roman!
```

In the above example, you must follow **.FAMILY H** with a **.FT** request that's valid for **Helvetica**.

Please see the Appendices, *Adding fonts to groff*, for information on adding fonts and families to *groff*, as well as to see a list of the extensions *mom* provides to *groff*'s basic **R**, **I**, **B**, **BI** styles.

Suggestion: When adding *families to groff*, I recommend following the established standard for the naming families and fonts. For example, if you add the **Garamond** family, name the font files

```
GARAMONDR
GARAMONDI
GARAMONDB
GARAMONDBI
```

GARAMOND then becomes a valid *family name* you can pass to **.FAMILY**. (You could, of course, shorten **GARAMOND** to just **G**, or **GD**.) **R**, **I**, **B**, and **BI** after **GARAMOND** are the *roman*, *italic*, *bold* and *bold-italic* fonts respectively.

.FONT R | B | BI | *<any other valid font style>*
Alias to **.FT**

.FT R | B | BI | *<any other valid font style>*
Set font

By default, *groff* permits **.FT** to take one of four possible arguments specifying the desired font:

```
R = (Medium) Roman
I = (Medium) Italic
B = Bold (Roman)
BI = Bold Italic
```

For example, if your *family* is **Helvetica**, entering

```
.FT B
```

will give you the *Helvetica bold font*. If your *family* were **Palatino**, you'd get the *Palatino bold font*.

Mom considerably extends the range of arguments you can pass to **.FT**, making it more convenient to add and access fonts of differing weights and shapes within the same family.

Have a look here for a list of the weight/style arguments *mom* allows. Be aware, though, that you must have the fonts, correctly installed and named, in order to use the arguments. (See *Adding fonts to groff* for instructions and information.) Please also read the *ADDITIONAL NOTE* found in the description of the **.FAMILY** macro.

How *mom* reacts to an invalid argument to **.FT** depends on which version of *groff* you're using. If your *groff* version is 1.19.2 or later, *mom* will issue a warning and, depending on how you've set up the fallback font, either continue processing using the fallback font, or abort (allowing you to correct the problem). In earlier versions, *mom* will silently continue processing, using either the fallback font or the font that was in effect prior to the invalid **.FT** call.

.FT will also accept, as an argument, a full *family* and *font name*.

For example,

```
.FT HB
```

will set subsequent type in *Helvetica Bold*.

However, I strongly recommend keeping *family* and *font* separate except where doing so is genuinely inconvenient.

For inline control of *fonts*, see *Inline Escapes*, font control.

.HI [<*measure*> **]**

Hanging indent — the optional argument requires a unit of measure.

A hanging indent looks like this:

```
The thousand injuries of Fortunato I had borne as best I
  could, but when he ventured upon insult, I vowed
  revenge.  You who so well know the nature of my soul
  will not suppose, however, that I gave utterance to a
  threat, at length I would be avenged...
```

The first line of text *hangs* outside the *left margin*.

In order to use *hanging indents*, you must first have a *left indent* active (set with either **.IL** or **.IB**). **Mom** will not hang text outside the *left margin set* with **.L_MARGIN** or outside the *left margin* of a *tab*.

The first time you invoke **.HI**, you must give it a **measure**. If you want the first line of a paragraph to *hang by*, say, *1 pica*, do

```
.IL 1P
.HI 1P
```

Subsequent invocations of **.HI** do not require you to supply a *measure*; *mom* keeps track of the last measure you gave it.

Generally speaking, you should invoke **.HI** immediately prior to the line you want hung (i.e. without any intervening control lines). And because *hanging indents* affect only one line, there's no need to turn them off.

IMPORTANT: Unlike **IL**, **IR** and **IB**, measures given to **.HI** are NOT additive. Each time you pass a measure to **.HI**, the measure is treated literally. *Recipe:* A numbered list using *hanging indents*

Note: mom has macros for setting lists. This recipe exists to demonstrate the use of *hanging indents* only.

```
.PAGE 8.5i 11i 1i 1i 1i 1i
.FAMILY  T
.FT      R
.PT_SIZE 12
.LS      14
.JUSTIFY
.KERN
.SS 0
.IL \w'\0\0.'
.HI \w'\0\0.'
1.\0The most important point to be considered is whether
the answer to the meaning of Life, the Universe, and
Everything really is 42.  We have no one's word on the
subject except Mr. Adams's.
.HI
2.\0If the answer to the meaning of Life, the Universe,
and Everything is indeed 42, what impact does this have on
the politics of representation?  42 is, after all not a
prime number.  Are we to infer that prime numbers don't
deserve equal rights and equal access in the universe?
.HI
3.\0If 42 is deemed non-exclusionary, how do we present
it as the answer and, at the same time, forestall debate
on its exclusionary implications?
```

First, we invoke a left indent with a measure equal to the width of 2 figures spaces plus a period (using the \w inline escape). At this point, the left indent is active; text afterward would normally be indented. However, we invoke a hanging indent of exactly the same width, which hangs the first line (and first line only!) to the left of the indent by the same distance (in this case, that means "out to the left margin"). Because we begin the first line with a number, a period, and a figure space, the actual text (*The most important point...*) starts at exactly the same spot as the indented lines that follow.

Notice that subsequent invocations of **.HI** don't require a *measure* to be given.

Paste the example above into a file and preview it with

pdfmom filename.mom | ps2pdf - filename.pdf

to see hanging indents in action.

.IB [*<left measure>* *<right measure>* **]**

Indent both — the optional argument requires a unit of measure

.IB allows you to set or invoke a left and a right indent at the same time.

At its first invocation, you must supply a measure for both indents; at subsequent invocations when you wish to supply a measure, both must be given again. As with **.IL** and **.IR**, the measures are added to the values previously passed to the macro. Hence, if you wish to change just one of the values, you must give an argument of zero to the other.

A word of advice: If you need to manipulate left and right indents separately, use a combination of **.IL** and **.IR** instead of **.IB**. You'll save yourself a lot of grief.

A *minus sign* may be prepended to the arguments to subtract from their current values. The \w inline escape may be used to specify text-dependent measures, in which case no unit of measure is required. For example,

.IB \w'margarine' \w'jello'

left indents text by the width of the word *margarine* and right indents by the width of *jello*.

Like **.IL** and **.IR**, **.IB** with no argument indents by its last active values. See the brief explanation of

how mom handles indents for more details.

Note: Calling a *tab* (with **.TAB <n>**) automatically cancels any active indents.

Additional note: Invoking **.IB** automatically turns off **.IL** and **.IR**.

.IL [*<measure>* **]**

Indent left — the optional argument requires a unit of measure

.IL indents text from the left margin of the page, or if you're in a *tab*, from the left edge of the *tab*. Once *IL* is on, the *left indent* is applied uniformly to every subsequent line of text, even if you change the line length.

The first time you invoke **.IL**, you must give it a measure. Subsequent invocations with a measure add to the previous measure. A minus sign may be prepended to the argument to subtract from the current measure. The **\w** inline escape may be used to specify a text-dependent measure, in which case no unit of measure is required. For example,

```
.IL \w'margarine'
```

indents text by the width of the word *margarine*.

With no argument, **.IL** indents by its last active value. See the brief explanation of how *mom* handles indents for more details.

Note: Calling a *tab* (with **.TAB <n>**) automatically cancels any active indents.

Additional note: Invoking **.IL** automatically turns off **IB**.

.IQ [*<measure>* **]**

IQ — quit any/all indents

IMPORTANT NOTE: The original macro for quitting all indents was **.IX**. This usage has been deprecated in favour of **IQ**. **.IX** will continue to behave as before, but *mom* will issue a warning to *stderr* indicating that you should update your documents.

As a consequence of this change, **.ILX**, **.IRX** and **.IBX** may now also be invoked as **.ILQ**, **.IRQ** and **.IBQ**. Both forms are acceptable.

Without an argument, the macros to quit indents merely restore your original margins and line length. The measures stored in the indent macros themselves are saved so you can call them again without having to supply a measure.

If you pass these macros the optional argument **CLEAR**, they not only restore your original left margin and line length, but also clear any values associated with a particular indent style. The next time you need an indent of the same style, you have to supply a measure again.

.IQ CLEAR, as you'd suspect, quits and clears the values for all indent styles at once.

.IR [*<measure>* **]**

Indent right — the optional argument requires a unit of measure

.IR indents text from the *right margin* of the page, or if you're in a *tab*, from the end of the *tab*.

The first time you invoke **.IR**, you must give it a measure. Subsequent invocations with a measure add to the previous indent measure. A *minus sign* may be prepended to the argument to subtract from the current indent measure. The \w inline escape may be used to specify a text-dependent measure, in which case no *unit of measure* is required. For example,

```
.IR \w'jello'
```

indents text by the width of the word *jello*.

With no argument, **.IR** indents by its last active value. See the brief explanation of how *mom* handles indents for more details.

Note: Calling a *tab* (with **.TAB <n>**) automatically cancels any active indents.

Additional note: Invoking **.IR** automatically turns off **IB**.

.L_MARGIN <*left margin*>

Left Margin

L_MARGIN establishes the distance from the left edge of the printer sheet at which you want your type to start. It may be used any time, and remains in effect until you enter a new value.

Left indents and tabs are calculated from the value you pass to **.L_MARGIN**, hence it's always a good idea to invoke it before starting any serious typesetting. A unit of measure is required. Decimal fractions are allowed. Therefore, to set the left margin at 3 picas (1/2 inch), you'd enter either

```
.L_MARGIN 3P
```

or

```
.L_MARGIN .5i
```

If you use the macros **.PAGE**, **.PAGEWIDTH** or **.PAPER** without invoking **.L_MARGIN** (either before or afterward), *mom* automatically sets **.L_MARGIN** to *1 inch*.

Note: **.L_MARGIN** behaves in a special way when you're using the document processing macros.

.MCO

Begin multi-column setting.

.MCO (*Multi-Column On*) is the *macro* you use to begin *multi-column setting*. It marks the current baseline as the top of your columns, for use later with **.MCR**. See the introduction to columns for an explanation of *multi-columns* and some sample input.

Note: Do not confuse **.MCO** with the **.COLUMNS** macro in the document processing macros.

.MCR

Once you've turned *multi-columns* on (with **.MCO**), **.MCR**, at any time, returns you to the *top of your columns*.

.MCX [<*distance to advance below longest column*> **]**

Optional argument requires a unit of measure.

Exit multi-columns.

.MCX takes you out of any *tab* you were in (by silently invoking **.TQ**) and advances to the bottom of the longest column.

Without an argument, **.MCX** advances *1 linespace* below the longest column.

Linespace, in this instance, is the leading in effect at the moment **.MCX** is invoked.

If you pass the <*distance*> argument to **.MCX**, it advances *1 linespace* below the longest column (see above) *PLUS* the distance specified by the argument. The argument requires a unit of measure; therefore, to advance an extra 6 points below where **.MCX** would normally place you, you'd enter

```
.MCX 6p
```

Note: If you wish to advance a precise distance below the baseline of the longest column, use **.MCX** with an argument of **0** (zero; no *unit of measure* required) in conjunction with the **.ALD** macro, like this:

```
.MCX 0
.ALD 24p
```

The above advances to precisely *24 points* below the baseline of the longest column.

.NEWPAGE

Whenever you want to start a new page, use **.NEWPAGE**, by itself with no argument. **Mom** will finish up processing the current page and move you to the top of a new one (subject to the top margin set with **.T_MARGIN**).

.PAGE <*width*> **[** <*length*> **[** <*lm*> **[** <*rm*> **[** <*tm*> **[** <*bm*> **]]]]]**

All arguments require a unit of measure

IMPORTANT: If you're using the document processing macros, **.PAGE** must come after **.START**. Otherwise, it should go at the top of a document, prior to any text. And remember, when you're using the document processing macros, top margin and bottom margin mean something slightly different than when you're using just the typesetting macros (see Top and bottom margins in document processing).

.PAGE lets you establish paper dimensions and page margins with a single macro. The only required argument is page width. The rest are optional, but they must appear in order and you can't skip over any. *<lm>*, *<rm>*, *<tm>* and *<bm>* refer to the left, right, top and bottom margins respectively.

Assuming your page dimensions are 11 inches by 17 inches, and that's all you want to set, enter

```
.PAGE 11i 17i
```

If you want to set the left margin as well, say, at 1 inch, **PAGE** would look like this:

```
.PAGE 11i 17i 1i
```

Now suppose you also want to set the top margin, say, at 1–1/2 inches. *<tm>* comes after *<rm>* in the optional arguments, but you can't skip over any arguments, therefore to set the top margin, you must also give a right margin. The **.PAGE** macro would look like this:

```
.PAGE 11i 17i 1i 1i 1.5i
                    |   |
required right---+  +---top margin
        margin
```

Clearly, **.PAGE** is best used when you want a convenient way to tell *mom* just the dimensions of your printer sheet (width and length), or when you want to tell her everything about the page (dimensions and all the margins), for example

```
.PAGE 8.5i 11i 45p 45p 45p 45p
```

This sets up an 8½ by 11 inch page with margins of 45 points (5/8-inch) all around.

Additionally, if you invoke **.PAGE** with a top margin argument, any macros you invoke after **.PAGE** will almost certainly move the baseline of the first line of text down by one linespace. To compensate, do

```
.RLD 1v
```

immediately before entering any text, or, if it's feasible, make **.PAGE** the last macro you invoke prior to entering text.

Please read the *Important note* on page dimensions and papersize for information on ensuring *groff* respects your **.PAGE** dimensions and margins.

.PAGELENGTH *<length of printer sheet>*

tells *mom* how long your printer sheet is. It works just like **.PAGEWIDTH**.

Therefore, to tell *mom* your printer sheet is 11 inches long, you enter

```
.PAGELENGTH 11i
```

Please read the important note on page dimensions and papersize for information on ensuring *groff* respects your *PAGELENGTH*.

.PAGEWIDTH *<width of printer sheet>*

The argument to **.PAGEWIDTH** is the width of your printer sheet.

.PAGEWIDTH requires a unit of measure. Decimal fractions are allowed. Hence, to tell *mom* that the width of your printer sheet is 8½ inches, you enter

```
.PAGEWIDTH 8.5i
```

Please read the Important note on page dimensions and papersize for information on ensuring *groff* respects your *PAGEWIDTH*.

.PAPER *<paper type>*

provides a convenient way to set the page dimensions for some common printer sheet sizes. The argument *<paper type>* can be one of: **LETTER**, **LEGAL**, **STATEMENT**, **TABLOID**, **LEDGER**,

FOLIO, **QUARTO**, **EXECUTIVE**, **10x14**, **A3**, **A4**, **A5**, **B4**, **B5**.

.PRINTSTYLE

.PT_SIZE *<size of type in points>*

Point size of type, does not require a *unit of measure*.

.PT_SIZE (*Point Size*) takes one argument: the *size of type* in *points*. Unlike most other macros that establish the *size* or *measure* of something, **.PT_SIZE** does not require that you supply a *unit of measure* since it's a near universal convention that *type size* is measured in *points*. Therefore, to change the *type size* to, say, *11 points*, enter

```
.PT_SIZE 11
```

Point sizes may be *fractional* (e.g., *10.25* or *12.5*).

You can prepend a *plus* or a *minus sign* to the argument to **.PT_SIZE**, in which case the *point size* will be changed by + or − the original value. For example, if the *point size* is *12*, and you want *14*, you can do

```
.PT_SIZE +2
```

then later reset it to *12* with

```
.PT_SIZE -2
```

The *size of type* can also be changed inline.

Note: It is unfortunate that the **pic** preprocessor has already taken the name, PS, and thus *mom*'s macro for setting *point sizes* can't use it. However, if you aren't using **pic**, you might want to alias **.PT_SIZE** as **.PS**, since there'd be no conflict. For example

```
.ALIAS PS PT_SIZE
```

would allow you to set *point sizes* with **.PS**.

.R_MARGIN *<right margin>*

Right Margin

Requires a unit of measure.

IMPORTANT: **.R_MARGIN**, if used, must come after **.PAPER**, **.PAGEWIDTH**, **.L_MARGIN**, and/or **.PAGE** (if a right margin isn't given to PAGE). The reason is that **.R_MARGIN** calculates line length from the overall page dimensions and the left margin.

Obviously, it can't make the calculation if it doesn't know the page width and the left margin.

.R_MARGIN establishes the amount of space you want between the end of typeset lines and the right hand edge of the printer sheet. In other words, it sets the line length. **.R_MARGIN** requires a unit of measure. Decimal fractions are allowed.

The line length macro (LL) can be used in place of **.R_MARGIN**. In either case, the last one invoked sets the line length. The choice of which to use is up to you. In some instances, you may find it easier to think of a section of type as having a right margin. In others, giving a line length may make more sense.

For example, if you're setting a page of type you know should have 6-pica margins left and right, it makes sense to enter a left and right margin, like this:

```
.L_MARGIN 6P
.R_MARGIN 6P
```

That way, you don't have to worry about calculating the line length. On the other hand, if you know the line length for a patch of type should be 17 picas and 3 points, entering the line length with LL is much easier than calculating the right margin, e.g.,

```
.LL 17P+3p
```

If you use the macros **.PAGE**, **.PAGEWIDTH** or **PAPER** without invoking **.R_MARGIN** afterward, *mom* automatically sets **.R_MARGIN** to *1 inch*. If you set a line length after these macros (with **.LL**), the line length calculated by **.R_MARGIN** is, of course, overridden.

Note: **.R_MARGIN** behaves in a special way when you're using the document processing macros.

.ST <*tab number*> **L | R | C | J [QUAD]**

After *string tabs* have been marked off on an input line (see ***[ST]...*[STX]**), you need to *set* them by giving them a direction and, optionally, the **QUAD** argument.

In this respect, **.ST** is like **.TAB_SET** except that you don't have to give **.ST** an indent or a line length (that's already taken care of, inline, by ***[ST]...*[STX]**).

If you want string *tab 1* to be **left**, enter

```
.ST 1 L
```

If you want it to be *left* and *filled*, enter

```
.ST 1 L QUAD
```

If you want it to be justified, enter

```
.ST 1 J
```

.TAB <*tab number*>

After *tabs* have been defined (either with **.TAB_SET** or **.ST**), **.TAB** moves to whatever *tab number* you pass it as an argument.

For example,

```
.TAB 3
```

moves you to *tab 3*.

Note: **.TAB** breaks the line preceding it and advances 1 linespace. Hence,

```
.TAB 1
A line of text in tab 1.
.TAB 2
A line of text in tab 2.
```

produces, on output

```
A line of text in tab 1.
                                    A line of text in tab 2.
```

If you want the tabs to line up, use **.TN** ("Tab Next") or, more conveniently, the inline escape sequence ***[TB+]**:

```
.TAB 1
A line of text in tab 1.\*[TB+]
A line of text in tab 2.
```

which produces

```
A line of text in tab 1.   A line of text in tab 2.
```

If the text in your tabs runs to several lines, and you want the first lines of each tab to align, you must use the multi-column macros.

Additional note: Any indents in effect prior to calling a tab are automatically turned off by **TAB**. If you were happily zipping down the page with a left indent of *2 picas* turned on, and you call a *tab* whose indent from the left margin is *6 picas*, your new distance from the *left margin* will be *6 picas*, not I 6 picas plus the 2 pica indent.

Tabs are not by nature columnar, which is to say that if the text inside a *tab* runs to several lines, calling another *tab* does not automatically move to the baseline of the first line in the *previous tab*. To demonstrate:

```
TAB 1
Carrots
Potatoes
Broccoli
.TAB 2
$1.99/5 lbs
$0.25/lb
```

```
        $0.99/bunch
```

produces, on output

```
        Carrots
        Potatoes
        Broccoli
                    $1.99/5 lbs
                    $0.25/lb
                    $0.99/bunch
```

.TB *<tab number>*

Alias to **.TAB**

.TI [*<measure>* **]**

Temporary left indent — the optional argument requires a *unit of measure*

A temporary indent is one that applies only to the first line of text that comes after it. Its chief use is indenting the first line of paragraphs. (**Mom's .PP** macro, for example, uses a *temporary indent*.)

The first time you invoke **.TI**, you must give it a measure. If you want to *indent* the first line of a paragraph by, say, 2 ems, do

```
        .TI 2m
```

Subsequent invocations of **.TI** do not require you to supply a measure; *mom* keeps track of the last measure you gave it.

Because *temporary indents* are temporary, there's no need to turn them off.

IMPORTANT: Unlike **.IL**, **.IR** and **IB**, measures given to **.TI** are NOT additive. In the following example, the second **".TI 2P"** is exactly *2 picas*.

```
        .TI 1P
        The beginning of a paragraph. . .
        .TI 2P
        The beginning of another paragraph. . .
```

.TN Tab Next

Inline escape ***[TB+]**

TN moves over to the *next tab* in numeric sequence (*tab n+1*) without advancing on the page. See the *NOTE* in the description of the **.TAB** macro for an example of how **TN** works.

In *tabs* that aren't given the **QUAD** argument when they're set up with **.TAB_SET** or **ST**, you must terminate the line preceding **.TN** with the **\c** inline escape sequence. Conversely, if you did give a **QUAD** argument to **.TAB_SET** or **ST**, the **\c must not be used.**

If you find remembering whether to put in the **\c** bothersome, you may prefer to use the inline escape alternative to **.TN**, ***[TB+]**, which works consistently regardless of the fill mode.

Note: You must put text in the input line immediately after **.TN**. Stacking of **.TN**'s is not allowed. In other words, you cannot do

```
        .TAB 1
        Some text\c
        .TN
        Some more text\c
        .TN
        .TN
        Yet more text
```

The above example, assuming *tabs* numbered from *1* to *4*, should be entered

```
        .TAB 1
        Some text\c
        .TN
```

```
Some more text\c
.TN
\&\c
.TN
Yet more text
```

\& is a zero-width, non-printing character that *groff* recognizes as valid input, hence meets the requirement for input text following **.TN**.

.TQ **TQ** takes you out of whatever *tab* you were in, advances *1 linespace*, and restores the *left margin*, *line length*, *quad direction* and *fill mode* that were in effect prior to invoking any *tabs*.

.T_MARGIN *<top margin>*

Top margin

Requires a unit of measure

.T_MARGIN establishes the distance from the top of the printer sheet at which you want your type to start. It requires a unit of measure, and decimal fractions are allowed. To set a top margin of 2½ centimetres, you'd enter

```
.T_MARGIN 2.5c
```

.T_MARGIN calculates the vertical position of the first line of type on a page by treating the top edge of the printer sheet as a baseline. Therefore,

```
.T_MARGIN 1.5i
```

puts the baseline of the first line of type 1½ inches beneath the top of the page.

Note: **.T_MARGIN** means something slightly different when you're using the document processing macros. See Top and bottom margins in document processing for an explanation.

IMPORTANT: **.T_MARGIN** does two things: it establishes the top margin for pages that come after it and it moves to that position on the current page. Therefore, **.T_MARGIN** should only be used at the top of a file (prior to entering text) or after NEWPAGE, like this:

```
.NEWPAGE
.T_MARGIN 6P
<text>
```

Authors

Peter Schaffter wrote *mom*. Deri James added PDF support. Bernd Warken contributed this man page.

See also

/usr/share/doc/groff/html/mom/toc.html

mom manual (installed version)

mom manual

(latest version)

mom homepage

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

pdfmom(1), *groff*(1), *troff*(1)

Name

groff_ms – GNU *roff* manuscript macro package for formatting documents

Synopsis

groff –ms [*option* ...] [*file* ...]
groff –m ms [*option* ...] [*file* ...]

Description

The GNU implementation of the *ms* macro package is part of the *groff* document formatting system. The *ms* package is suitable for the composition of letters, memoranda, reports, and books.

These *groff* macros support cover page and table of contents generation, automatically numbered headings, several paragraph styles, a variety of text styling options, footnotes, and multi-column page layouts. *ms* supports the *tbl*(1), *eqn*(1), *pic*(1), and *refer*(1) preprocessors for inclusion of tables, mathematical equations, diagrams, and consistently formatted bibliographic citations.

groff ms is mostly compatible with the documented interface and behavior of AT&T Unix Version 7 *ms*. It recreates most extensions from 4.2BSD (Berkeley) and Research Tenth Edition Unix.

Document structure

The *ms* macro package expects a certain amount of structure: a well-formed document contains at least one paragraphing or heading macro call. To compose a simple document from scratch, begin it by calling **LP** or **PP**. Organize longer documents as follows.

Document type
Calling the **RP** macro at the beginning of your document puts the document description (see below) on a cover page. Otherwise, *ms* places this information on the first page, followed immediately by the body text. Some document types found in other *ms* implementations are specific to AT&T or Berkeley, and are not supported in *groff ms*.

Format and layout
By setting registers and strings, you can configure your document's typeface, margins, spacing, headers and footers, and footnote arrangement. See subsection "Document control settings" below.

Document description
A document description consists of any of: a title, one or more authors' names and affiliated institutions, an abstract, and a date or other identifier. See subsection "Document description macros" below.

Body text
The main matter of your document follows its description (if any). *ms* supports highly structured text consisting of paragraphs interspersed with multi-level headings (chapters, sections, subsections, and so forth) and augmented by lists, footnotes, tables, diagrams, and similar material. The preponderance of subsections below covers these matters.

Table of contents
Macros enable the collection of entries for a table of contents (or index) as the material they discuss appears in the document. A macro call at the end of the document emits the collected entries. This material necessarily follows the rest of the text since GNU *troff* is a single-pass formatter; it cannot determine the page number of a division of the text until it has been set and output. Since *ms* output was designed for the production of hard copy, the traditional procedure was to manually relocate the pages containing the table of contents between the cover page and the body text. Today, page resequencing is more often done in the digital domain. An index works similarly, but because it typically needs to be sorted after collection, its preparation requires separate processing.

Document control settings

The following tables list the document control registers, strings, and special characters. For any parameter whose default is unsatisfactory, define it before calling any *ms* macro other than **RP**.

Margin settings

Parameter	Definition	Effective	Default
`\n[PO]`	Page offset (left margin)	next page	`1i (0)`
`\n[LL]`	Line length	next paragraph	`6.5i (65n)`
`\n[LT]`	Title line length	next paragraph	`6.5i (65n)`
`\n[HM]`	Top (header) margin	next page	`1i`
`\n[FM]`	Bottom (footer) margin	next page	`1i`

Titles (headers, footers)

Parameter	Definition	Effective	Default
`\*[LH]`	Left header text	next header	*empty*
`\*[CH]`	Center header text	next header	`-\n[%]-`
`\*[RH]`	Right header text	next header	*empty*
`\*[LF]`	Left footer text	next footer	*empty*
`\*[CF]`	Center footer text	next footer	*empty*
`\*[RF]`	Right footer text	next footer	*empty*

Text settings

Parameter	Definition	Effective	Default
`\n[PS]`	Point (type) size	next paragraph	`10p`
`\n[VS]`	Vertical spacing (leading)	next paragraph	`12p`
`\n[HY]`	Hyphenation mode	next paragraph	`6`
`\*[FAM]`	Font family	next paragraph	`T`

Paragraph settings

Parameter	Definition	Effective	Default
`\n[PI]`	Indentation	next paragraph	`5n`
`\n[PD]`	Paragraph distance (spacing)	next paragraph	`0.3v (1v)`
`\n[QI]`	Quotation indentation	next paragraph	`5n`
`\n[PORPHANS]`	# of initial lines kept	next paragraph	`1`

Heading settings

Parameter	Definition	Effective	Default
`\n[PSINCR]`	Point (type) size increment	next heading	`1p`
`\n[GROWPS]`	Size increase depth limit	next heading	`0`
`\n[HORPHANS]`	# of following lines kept	next heading	`1`
`\*[SN-STYLE]`	Numbering style (alias)	next heading	`\*[SN-DOT]`

***[SN-STYLE]** can alternatively be made an alias of ***[SN-NO-DOT]** with the **als** request.

Footnote settings

Parameter	Definition	Effective	Default
`\n[FI]`	Indentation	next footnote	`2n`
`\n[FF]`	Format	next footnote	`0`
`\n[FPS]`	Point (type) size	next footnote	`\n[PS]-2p`
`\n[FVS]`	Vertical spacing (leading)	next footnote	`\n[FPS]+2p`
`\n[FPD]`	Paragraph distance (spacing)	next footnote	`\n[PD]/2`
`\*[FR]`	Line length ratio	*special*	`11/12`

Display settings

Parameter	Definition	Effective	Default
`\n[DD]`	Display distance (spacing)	*special*	`0.5v (1v)`
`\n[DI]`	Display indentation	*special*	`0.5i`

Other settings

Parameter	Definition	Effective	Default
`\n[MINGW]`	Minimum gutter width	next page	`2n`
`\n[TC-MARGIN]`	TOC page number margin width	next **PX** call	`\w'000'`
`\[TC-LEADER]`	TOC leader character	next **PX** call	`.\h'1m'`

For entries marked "*special*" in the "Effective" column, see the discussion in the applicable section below. The **PO**, **LL**, and **LT** register defaults vary by output device and paper format; the values shown are for typesetters using U.S. letter paper, and then terminals. See section "Paper format" of *groff*(1). The **PD** and **DD** registers use the larger value if the vertical motion quantum of the output device is too coarse for the smaller one; usually, this is the case only for output to terminals. The "gutter" affected by **\n[MINGW]** is the gap between columns in multiple-column page arrangements. The **TC-MARGIN** register and **TC-LEADER** special character affect the formatting of tables of contents assembled by the **XS**, **XA**, and **XE** macros.

Document description macros

Define information describing the document by calling the macros below in the order shown; **DA** or **ND** can be called to set the document date (or other identifier) at any time before (a) the abstract, if present, or (b) its information is required in a header or footer. Use of these macros is optional, except that **TL** is mandatory if any of **RP**, **AU**, **AI**, or **AB** is called, and **AE** is mandatory if **AB** is called.

.RP [**no-repeat-info**] [**no-renumber**]
: Use the "report" (AT&T: "released paper") format for your document, creating a separate cover page. The default arrangement is to place most of the document description (title, author names and institutions, and abstract, but not the date) at the top of the first page. If the optional **no-repeat-info** argument is given, *ms* produces a cover page but does not repeat any of its information subsequently (but see the **DA** macro below regarding the date). Normally, **RP** sets the page number following the cover page to 1. Specifying the optional **no-renumber** argument suppresses this alteration. Optional arguments can occur in any order. *ms* recognizes **no** as a synonym of **no-repeat-info** to maintain AT&T compatibility. Options other than **no** are GNU extensions.

.TL
: Specify the document title. *ms* collects text on input lines following this call into the title until reaching **AU**, **AB**, or a heading or paragraphing macro call.

.AU
: Specify an author's name. *ms* collects text on input lines following this call into the author's name until reaching **AI**, **AB**, another **AU**, or a heading or paragraphing macro call. Call it repeatedly to specify multiple authors.

.AI
: Specify the preceding author's institutional affiliation. An **AU** call is usefully followed by at most one **AI** call; if there are more, the last **AI** call controls. *ms* collects text on input lines following this call into the author's institution until reaching **AU**, **AB**, or a heading or paragraphing macro call.

.DA [*x* ...]
: Typeset the current date, or any arguments *x*, in the center footer, and, if **RP** is also called, left-aligned at the end of the document description on the cover page.

.ND [*x* ...]
: Typeset the current date, or any arguments *x*, if **RP** is also called, left-aligned at the end of the document description on the cover page. This is *groff ms*'s default.

.AB [**no**]
: Begin the abstract. *ms* collects text on input lines following this call into the abstract until reaching an **AE** call. By default, *ms* places the word "ABSTRACT" centered and in italics above the text of the abstract. The optional argument "**no**" suppresses this heading.

.AE
: End the abstract.

Text settings

The **FAM** string, a GNU extension, sets the font family for body text; the default is "**T**". The **PS** and **VS** registers set the type size and vertical spacing (distance between text baselines), respectively. The font family

and type size are ignored on terminals. Set these parameters before the first call of a heading, paragraphing, or (non-date) document description macro to apply them to headers, footers, and (for **FAM**) footnotes.

The **HY** register defines the automatic hyphenation mode used with the **hy** request. Setting **\n[HY]** to **0** disables automatic hyphenation. This is a Research Tenth Edition Unix extension.

Typographical symbols

ms provides a few strings to obtain typographical symbols not easily entered with the keyboard. These and many others are available as special character escape sequences—see *groff_char*(7).

***[−]** Interpolate an em dash.

***[Q]**
***[U]** Interpolate typographer's quotation marks where available, and neutral double quotes otherwise. ***[Q]** is the left quote and ***[U]** the right.

Paragraphs

Paragraphing macros *break*, or terminate, any pending output line so that a new paragraph can begin. Several paragraph types are available, differing in how indentation applies to them: to left, right, or both margins; to the first output line of the paragraph, all output lines, or all but the first. These calls insert vertical space in the amount stored in the **PD** register, except at page or column breaks, or adjacent to displays.

The **PORPHANS** register defines the minimum number of initial lines of any paragraph that must be kept together to avoid isolated lines at the bottom of a page. If a new paragraph starts close to the bottom of a page, and there is insufficient space to accommodate **\n[PORPHANS]** lines before an automatic page break, *groff ms* forces a page break before formatting the paragraph. This is a GNU extension.

.LP Set a paragraph without any (additional) indentation.

.PP Set a paragraph with a first-line left indentation in the amount stored in the **PI** register.

.IP [*mark* [*width*]]
Set a paragraph with a left indentation. The optional *mark* is not indented, is useful in the construction of lists, and is empty by default. *width* overrides the indentation amount in **\n[PI]**; its default unit is "**n**". Once specified, *width* applies to further **IP** calls until specified again or a heading or different paragraphing macro is called.

.QP Set a paragraph indented from both left and right margins by **\n[QI]**.

.QS
.QE Begin (**QS**) and end (**QE**) a region where each paragraph is indented from both margins by **\n[QI]**. The text between **QS** and **QE** can be structured further by use of other paragraphing macros.

.XP Set an "exdented" paragraph—one with a left indentation of **\n[PI]** on every line *except* the first (also known as a hanging indent). This is a Berkeley extension.

Headings

Use headings to create a hierarchical structure for your document. The *ms* macros print headings in **bold** using the same font family and, by default, type size as the body text. Headings are available with and without automatic numbering. Text on input lines following the macro call becomes the heading's title. Call a paragraphing macro to end the heading text and start the section's content.

.NH [*depth*]
Set an automatically numbered heading. *ms* produces a numbered heading in the form $a.b.c...$, to any level desired, with the numbering of each depth increasing automatically and being reset to zero when a more significant depth is increased. "**1**" is the most significant or coarsest division of the document. Only non-zero values are output. If *depth* is omitted, *ms* assumes "**1**". If you specify *depth* such that an ascending gap occurs relative to the previous **NH** call—that is, you "skip a depth", as by "**.NH 1**" and then "**.NH 3**", *groff ms* emits a warning on the standard error stream.

.NH S *heading-depth-index ...*

Alternatively, you can give **NH** a first argument of "**S**", followed by integers to number the heading depths explicitly. Further automatic numbering, if used, resumes using the specified indices as their predecessors. This feature is a Berkeley extension.

After **NH** is called, the assigned number is made available in the strings **SN−DOT** (as it appears in a printed heading with default formatting, followed by a terminating period) and **SN−NO−DOT** (with the terminating period omitted). These, and **SN−STYLE**, are GNU extensions.

You can control the style used to print numbered headings by defining an appropriate alias for the string **SN−STYLE**. By default, ***[SN−STYLE]** is aliased to ***[SN−DOT]**. If you prefer to omit the terminating period from numbers appearing in numbered headings, you may alias it to ***[SN−NO−DOT]**. Any such change in numbering style becomes effective from the next use of **NH** following redefinition of the alias for ***[SN−STYLE]**. The formatted number of the current heading is available in ***[SN]** (a feature first documented by Berkeley); this string facilitates its inclusion in, for example, table captions, equation labels, and **XS/XA/XE** table of contents entries.

.SH [*depth*]

Set an unnumbered heading. The optional *depth* argument is a GNU extension indicating the heading depth corresponding to the *depth* argument of **NH**. It matches the type size at which the heading is set to that of a numbered heading at the same depth when the **\n[GROWPS]** and **\n[PSINCR]** heading size adjustment mechanism is in effect.

The **PSINCR** register defines an increment in type size to be applied to a heading at a lesser depth than that specified in **\n[GROWPS]**. The value of **\n[PSINCR]** should be specified in points with the "**p**" scaling unit and may include a fractional component.

The **GROWPS** register defines the heading depth above which the type size increment set by **\n[PSINCR]** becomes effective. For each heading depth less than the value of **\n[GROWPS]**, the type size is increased by **\n[PSINCR]**. Setting **\n[GROWPS]** to a value less than 2 disables the incremental heading size feature.

In other words, if the value of **GROWPS** register is greater than the *depth* argument to a **NH** or **SH** call, the type size of a heading produced by these macros increases by **\n[PSINCR]** units over **\n[PS]** multiplied by the difference of **\n[GROWPS]** and *depth*. **GROWPS** and **PSINCR** are GNU extensions.

In *groff ms,* the **NH** and **SH** macros consult the **HORPHANS** register to prevent the output of isolated headings at the bottom of a page; it specifies the minimum number of lines of the subsequent paragraph that must be kept on the same page as the heading. If insufficient space remains on the current page to accommodate the heading and this number of lines of paragraph text, *groff ms* forces a page break before setting the heading. Any display macro call or *tbl*, *pic*, or *eqn* region between the heading and the subsequent paragraph suppresses this grouping. This is a GNU extension.

Typeface and decoration

The *ms* macros provide a variety of ways to style text. Attend closely to the ordering of arguments labeled *pre* and *post,* which is not intuitive. Support for *pre* arguments is a GNU extension.

.B [*text* [*post* [*pre*]]]

Style *text* in bold, followed by *post* in the previous font style without intervening space, and preceded by *pre* similarly. Without arguments, *ms* styles subsequent text in bold until the next paragraphing, heading, or no-argument typeface macro call.

.R [*text* [*post* [*pre*]]]

As **B**, but use the roman style (upright text of normal weight) instead of bold. Argument recognition is a GNU extension.

.I [*text* [*post* [*pre*]]]

As **B**, but use an italic or oblique style instead of bold.

.BI [*text* [*post* [*pre*]]]
: As **B**, but use a bold italic or bold oblique style instead of upright bold. This is a Research Tenth Edition Unix extension.

.CW [*text* [*post* [*pre*]]]
: As **B**, but use a constant-width (monospaced) roman typeface instead of bold. This is a Research Tenth Edition Unix extension.

.BX [*text*]
: Typeset *text* and draw a box around it. On terminals, reverse video is used instead. If you want *text* to contain space, use unbreakable space or horizontal motion escape sequences (**\~**, *****_space_, **\^**, **\|**, **\0**, or **\h**).

.UL [*text* [*post*]]
: Typeset *text* with an underline. On terminals, *text* is bracketed with underscores ("_"). *post,* if present, is set after *text* with no intervening space.

.LG
: Set subsequent text in larger type (2 points larger than the current size) until the next type size, paragraphing, or heading macro call. Call the macro multiple times to enlarge the type size further.

.SM
: Set subsequent text in smaller type (2 points smaller than the current size) until the next type size, paragraphing, or heading macro call. Call the macro multiple times to reduce the type size further.

.NL
: Set subsequent text at the normal type size (**\n[PS]**).

When *pre* is used, a hyphenation control escape sequence **\%** that would ordinarily start *text* must start *pre* instead.

groff ms also offers strings to begin and end super- and subscripting. These are GNU extensions.

***{**
***}**
: Begin and end superscripting, respectively.

***<**
***>**
: Begin and end subscripting, respectively.

Indented regions

You can indent a region of text while otherwise formatting it normally. Such indented regions can be nested.

.RS
: Begin a region where headings, paragraphs, and displays are indented (further) by **\n[PI]**.

.RE
: End the (next) most recent indented region.

Keeps, boxed keeps, and displays

On occasion, you may want to *keep* several lines of text, or a region of a document, together on a single page, preventing an automatic page break within certain boundaries. This can cause a page break to occur earlier than it normally would.

You can alternatively specify a *floating keep:* if a keep cannot fit on the current page, *ms* holds it, allowing text following the keep (in the source document) to fill in the remainder of the current page. When the page breaks by reaching its bottom or by **bp** request, *ms* puts the floating keep at the beginning of the next page.

.KS
: Begin a keep.

.KF
: Begin a floating keep.

.KE
: End (floating) keep.

As an alternative to the keep mechanism, the **ne** request forces a page break if there is not at least the amount of vertical space specified in its argument remaining on the page.

A *boxed keep* has a frame drawn around it.

.B1 Begin a keep with a box drawn around it.

.B2 End boxed keep.

Boxed keep macros cause breaks; to box words within a line, recall **BX** in section "Highlighting" above. *ms* draws box lines close to the text they enclose so that they are usable within paragraphs. When boxing entire paragraphs thus, you may improve their appearance by calling **B1** after the first paragraphing macro, and invoking the **sp** request before calling **B2**.

If you want a boxed keep to float, enclose the **B1** and **B2** calls within a pair of **KF** and **KE** calls.

Displays turn off filling; lines of verse or program code are shown with their lines broken as in the source document without requiring **br** requests between lines. Displays can be kept on a single page or allowed to break across pages. The **DS** macro begins a kept display of the layout specified in its first argument; non-kept displays are begun with dedicated macros corresponding to their layout.

.DS L
.LD Begin (**DS**: kept) left-aligned display.

.DS [**I** [*indent*]]
.ID [*indent*]
Begin (**DS**: kept) display indented by *indent* if specified, **\n[DI]** otherwise.

.DS B
.BD Begin (**DS**: kept) block display: the entire display is left-aligned, but indented such that the longest line in the display is centered on the page.

.DS C
.CD Begin (**DS**: kept) centered display: each line in the display is centered.

.DS R
.RD Begin (**DS**: kept) right-aligned display. This is a GNU extension.

.DE End any display.

groff ms inserts the distance stored in **\n[DD]** before and after each pair of display macros; this is a Berkeley extension. This distance replaces any adjacent inter-paragraph distance or subsequent spacing prior to a section heading. The **DI** register is a GNU extension; its value is an indentation applied to displays created with **DS** and **ID** without arguments, to "**.DS I**" without an indentation argument, and to equations set with "**.EQ I**". Changes to either register take effect at the next display boundary.

Tables, figures, equations, and references

ms often sees use with the *tbl*, *pic*, *eqn*, and *refer* preprocessors. *groff ms* applies the **\n[DD]** distance to regions of the document preprocessed with *eqn*, *pic*, and *tbl*. Mark text meant for preprocessors by enclosing it in pairs of tokens as follows, with nothing between the dot and the macro name. Preprocessors match these tokens only at the start of an input line. The formatter interprets them as macro calls.

.TS [**H**]
.TE Demarcate a table to be processed by the *tbl* preprocessor. The optional **H** argument instructs *ms* to repeat table rows (often column headings) at the top of each new page the table spans, if applicable; calling the **TH** macro marks the end of such rows. *tbl*(1) provides a comprehensive reference to the preprocessor and offers examples of its use.

.PS *h v*
.PE
.PF **PS** marks the start of a *pic*(1) preprocessor diagram; either of **PE** or **PF** ends it, the latter with "flyback" to the vertical position at its top. *h* and *v* are the horizontal and vertical dimensions of the picture; *pic* supplies them automatically.

.EQ [*align* [*label*]]

.EN Demarcate mathematics to be processed by the *eqn* preprocessor. *ms* centers the equation by default; *align* can be **C**, **L**, or **I** to (explicitly) center, left-align, or indent it by **\n[DI]**, respectively. *ms* right-aligns any *label*. See *eqn*(1).

.[

.] Demarcate a bibliographic citation to be processed by the *refer* preprocessor. *refer*(1) provides a comprehensive reference to the preprocessor and the format of its bibliographic database.

When *refer* emits collected references (as might be done on a "Works Cited" page), it interpolates the string ***[REFERENCES]** as an unnumbered heading (**.SH**).

Attempting to place a multi-page table inside a keep can lead to unpleasant results, particularly if the *tbl* "**allbox**" option is used.

Footnotes

A footnote is typically anchored to a place in the text with a *mark,* which is a small integer, a symbol, or arbitrary user-specified text.

****** Place an *automatic number,* an automatically generated numeric footnote mark, in the text. Each time this string is interpolated, the number it produces increments by one. Automatic numbers start at 1. This is a Berkeley extension.

Enclose the footnote text in **FS** and **FE** macro calls to set it at the nearest available "foot", or bottom, of a text column or page.

.FS [*mark*]

Begin a footnote. The **FS−MARK** hook (see below) is called with any supplied *mark* argument, which is then also placed at the beginning of the footnote text. If *mark* is omitted, the next pending automatic number enqueued by interpolation of the * string is used, and if none exists, nothing is prefixed.

.FE End footnote text.

groff ms provides a hook macro, **FS−MARK**, for user-determined operations to be performed when the **FS** macro is called. It is passed the same arguments as **FS** itself. By default, this macro has an empty definition. **FS−MARK** is a GNU extension.

Footnote text is formatted as paragraphs are, using analogous parameters. The registers **FI**, **FPD**, **FPS**, and **FVS** correspond to **PI**, **PD**, **PS**, and **VS**, respectively; **FPD**, **FPS**, and **FVS** are GNU extensions.

The **FF** register controls the formatting of automatically numbered footnote paragraphs, and those for which **FS** is given a *mark* argument, at the bottom of a column or page as follows.

0 Set an automatic number, or a specified **FS** *mark* argument, as a superscript (on typesetters) or surrounded by square brackets (on terminals). The footnote paragraph is indented as with **PP** if there is an **FS** argument or an automatic number, and as with **LP** otherwise. This is the default.

1 As **0**, but set *mark* as regular text, and follow an automatic number with a period.

2 As **1**, but without indentation (like **LP**).

3 As **1**, but set the footnote paragraph with *mark* hanging (like **IP**).

Language and localization

groff ms provides several strings that you can customize for your own purposes, or redefine to adapt the macro package to languages other than English. It is already localized for Czech, German, Spanish, French, Italian, Polish, Russian, and Swedish. Load the desired localization macro package after *ms*; see *groff_tmac*(5).

String	Default
`\*[REFERENCES]`	`References`

```
\*[ABSTRACT]     \f[I]ABSTRACT\f[]
\*[TOC]          Table of Contents
\*[MONTH1]       January
\*[MONTH2]       February
\*[MONTH3]       March
\*[MONTH4]       April
\*[MONTH5]       May
\*[MONTH6]       June
\*[MONTH7]       July
\*[MONTH8]       August
\*[MONTH9]       September
\*[MONTH10]      October
\*[MONTH11]      November
\*[MONTH12]      December
```

The default for **ABSTRACT** includes font selection escape sequences to set the word in italics.

Headers and footers

There are multiple ways to produce headers and footers. One is to define the strings **LH**, **CH**, and **RH** to set the left, center, and right headers, respectively; and **LF**, **CF**, and **RF** to set the left, center, and right footers. This approach suffices for documents that do not distinguish odd- and even-numbered pages.

Another method is to call macros that set headers or footers for odd- or even-numbered pages. Each such macro takes a delimited argument separating the left, center, and right header or footer texts from each other. You can replace the neutral apostrophes (') shown below with any character not appearing in the header or footer text. These macros are Berkeley extensions.

.OH '*left*'*center*'*right*'
.OF '*left*'*center*'*right*'
.EH '*left*'*center*'*right*'
.EF '*left*'*center*'*right*'

The **OH** and **EH** macros define headers for odd- (recto) and even-numbered (verso) pages, respectively; the **OF** and **EF** macros define footers for them.

With either method, a percent sign **%** in header or footer text is replaced by the current page number. By default, *ms* places no header on a page numbered "1" (regardless of its number format).

.P1 Typeset the header even on page 1. To be effective, this macro must be called before the header trap is sprung on any page numbered "1". This is a Berkeley extension.

For even greater flexibility, *ms* is designed to permit the redefinition of the macros that are called when formatter traps that ordinarily cause the headers and footers to be output are sprung. **PT** ("page trap") is called by *ms* when the header is to be written, and **BT** ("bottom trap") when the footer is to be. The *groff* page location trap that *ms* sets up to format the header also calls the (normally undefined) **HD** macro after **PT**; you can define **HD** if you need additional processing after setting the header. The **HD** hook is a Berkeley extension. Any such macros you (re)define must implement any desired specialization for odd-, even-, or first numbered pages.

Tab stops

Use the **ta** request to set tab stops as needed.

.TA Reset the tab stops to the *ms* default (every 5 ens). Redefine this macro to create a different set of default tab stops.

Margins

Control margins using the registers summarized in the "Margins" portion of the table in section "Document control settings" above. There is no setting for the right margin; the combination of page offset **\n[PO]** and line length **\n[LL]** determines it.

Multiple columns

ms can set text in as many columns as reasonably fit on the page. The following macros force a page break if a multi-column layout is active when they are called. **\n[MINGW]** is the default minimum gutter width; it is a GNU extension. When multiple columns are in use, keeps and the **HORPHANS** and **PORPHANS** registers work with respect to column breaks instead of page breaks.

.1C Arrange page text in a single column (the default).

.2C Arrange page text in two columns.

.MC [*column-width* [*gutter-width*]]
Arrange page text in multiple columns. If you specify no arguments, it is equivalent to the **2C** macro. Otherwise, *column-width* is the width of each column and *gutter-width* is the minimum distance between columns.

Creating a table of contents

Define an entry to appear in the table of contents by bracketing its text between calls to the **XS** and **XE** macros. A typical application is to call them immediately after **NH** or **SH** and repeat the heading text within them. The **XA** macro, used within **XS/XE** pairs, supplements an entry—for instance, when it requires multiple output lines, whether because a heading is too long to fit or because style dictates that page numbers not be repeated. You may wish to indent the text thus wrapped to correspond to its heading depth; this can be done in the entry text by prefixing it with tabs or horizontal motion escape sequences, or by providing a second argument to the **XA** macro. **XS** and **XA** automatically associate the page number where they are called with the text following them, but they accept arguments to override this behavior. At the end of the document, call **TC** or **PX** to emit the table of contents; **TC** resets the page number to **i** (Roman numeral one), and then calls **PX**. All of these macros are Berkeley extensions.

.XS [*page-number*]
.XA [*page-number* [*indentation*]]
.XE Begin, supplement, and end a table of contents entry. Each entry is associated with *page-number* (otherwise the current page number); a *page-number* of "**no**" prevents a leader and page number from being emitted for that entry. Use of **XA** within **XS/XE** is optional; it can be repeated. If *indentation* is present, a supplemental entry is indented by that amount; ens are assumed if no unit is indicated. Text on input lines between **XS** and **XE** is stored for later recall by **PX**.

.PX [**no**]
Switch to single-column layout. Unless "**no**" is specified, center and interpolate ***[TOC]** in bold and two points larger than the body text. Emit the table of contents entries.

.TC [**no**]
Set the page number to 1, the page number format to lowercase Roman numerals, and call **PX** (with a "**no**" argument, if present).

The remaining features in this subsection are GNU extensions. *groff ms* obviates the need to repeat heading text after **XS** calls. Call **XN** and **XH** after **NH** and **SH**, respectively. Text to be appended to the formatted section heading, but not to appear in the table of contents entry, can follow these calls.

.XN *heading-text*
Format *heading-text* and create a corresponding table of contents entry; the indentation is computed from the *depth* argument of the preceding **NH** call.

.XH *depth heading-text*
As **XN**, but use *depth* to determine the indentation.

groff ms encourages customization of table of contents entry production. (Re-)define any of the following macros as desired.

.XN-REPLACEMENT *heading-text*

.XH−REPLACEMENT *depth heading-text*
: These hook macros implement **XN** and **XH**, and call **XN−INIT** and **XH−INIT**, respectively, then call **XH−UPDATE−TOC** with the arguments given them.

.XH−INIT
.XN−INIT
: These hook macros do nothing by default.

.XH−UPDATE−TOC *depth heading-text*
: Bracket *heading-text* with **XS** and **XE** calls, indenting it by 2 ens per level of *depth* beyond the first.

You can customize the style of the leader that bridges each table of contents entry with its page number; define the **TC−LEADER** special character by using the **char** request. A typical leader combines the dot glyph "**.**" with a horizontal motion escape sequence to spread the dots. The width of the page number field is stored in the **TC−MARGIN** register.

Differences from AT&T *ms*

The *groff ms* macros are an independent reimplementation, using no AT&T code. Since they take advantage of the extended features of *groff*, they cannot be used with AT&T *troff*. *groff ms* supports features described above as Berkeley and Research Tenth Edition Unix extensions, and adds several of its own.

- The internals of *groff ms* differ from those of AT&T *ms*. Documents that depend upon implementation details of AT&T *ms* may not format properly with *groff ms*. Such details include macros whose function was not documented in the AT&T *ms* manual ("Typing Documents on the UNIX System: Using the −ms Macros with Troff and Nroff", M. E. Lesk, Bell Laboratories, 1978).
- The error-handling policy of *groff ms* is to detect and report errors, rather than to ignore them silently.
- Research Tenth Edition Unix supported **P1/P2** macros to bracket code examples; *groff ms* does not.
- *groff ms* does not work in GNU *troff*'s AT&T compatibility mode. If loaded when that mode is enabled, it aborts processing with a diagnostic message.
- Multiple line spacing is not supported. Use a larger vertical spacing instead.
- *groff ms* uses the same header and footer defaults in both *nroff* and *troff* modes as AT&T *ms* does in *troff* mode; AT&T's default in *nroff* mode is to put the date, in U.S. traditional format (e.g., "January 1, 2021"), in the center footer (the **CF** string).
- Many *groff ms* macros, including those for paragraphs, headings, and displays, cause a reset of paragraph rendering parameters, and may change the indentation; they do so not by incrementing or decrementing it, but by setting it absolutely. This can cause problems for documents that define additional macros of their own that manipulate indentation. Use **RS** and **RE** instead of the **in** request.
- AT&T *ms* interpreted the values of the registers **PS** and **VS** in points, and did not support the use of scaling units with them. *groff ms* interprets values of the registers **PS**, **VS**, **FPS**, and **FVS**, equal to or larger than 1,000 (one thousand) as decimal fractions multiplied by 1,000. (Register values are converted to and stored as basic units. See "Measurements" in the *groff* Texinfo manual or in *groff*(7)). This threshold makes use of a scaling unit with these parameters practical for high-resolution devices while preserving backward compatibility. It also permits expression of non-integral type sizes. For example, "**groff −rPS=10.5p**" at the shell prompt is equivalent to placing "**.nr PS 10.5p**" at the beginning of the document.
- AT&T *ms*'s **AU** macro supported arguments whose values were used with some non-**RP** document types; that of *groff ms* does not.
- Right-aligned displays are available. The AT&T *ms* manual observes that "it is tempting to assume that "**.DS R**" will right adjust lines, but it doesn't work". In *groff ms*, it does.
- To make *groff ms* use the default page offset (which also specifies the left margin), the **PO** register must stay undefined until the first *ms* macro is called. This implies that **\n[PO]** should not be used early in the document, unless it is changed also: accessing an undefined register automatically defines it.

- *groff ms* supports the **PN** register, but it is not necessary; you can access the page number via the usual **%** register and invoke the **af** request to assign a different format to it if desired. (If you redefine the *ms* **PT** macro and desire special treatment of certain page numbers—like "**1**"—you may need to handle a non-Arabic page number format, as *groff ms*'s **PT** does; see the macro package source. *groff ms* aliases the **PN** register to **%**.)
- The AT&T *ms* manual documents registers **CW** and **GW** as setting the default column width and "inter-column gap", respectively, and which applied when **MC** was called with fewer than two arguments. *groff ms* instead treats **MC** without arguments as synonymous with **2C**; there is thus no occasion for a default column width register. Further, the **MINGW** register and the second argument to **MC** specify a *minimum* space between columns, not the fixed gutter width of AT&T *ms*.
- The AT&T *ms* manual did not document the **QI** register; Berkeley and *groff ms* do.
- *groff ms* sets the register **GS** to 1; the AT&T *ms* package does not use it. A document can test its value to determine whether it is being formatted with *groff ms* or another implementation.

Unix Version 7 macros unimplemented by *groff ms*

Several macros described in the Unix Version 7 *ms* documentation are unimplemented by *groff ms* because they are specific to the requirements of documents produced internally by Bell Laboratories, some of which also require a glyph for the Bell System logo that *groff* does not support. These macros implemented several document type formats (**EG**, **IM**, **MF**, **MR**, **TM**, **TR**), were meaningful only in conjunction with the use of certain document types (**AT**, **CS**, **CT**, **OK**, **SG**), stored the postal addresses of Bell Labs sites (**HO**, **IH**, **MH**, **PY**, **WH**), or lacked a stable definition over time (**UX**).

Legacy features

groff ms retains some legacy features solely to support formatting of historical documents; contemporary ones should not use them because they can render poorly. See *groff_char*(7) instead.

AT&T *ms* accent mark strings

AT&T *ms* defined accent mark strings as follows.

String	Description
`\*[']`	Apply acute accent to subsequent glyph.
`\*[`]`	Apply grave accent to subsequent glyph.
`\*[:]`	Apply dieresis (umlaut) to subsequent glyph.
`\*[^]`	Apply circumflex accent to subsequent glyph.
`\*[~]`	Apply tilde accent to subsequent glyph.
`\*[C]`	Apply caron to subsequent glyph.
`\*[,]`	Apply cedilla to subsequent glyph.

Berkeley *ms* accent mark and glyph strings

Berkeley *ms* offered an **AM** macro; calling it redefined the AT&T accent mark strings (except for ***C**), applied them to the *preceding* glyph, and defined additional strings, some for spacing glyphs.

.AM Enable alternative accent mark and glyph-producing strings.

String	Description
`\*[']`	Apply acute accent to preceding glyph.
`\*[`]`	Apply grave accent to preceding glyph.
`\*[:]`	Apply dieresis (umlaut) to preceding glyph.
`\*[^]`	Apply circumflex accent to preceding glyph.
`\*[~]`	Apply tilde accent to preceding glyph.
`\*[,]`	Apply cedilla to preceding glyph.
`\*[/]`	Apply stroke (slash) to preceding glyph.
`\*[v]`	Apply caron to preceding glyph.
`\*[_]`	Apply macron to preceding glyph.
`\*[.]`	Apply underdot to preceding glyph.

`\*[o]`	Apply ring accent to preceding glyph.
`\*[?]`	Interpolate inverted question mark.
`\*[!]`	Interpolate inverted exclamation mark.
`\*[8]`	Interpolate small letter sharp s.
`\*[q]`	Interpolate small letter o with hook accent (ogonek).
`\*[3]`	Interpolate small letter yogh.
`\*[d-]`	Interpolate small letter eth.
`\*[D-]`	Interpolate capital letter eth.
`\*[th]`	Interpolate small letter thorn.
`\*[TH]`	Interpolate capital letter thorn.
`\*[ae]`	Interpolate small ae ligature.
`\*[AE]`	Interpolate capital ae ligature.
`\*[oe]`	Interpolate small oe ligature.
`\*[OE]`	Interpolate capital oe ligature.

Naming conventions

External names available to documents that use the *groff ms* macros contain only uppercase letters and digits. The package reserves for internal use the following identifiers:

- those containing the characters ∗, @, and **:**; and
- those containing only uppercase letters and digits.

When selecting a name for your document's own macros, strings, and registers, avoid those reserved by *groff ms* and those defined by GNU *troff*. See *groff*(7) for complete lists thereof.

groff ms organizes most of its internal names into modules. The naming convenion is as follows.

- Names used only within one module are of the form *module*∗*name*.
- Names used outside the module in which they are defined are of the form *module*@*name*.
- Names associated with a particular environment are of the form *environment***:***name*; these are used only within the **par** module.
- *name* does not have a module prefix.
- Names constructed to implement arrays are of the form *array***!***index*.

Files

/usr/share/groff/1.24.2/tmac/s.tmac
: implements the package.

/usr/share/groff/1.24.2/tmac/refer−ms.tmac
: implements *refer*(1) support for *ms*.

/usr/share/groff/1.24.2/tmac/ms.tmac
: is a wrapper enabling the package to be loaded with the option "**−m ms**".

Authors

The GNU version of the *ms* macro package was written by James Clark and contributors. This document was written by Clark, Larry Kollar, and G. Branden Robinson.

See also

A manual is available in source and rendered form. On your system, it may be compressed and/or available in additional formats.

/usr/share/doc/groff/ms.ms
/usr/share/doc/groff/ms.ps
: "Using *groff* with the *ms* Macro Package"; Larry Kollar and G. Branden Robinson.

/usr/share/doc/groff/msboxes.ms
/usr/share/doc/groff/msboxes.pdf
: "Using PDF boxes with *groff* and the *ms* macros"; Deri James. **BOXSTART** and **BOXSTOP** macros are available via the *sboxes* extension package, enabling colored, bordered boxes when the **pdf** output device is used.

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

groff(1), *troff*(1), *tbl*(1), *pic*(1), *eqn*(1), *refer*(1)

Name

groff_rfc1345 − special character names from RFC 1345 and Vim digraphs

Description

The file *rfc1345.tmac* defines special character escape sequences for *groff*(7) based on the glyph mnemonics specified in RFC 1345 and the digraph table of the text editor Vim. Each escape sequence translates to a Unicode code point, and will render correctly if the underlying font is a Unicode font that covers the code point.

For example, "**\[Rx]**" is the "recipe" or "prescription take" symbol, and maps to the code point U+211E. *groff* lets you write it as "**\[u211E]**", but "**\[Rx]**" is more mnemonic.

For a list of the glyph names provided, please see the file *rfc1345.tmac*, which contains definitions of the form

```
.char \[Rx] \[u211E]     \" PRESCRIPTION TAKE
```

where **.char**'s first argument defines a *groff* special character escape sequence with a mnemonic glyph name, its second argument is a special character escape sequence based on the code point, and the comment describes the glyph defined.

The RFC 1345 glyph names cover a wide range of Unicode code points, including supplemental Latin, Greek, Cyrillic, Hebrew, Arabic, Hiragana, Katakana, and Bopomofo letters, punctuation, math notation, currency symbols, industrial and entertainment icons, and box-drawing symbols.

The Vim digraph table is largely a subset of RFC 1345 (being limited to two-character mnemonics), but, as a newer implementation, adds several mnemonics not specified in the RFC (Latin capital and small letters W and Y with grave accent, bullet, horizontal ellipsis, quadruple prime, "approaches the limit", and two mappings each for the Euro and rouble signs). These have also been added to *rfc1345.tmac*.

rfc1345.tmac contains about 1,700 glyph names. It is not an error to load *rfc1345.tmac* if your font does not have all the glyphs, as long as it contains the glyphs that you actually use in your document.

The RFC 1345 mnemonics are not identical in every case to the mappings for special character glyph names that are built in to *groff*; for example, " **\[<<]**" means the "much less than" sign (U+226A) when *rfc1345.tmac* is not loaded and this special character is not otherwise defined by a document or macro package. *rfc1345.tmac* redefines "**\[<<]**" to the "left-pointing double angle quotation mark" (U+00AB). See *groff_char*(7) for the full list of predefined special character escape sequences.

Usage

Load the *rfc1345.tmac* file. This can be done by either adding "**.mso rfc1345.tmac**" to your document before the first use of any of the glyph names the macro file defines, or by using the *troff*(1) option "**−m rfc1345**" from the shell.

Bugs

As the *groff* Texinfo manual notes, "[o]nly the current font is checked for ligatures and kerns; neither special fonts nor entities defined with the **char** request (and its siblings) are taken into account." Many of the characters defined in *rfc1345.tmac* are accented Latin letters, and will be affected by this deficiency, producing subpar typography.

Files

/usr/share/groff/1.24.2/tmac/rfc1345.tmac
: implements the character mappings.

Authors

rfc1345.tmac was contributed by Dorai Sitaram.

See also

RFC 1345, by Keld Simonsen, June 1992.

The Vim digraph table can be listed using the *vim*(1) command "**:help digraph−table**".

groff_char(7)

Name

groff_trace – macros for debugging GNU *roff* documents

Synopsis

groff –m trace [*option* ...] [*file* ...]

Description

trace is a macro package for the *groff* (7) document formatting system, designed as an aid for debugging documents written in its language. It issues a message to the standard error stream upon entry to and exit from each macro call. This can ease the process of isolating errors in macro definitions.

Activate the package by specifying the command-line option "**–m trace**" to the formatter program (often *groff* (1)). You can achieve finer control by including the macro file within the document; invoke the **mso** request, as in "**.mso trace.tmac**". Only macros that are defined after this invocation are traced. If the **trace–full** register is set to a true value, as with the command-line option "**–r trace–full=1**", register and string assignments, along with some other requests, are traced also. If another macro package should be traced as well, specify it after "**–m trace**" on the command line.

The macro file *trace.tmac* is unusual because it does not contain any macros to be called by a user. Instead, *groff*'s macro definition and alteration facilities are wrapped such that they display diagnostic messages.

Limitations

Because *trace.tmac* wraps the **de** request (and its cousins), macro arguments are expanded one level more. This causes problems if an argument uses four or more backslashes to delay interpretation of an escape sequence. For example, the macro call

```
.foo \\\\n[bar]
```

normally passes "\\n[bar]" to macro "foo", but with **de** redefined, it passes "\n[bar]" instead.

The solution to this problem is to use *groff*'s **\E** escape sequence, an escape character that is not interpreted in copy mode.

```
.foo \En[bar]
```

Files

/usr/share/groff/1.24.2/tmac/trace.tmac
implements the package.

Examples

We will illustrate *trace.tmac* using the shell's "here document" feature to supply *groff* with a document on the standard input stream. Since we are interested only in diagnostic messages appearing on the standard error stream, we discard the formatted output by redirecting the standard output stream to */dev/null*.

Observing nested macro calls

Macro calls can be nested, even with themselves. Tracing recurses along with them; this feature can help to detangle complex call stacks.

```
$ cat <<EOF | groff -m trace > /dev/null
.de countdown
. nop \\$1
. nr count (\\$1 - 1)
. if \\n[count] .countdown \\n[count]
..
.countdown 3
blastoff
EOF
 *** .de countdown
 *** de trace enter: .countdown "3"
  *** de trace enter: .countdown "2"
   *** de trace enter: .countdown "1"
   *** trace exit: .countdown "1"
```

```
 *** trace exit: .countdown "2"
*** trace exit: .countdown "3"
```

Tracing with the mso request

Now let us activate tracing within the document, not with a command-line option. We might do so when employing a macro package like *ms* or *mom*, to avoid distraction by traces of macros we didn't write.

```
$ cat <<EOF | groff -ms > /dev/null
.LP
This is my introductory paragraph.
.mso trace.tmac
.de Mymac
..
.Mymac
.PP
Let us review the existing literature.
EOF
 *** .de Mymac
 *** de trace enter: .Mymac
 *** trace exit: .Mymac
```

As tracing was not yet active when the macros "LP" and "PP" were defined (by *s.tmac*), their calls were not traced; contrast with the macro "Mymac".

Authors

trace.tmac was written by James Clark. This document was written by Bernd Warken and G. Branden Robinson.

See also

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

groff(1)
: gives an overview of the *groff* document formatting system.

troff(1)
: supplies details of the **-m** command-line option.

groff_tmac(5)
: offers a survey of *groff* macro packages.

groff(7)
: is a reference manual for the *groff* language.

Name

groff_www – GNU *roff* macros for authoring web pages

Synopsis

groff –m www [*option* . . .] [*file* . . .]

Description

This manual page describes the GNU *www* macro package, which is part of the *groff*(7) document formatting system. This macro file is automatically loaded by the default *troffrc* file when the formatter (usually *groff*(1)) is called with either of the options "**–T html**" or "**–T xhtml**". The HTML output driver *grohtml*(1) is in a beta state; *groff* distributes these macros to encourage its testing.

Macro	Meaning
JOBNAME	split output into multiple files
HX	automatic heading level cut-off
BCL	specify colours on a web page
BGIMG	specify background image
URL	create a URL using two parameters
MTO	create an HTML email address
TAG	generate an HTML name
IMG	include an image file
PIMG	include PNG image
MPIMG	place PNG on the margin and wrap text around it
HnS	begin heading
HnE	end heading
LK	emit automatically collected links
HR	produce a horizontal rule
NHR	suppress automatic generation of rules
HTL	record document title as HTML metadata only
HEAD	add data to <head> block
ULS	unordered list begin
ULE	unordered list end
OLS	ordered list begin
OLE	ordered list end
DLS	definition list begin
DLE	definition list end
LI	insert a list item
DC	generate a drop capital
HTML	transmit raw HTML to output driver
CDS	code example begin
CDE	code example end
ALN	place links on left of main text
LNS	start two-column table with links on the left
LNE	end two-column table started with **.LNS**
LINKSTYLE	initialize default URL attributes

Macros

JOBNAME *file-name-stem*

Split output into multiple HTML files. In *ms* documents, a file is split whenever an **.SH** or "**.NH 1**" call is encountered. Its argument is the file name stem used for output files. This option is equivalent to *grohtml*'s **–j** option.

HX *n* Set section heading level above which *grohtml* is not to automatically generate links to *n*. For example, when used with the *groff_ms*(7) package, "**.HX 2**" causes *grohtml* to generate a list of links for *ms* macro calls "**.NH 1**" and "**.NH 2**" but not for "**.NH 3**", whereas "**.HX 0**" generates no hyperlinks to section headings. Giving *groff* the option "**–P –l**" disables automatic heading linkage.

BCL *foreground background active not-visited visited*
: Configure colours used for page foreground, page background, active hypertext link, hypertext links not yet visited, and visited hypertext links.

BGIMG *file*
: Set the background image to *file.*

URL *url* [*link-text* [*post*]]
: Generate a hyperlink to *url* with optional *link-text* followed immediately by non-linked text *post.*

  If *link-text* is absent, *url* is formatted as the link text. Hyphenation is disabled while formatting a URL; insert explicit break points with the **\:** escape sequence.

MTO *address* [*link-text* [*post*]]
: Generate an email hyperlink to *address* with optional *link-text* followed immediately by non-linked text *post.*

  If *link-text* is absent, *address* is formatted as the link text. Hyphenation is disabled while formatting a URL; insert explicit break points with the **\:** escape sequence.

TAG *name*
: Generate an anchor *name* marking its location in the document as a target for hyperlinks.

IMG *file-name* [**−C**|**−L**|**−R**] [*width* [*height*]]
: Insert graphical image *file-name.* The optional second argument aligns the image to the center (**−C**; default), left (**−L**), or right (**−R**). The optional *width* defaults to 100 units (in the context of the HTML renderer) and the optional *height* to the same measurement as *width* .

PIMG [**−C**|**−L**|**−R**] *file-name* [*width* [*height*]]
: Insert graphical image *file-name,* assumed to be in PNG format. Compared to **IMG**, this macro has the advantage of working with both PostScript and HTML devices since it converts *file-name* into EPS format using *pngtopnm*(1), *pnmcrop*(1), and *pnmtops*(1). *height* and *width* may be given as percentages (of the line length **\n[.l]** and page length **\n[.p]**, respectively).

  If the document isn't processed with "**−T html**" or "**−T xhtml**", you must use *groff*(1)'s **−U** option.

MPIMG [**−L**|**−R**] [**−G** *gap*] *file-name* [*width* [*height*]]
: As **PIMG**, but place the image at a margin and wrap text around it. The image is aligned the to the left (**−L**, default) or right (**−R**) margin. **−G** *gap* imposes horizontal space of *gap* pixels between the picture and the text that wraps around it. The default *gap* is zero.

HnS *n*
: Begin heading at level *n.*

HnE
: End heading text.

LK
: Direct *grohtml* to emit the list of automatically generated hyperlinks at this location.

HR
: Generate a full-width horizontal rule when used with "**−T html**" or "**−T xhtml**".

NHR
: Suppress the horizontal rules at the document's top and bottom that *grohtml* emits by default.

HTL
: Generate an HTML title only. This differs from *groff_ms*(7)'s **TL** macro, which generates both an HTML title and an <H1> heading. Use it to provide an HTML title as document metadata but not as formatted text. The title ends when vertical space (**.sp**) or a break (**.br**) is seen.

HEAD
: Add arbitrary data to the HTML <head> element. Ignored if not processed with "**−T html**" or "**−T xhtml**".

HTML
: All text after this macro is treated as raw HTML. Ignored if not processed with "**−T html**" or "**−T xhtml**".

DC *L text* [*color*]
: Format a "drop cap"—a large character *L* (usually a capital letter) at a larger size and with a lower text baseline than the following *text,* such that the cap-heights of *L* and *text* align, in the specified *color* (default: black).

CDS
: Begin code display; a monospaced font is selected and filling is disabled.

CDE
: End code display.

ALN [*color* [*width-percentage*]]
: Enable a "navigation pane" containing links to section headings, aligned to the left of running text, and configure its rendering parameters. Columnation is achieved with an HTML table. *color* indicates an HTML background color to be used in the navigation column; the default is **#eeeeee**. *width-percentage* allocates the given percentage of the navigation pane's table column; the default is **30**. Call this macro at most once, at the beginning of the document.

LNS
: Begin text indexed by the navigation pane configured by **ALN**.

LNE
: End text indexed by navigation pane started by **LNS**.

LINKSTYLE *color* [*font-style* [*open-glyph close-glyph*]]
: Configure rendering of formatted hyperlink targets for devices other than **html** and **xhtml**; render targets in *color* (default: blue) using the typeface *font-style* (default: **CR**) bracketed with *open-glyph* and *close-glyph* (defaults: **\[la]** and **\[ra]**, respectively).

Section heading links

By default *grohtml* generates links to all section headings and places these at the top of the HTML document. Configure this behavior with **HX** and/or **LK**.

Limitations of *grohtml*

tbl(1) tables are rendered as PNG images.

All URLs currently are treated as consuming no textual space in *groff*. This could be considered as a bug since it causes some problems. To circumvent this, *www.tmac* inserts a zero-width character which expands to a harmless space (only if run with **−Thtml** or **−Txhtml**).

Some of the macros, like **ALN**, **LNS**, and **LNE**, are integrated only with the *groff_ms*(7) package.

Files

/usr/share/groff/1.24.2/tmac/www.tmac

Authors

The *www* macro package was written by Gaius Mulley, with additions by Werner Lemberg and Bernd Warken.

See also

groff(1), *troff*(1), *grohtml*(1), *netpbm*(1)

Name

roff – concepts and history of *roff* typesetting

Description

The term *roff* denotes a family of document formatting systems known by names like *troff*, *nroff*, and *ditroff*. A *roff* system consists of an interpreter for an extensible text formatting language and a set of programs for preparing output for various devices and file formats. Unix-like operating systems often distribute a *roff* system. The manual pages on Unix systems ("man pages") and bestselling books on software engineering, including Brian Kernighan and Dennis Ritchie's *The C Programming Language* and W. Richard Stevens's *Advanced Programming in the Unix Environment*, have been written using *roff* systems. GNU *roff*—*groff*—is arguably the most widespread *roff* implementation.

Below we present typographical concepts foundational to understanding any *roff* implementation, narrate the development history of some *roff* systems, detail the command pipeline managed by *groff*(1), survey the formatting language, suggest tips for editing *roff* input, and recommend further reading materials.

Concepts

AT&T *troff* was designed to take input as it would be composed on a typewriter, including the teletypewriters used as early computer terminals, and relieve the user drafting a document of concern with details like line length maintenance, hyphenation breaking, and consistent paragraph indentation. Early in its development, the program gained the ability to prepare output for a phototypesetter; a document could then be prepared for output to a teletypewriter, a phototypesetter, or both. GNU *troff* continues this tradition of permitting an author to compose a single master version of a document which can then be rendered upon a variety of output formats or devices, including PDF, HTML, laser printers, and terminal displays.

roff input contains text interspersed with instructions to control the formatter. Even in the absence of such instructions, a *roff* formatter still processes its input in several ways, by filling, hyphenating, breaking, and adjusting it, and supplementing it with inter-sentence space. These processes are basic to typesetting, and can be controlled at the input document's discretion.

Instructions to *roff* take two forms. One is the *request*, which occupies an input line and starts with the *control character*, a dot " **.**". The other is the *escape sequence*, which can occur almost anywhere and starts with the *escape character*, a backslash " \". Both characters are configurable.

```
.br \" break the output line
```

When a device-independent *roff* formatter starts up, it obtains information about the device for which it is preparing output from the latter's description file (see *groff_font*(5)). An essential property is the length of the output line, such as "6.5 inches".

The formatter interprets plain text files employing the Unix line-ending convention. It reads input a character at a time, collecting words as it goes, and fits as many words together on an output line as it can—this is known as *filling*. To a *roff* system, a *word* is any sequence of one or more characters that aren't spaces or newlines. The exceptions separate words.

A *roff* formatter attempts to detect boundaries between sentences, and supplies supplemental inter-sentence space between them. It flags certain characters (normally "**!**", "**?**", and "**.**") as potentially ending a sentence. When the formatter encounters one of these *end-of-sentence characters* at the end of an input line, or one of them is followed by two (unescaped) spaces on the same input line, it appends an inter-word space followed by an inter-sentence space in the output. The dummy character escape sequence **\&** can be used after an end-of-sentence character to defeat end-of-sentence detection on a per-instance basis. Normally, the occurrence of a visible non-end-of-sentence character (as opposed to a space or tab) immediately after an end-of-sentence character cancels detection of the end of a sentence. However, the formatter treats several characters *transparently* after the occurrence of an end-of-sentence character—it does not cancel end-of-sentence status upon encountering them. Such characters are often used as footnote marks or to close quotations and parentheticals. The default set is **"**, **'**, **)**, **]**, ∗, **\[dg]**, **\[dd]**, **\[rq]**, and **\[cq]**. The last four are examples of *special characters,* escape sequences whose purpose is to obtain glyphs that are not easily typed at the keyboard, or which have special meaning to the formatter (like **** itself).

When an output line is nearly full, it is uncommon for the next word collected from the input to exactly fill it—often, there is room left over for only part of the next word. *Hyphenation* is the process of splitting a word so that it appears partially on one line, followed by a hyphen to indicate to the reader that the word has been broken, and that its remainder lies on the next. Hyphenation break points can be manually specified; GNU *troff* also uses a hyphenation algorithm and language-specific pattern files (based on TEX's) to decide which words can be hyphenated and where. Hyphenation does not always occur even when the hyphenation rules for a word allow it; it can be disabled, and when not disabled there are several parameters that can prevent it in certain circumstances.

Once an output line is full, the formatter places the next word (or remainder of a hyphenated one) on a different output line; this is called a *break.* In this document and in *roff* discussions generally, a "break" if not further qualified always refers to the termination of an output line. When the formatter is filling text, it introduces breaks automatically to keep output lines from exceeding the configured line length. After an automatic break, the formatter *adjusts* the line if applicable (see below), and then resumes collecting and filling text on the next output line.

Sometimes, a line cannot be broken automatically. This usually does not happen with natural language text unless the output line length has been manipulated to be extremely short, but it can with specialized text like program source code. *groff* provides a means of telling the formatter where the line may be broken without hyphens. This is done with the non-printing break point escape sequence **\:**.

There are several ways to cause a break at a predictable location. A blank input line not only causes a break, but by default it also outputs a one-line vertical space (effectively a blank output line). Macro packages may discourage or disable this "blank line method" of paragraphing in favor of their own macros. A line that begins with one or more spaces causes a break. The spaces are output at the beginning of the next line without being *adjusted* (see below). Again, macro packages may provide other methods of producing indented paragraphs. Trailing spaces on *text lines* (see below) are discarded. The formatter breaks the pending output line without adjustment upon encountering the end of input.

After performing an automatic break, the formatter may then *adjust* the line, widening inter-word spaces until the text reaches the right margin. Extra spaces between words are preserved. Leading and trailing spaces are handled as noted above. You can align text to the left or right margin only, or center it, using *requests.*

A *roff* formatter translates horizontal tab characters, also called simply "tabs", in the input into movements to the next tab stop. Tab stops are by default located every half inch measured from the drawing position corresponding to the beginning of the input line; see subsection "Page geometry" below. With them, simple tables can be made. However, this method can be deceptive, as the appearance (and width) of the text in an editor and the results from the formatter can vary greatly, particularly when proportional typefaces are used. A tab character does not cause a break and therefore does not interrupt filling. The formatter provides facilities for sophisticated table composition; there are many details to track when using the "tab" and "field" low-level features, so most users turn to the *tbl*(1) preprocessor to lay out tables.

Requests and macros

A *request* is an instruction to the formatter that occurs after a *control character,* which is recognized at the beginning of an input line. The regular control character is a dot "**.**". Its counterpart, the *no-break control character,* a neutral apostrophe "**'**", suppresses the break implied by some requests. These characters were chosen because it is uncommon for lines of text in natural languages to begin with them. If you require a formatted period or apostrophe (closing single quotation mark) where the formatter expects a control character, prefix the dot or neutral apostrophe with the dummy character escape sequence, "**\&**".

An input line beginning with a control character is called a *control line.* Every line of input that is not a control line is a *text line.*

Requests often take *arguments,* words (separated from the request name and each other by spaces) that specify details of the action you expect the formatter to perform. If a request is meaningless without arguments, it is typically ignored. Requests and escape sequences comprise the control language of the formatter. Of key importance are the requests that define macros. Macros are invoked like requests, enabling the request repertoire to be extended or overridden. (Argument handling in macros is more flexible but also more complex.)

A *macro* can be thought of as an abbreviation you can define for a collection of control and text lines. When the macro is *called* by giving its name after a control character, it is replaced with what it stands for. The process of textual replacement is known as *interpolation.* Interpolations are handled as soon as they are recognized, and once performed, a *roff* formatter scans the replacement for further requests, macro calls, and escape sequences. In *roff* systems, the "**de**" request defines a macro.

Page geometry

roff systems format text under certain assumptions about the size of the output medium, or page. For the formatter to correctly break a line it is filling, it must know the line length, which it derives from the page width. For it to decide whether to write an output line to the current page or wait until the next one, it must know the page length. A device's *resolution* converts practical units like inches or centimeters to *basic units,* a convenient length measure for the output device or file format. The formatter and output driver use basic units to reckon page measurements. The device description file defines its resolution and page dimensions (see *groff_font*(5)).

A *page* is a two-dimensional structure upon which a *roff* system imposes a rectangular coordinate system with its its origin near the upper left corner. Coordinate values are in basic units and increase down and to the right. Useful ones are typically positive and within numeric ranges corresponding to the page boundaries.

Text is arranged on a one-dimensional lattice of text baselines from the top to the bottom of the page. A *text baseline* is a (usually invisible) line upon which the glyphs of a typeface are aligned. *Vertical spacing* is the distance between adjacent text baselines. Typographic tradition sets this quantity to 120% of the type size. The initial vertical drawing position is one unit of vertical spacing below the page top. Typographers term this unit a *vee.*

While the formatter (and, later, output driver) is processing a page, it keeps track of its *drawing position,* which is the location at which the next glyph will be written, from which the next motion will be measured, or where a geometric object will commence rendering. Notionally, glyphs are drawn from the text baseline upward and to the right. (*groff* does not yet support right-to-left scripts.) A glyph therefore "starts" at its bottom-left corner. The formatter's origin is one vee below the page top to prevent a glyph from lying partially or wholly off the page.

Further, it is conventional not to write or draw at the extreme edges of the page. Typesetters configure a *page offset,* a rightward shift from the left edge that defines the zero point from which the formatter reckons the line indentation and length. (*groff*'s terminal output devices have page offsets of zero.)

Combining the foregoing facts results in an origin that lies at the page offset in the horizontal dimension and at the text baseline (using the default vertical spacing) in the vertical dimension. A document can change these prior to its first written or drawn output.

Vertical spacing has an impact on page-breaking decisions. Generally, when a break occurs, the formatter automatically moves the drawing position to the next text baseline. If the formatter were already writing to the last line that fits on the page, advancing by one vee would place the next text baseline off the page. To avoid that, *roff* formatters instruct the output driver to eject the page, start a new one, and again place the drawing position at the page offset one vee below the page top; this is a *page break.*

When the last line of input text corresponds to the last output line that fits on the page, the break caused by the end of input also breaks the page, producing a useless blank one. Macro packages keep users from having to confront this difficulty by setting "traps"; moreover, all but the simplest page layouts tend to have headers and footers, or at least bear vertical margins of at least one vee.

Other language elements

Escape sequences start with the *escape character,* a backslash ****, and are followed by at least one additional character. They can appear anywhere in the input.

With requests, the escape and control characters can be changed; further, escape sequence recognition can be turned off and back on.

Strings store character sequences. In *groff*, they can be parameterized (given arguments) as macros can.

Registers store numerical values, including measurements. The latter are generally in basic units; *scaling units* can be appended to numeric expressions to clarify their meaning when stored or interpolated. Each register can be assigned a *format,* causing its value to interpolate with leading zeroes, in Roman numerals, or alphabetically. Some read-only registers are string-valued, meaning that they interpolate text and lack a format.

Fonts are identified either by a name or by a *mounting position* (a non-negative number). Four styles are available on all devices. **R** is "roman": normal, upright text. **B** is **bold**, an upright typeface with a heavier weight. **I** is *italic*, a face that is oblique on typesetter output devices and usually underlined instead on terminal devices. **BI** is ***bold-italic***, combining both of the foregoing style variations. Typesetting devices group these four styles into *families* of text fonts; they also typically offer one or more *special* fonts that provide unstyled glyphs; see *groff_char*(7).

groff supports named *colors* for glyph rendering and drawing of geometric objects. Stroke and fill colors are distinct; the stroke color is used for glyphs.

Glyphs are visual representation forms of *characters.* In *groff,* the distinction between those two elements is not always obvious (and a full discussion is beyond our scope). In brief, "A" is a character when we consider it in the abstract: to make it a glyph, we must select a typeface with which to render it, and determine its type size and color. The formatting process turns input characters into output glyphs. A few characters commonly seen on keyboards are treated specially by the *roff* language and may not look correct in output if used unthinkingly; they are the (double) quotation mark (`"`), the neutral apostrophe (`'`), the minus sign (`-`), the backslash (`\`), the caret or circumflex accent (`^`), the grave accent (`` ` ``), and the tilde (`~`). All of these and more can be produced with *special character* escape sequences; see *groff_char*(7).

groff offers *streams*, identifiers for writable files, but for security reasons this feature is disabled by default.

A further few language elements arise as page layouts become more sophisticated and demanding. *Environments* collect formatting parameters like line length and typeface. A *diversion* stores formatted output for later use. A *trap* is a condition on the input or output, tested automatically by the formatter, that is associated with a macro: fulfilling the condition *springs* the trap—calls the macro.

Footnote support often exercises all three of the foregoing features. A simple implementation might work as follows. The author writes a pair of macros: one starts a footnote and the other ends it. They further set a trap a small distance above the page bottom, reserving a footnote area. The author calls the first macro where a footnote mark is desired. The macro establishes a diversion so that the footnote text is collected at the place in the body text where its corresponding mark appears. It further creates an environment for the footnote so that it sets at a smaller typeface. The footnote text is formatted in the diversion using that environment, but it does not yet appear in the output. The document author calls the footnote end macro, which returns to the previous environment and ends the diversion. Later, after body text nearly fills the page, the trap springs. The macro called by the trap draws a line across the page and emits the stored diversion by calling it like a macro. Thus, the footnote is rendered.

History

Computer-driven document formatting dates back to the 1960s. The *roff* system is intimately connected with Unix, but its origins lie with the earlier operating systems CTSS, GECOS, and Multics.

The predecessor—*RUNOFF*

roff's ancestor *RUNOFF* was written in the MAD language by Jerry Saltzer to prepare his Ph.D. thesis on the Compatible Time Sharing System (CTSS), a project of the Massachusetts Institute of Technology (MIT). This program is referred to in full capitals, both to distinguish it from its many descendants, and because bits were expensive in those days; five- and six-bit character encodings were still in widespread usage, and mixed-case alphabetics in file names seen as a luxury. *RUNOFF* introduced a syntax of inlining formatting directives amid document text, by beginning a line with a period (an unlikely occurrence in human-readable material) followed by a "control word". Control words with obvious meaning like ".line length *n*" were supported as well as an abbreviation system; the latter came to overwhelm the former in popular usage and later derivatives of the program. A sample of control words from a *RUNOFF* manual of December 1966 was documented as follows (with the parameter notation slightly altered). The abbreviations will be familiar to *roff* veterans.

Abbreviation	Control word
.ad	.adjust
.bp	.begin page
.br	.break
.ce	.center
.in	.indent *n*
.ll	.line length *n*
.nf	.nofill
.pl	.paper length *n*
.sp	.space [*n*]

In 1965, MIT's Project MAC teamed with Bell Telephone Laboratories and General Electric (GE) to inaugurate the Multics project. After a few years, Bell Labs discontinued its participation in Multics, famously prompting the development of Unix. Meanwhile, Saltzer's *RUNOFF* proved influential, seeing many ports and derivations elsewhere.

In 1969, Doug McIlroy wrote one such reimplementation, adding extensions, in the BCPL language for a GE 645 running GECOS at the Bell Labs location in Murray Hill, New Jersey. In its manual, the control commands were termed "requests", their two-letter names were canonical, and the control character was configurable with a **.cc** request. Other familiar requests emerged at this time; no-adjust (**.na**), need (**.ne**), page offset (**.po**), tab configuration (**.ta**, though it worked differently), temporary indent (**.ti**), character translation (**.tr**), and automatic underlining (**.ul**; on *RUNOFF* you had to backspace and underscore in the input yourself). **.fi** to enable filling of output lines got the name it retains to this day. McIlroy's program also featured a heuristic system for automatically placing hyphenation points, designed and implemented by Molly Wagner. It furthermore introduced numeric variables, termed registers. By 1971, this program had been ported to Multics and was known as *roff*, a name McIlroy attributes to Bob Morris, to distinguish it from CTSS *RUNOFF*.

Unix and *roff*

McIlroy's *roff* was one of the first Unix programs. In Ritchie's term, it was "transliterated" from BCPL to DEC PDP-7 assembly language for the fledgling Unix operating system. Automatic hyphenation was managed with **.hc** and **.hy** requests, line spacing control was generalized with the **.ls** request, and what later *roff*s would call diversions were available via "footnote" requests. This *roff* indirectly funded operating systems research at Murray Hill; AT&T prepared patent applications to the U.S. government with it. This arrangement enabled the group to acquire a PDP-11; *roff* promptly proved equal to the task of formatting the manual for what would become known as "First Edition Unix", dated November 1971.

Output from all of the foregoing programs was limited to line printers and paper terminals such as the IBM 2471 (based on the Selectric line of typewriters) and the Teletype Corporation Model 37. Proportionally spaced type was unavailable.

New *roff* and Typesetter *roff*

The first years of Unix were spent in rapid evolution. The practicalities of preparing standardized documents like patent applications (and Unix manual pages), combined with McIlroy's enthusiasm for macro languages, perhaps created an irresistible pressure to make *roff* extensible. Joe Ossanna's *nroff*, literally a "new roff", was the outlet for this pressure. By the time of Unix Version 3 (February 1973)—and still in PDP-11 assembly language—it sported a swath of features now considered essential to *roff* systems: definition of macros (**.de**), diversion of text thither (**.di**), and removal thereof (**.rm**); trap planting (**.wh**; "when") and relocation (**.ch**; "change"); conditional processing (**.if**); and environments (**.ev**). Incremental improvements included assignment of the next page number (**.pn**); no-space mode (**.ns**) and restoration of vertical spacing (**.rs**); the saving (**.sv**) and output (**.os**) of vertical space; specification of replacement characters for tabs (**.tc**) and leaders (**.lc**); configuration of the no-break control character (**.c2**); shorthand to disable automatic hyphenation (**.nh**); a condensation of what were formerly six different requests for configuration of page "titles" (headers and footers) into one (**.tl**) with a length controlled separately from the line length (**.lt**); automatic line numbering (**.nm**); interactive input (**.rd**), which necessitated buffer-flushing (**.fl**), and was made convenient with early program cessation (**.ex**); source file inclusion in its modern form (**.so**; though *RUNOFF* had an ".append" control word for a similar purpose) and early advance to the next file argument (**.nx**); ignorable content (**.ig**); and

programmable abort (**.ab**).

Third Edition Unix also brought the *pipe*(2) system call, the explosive growth of a componentized system based around it, and a "filter model" that remains perceptible today. Equally importantly, the Bell Labs site in Murray Hill acquired a Graphic Systems C/A/T phototypesetter, and with it came the necessity of expanding the capabilities of a *roff* system to cope with a variety of proportionally spaced typefaces at multiple sizes. Ossanna wrote a parallel implementation of *nroff* for the C/A/T, dubbing it *troff* (for "typesetter roff"). Unfortunately, surviving documentation does not illustrate what requests were implemented at this time for C/A/T support; the *troff* (1) man page in Fourth Edition Unix (November 1973) does not feature a request list, unlike *nroff* (1). Apart from typesetter-driven features, Unix Version 4 *roff*s added string definitions (**.ds**); made the escape character configurable (**.ec**); and enabled the user to write diagnostics to the standard error stream (**.tm**). Around 1974, empowered with multiple type sizes, italics, and a symbol font specially commissioned by Bell Labs from Graphic Systems, Kernighan and Lorinda Cherry implemented *eqn* for typesetting mathematics. In the same year, for Fifth Edition Unix, Ossanna combined and reimplemented the two *roff*s in C, using that language's preprocessor to generate both from a single source tree.

Ossanna documented the syntax of the input language to the *nroff* and *troff* programs in the "Troff User's Manual", first published in 1976, with further revisions as late as 1992 by Kernighan. (The original version was entitled "Nroff/Troff User's Manual", which may partially explain why *roff* practitioners have tended to refer to it by its AT&T document identifier, "CSTR #54".) Its final revision serves as the *de facto* specification of AT&T *troff*, and all subsequent implementors of *roff* systems have done so in its shadow.

A small and simple set of *roff* macros was first used for the manual pages of Unix Version 4 and persisted for two further releases, but the first macro package to be formally described and installed was *ms* by Michael Lesk in Version 6. He also wrote a manual, "Typing Documents on the Unix System", describing *ms* and basic *nroff*/*troff* usage, updating it as the package accrued features. Sixth Edition (1975) additionally saw the debut of the *tbl* preprocessor for formatting tables, also by Lesk.

For Unix Version 7 (January 1979), McIlroy designed, implemented, and documented the *man* macro package, introducing most of the macros described in *groff_man*(7) today, and edited volume 1 of the Version 7 manual using it. Documents composed using *ms* featured in volume 2, edited by Kernighan.

Meanwhile, *troff* proved popular even at Unix sites that lacked a C/A/T device. Tom Ferrin of the University of California at San Francisco combined it with Allen Hershey's popular vector fonts to produce *vtroff*, which translated *troff*'s output to the command language used by Versatec and Benson-Varian plotters.

Ossanna had passed away unexpectedly in 1977, and after the release of Version 7, with the C/A/T typesetter becoming supplanted by alternative devices such as the Autologic APS-5 and the Mergenthaler Linotron 202, Kernighan undertook a revision and rewrite of *troff* to generalize its design. To implement this revised architecture, he developed the font and device description file formats and the page description language that remain in use today. He described these novelties in the article "A Typesetter-independent TROFF", last revised in 1982, and like the *troff* manual itself, it is widely known by a shorthand, "CSTR #97".

Kernighan's innovations prepared *troff* well for the introduction of the Adobe PostScript language in 1982 and a vibrant market in laser printers with built-in interpreters for it. An output driver for PostScript, *dpost*, was swiftly developed. However, AT&T's software licensing practices kept Ossanna's *troff*, with its tight coupling to the C/A/T's capabilities, in parallel distribution with device-independent *troff* throughout the 1980s. Today, however, all actively maintained *troff*s follow Kernighan's device-independent design.

groff—a free *roff* from GNU

The most important free *roff* project historically has been *groff*, the GNU implementation of *troff*, developed by James Clark starting in 1989 and distributed under copyleft licenses, ensuring to all the availability of source code and the freedom to modify and redistribute it, properties unprecedented in *roff* systems to that point. *groff* rapidly attracted contributors, and has served as a replacement for almost all applications of AT&T *troff*. (Exceptions include *mv*, a macro package for preparation of viewgraphs and slides, and the *ideal* preprocessor, which produces diagrams from mathematical constraints.) Beyond that, it has added numerous features; see *groff_diff* (7). Since its inception and for at least the following three decades, it has been used by practically all GNU/Linux and BSD operating systems.

groff continues to be developed, is available for almost all operating systems in common use (along with several obscure ones), and is free. These factors make *groff* the *de facto roff* standard today.

Other free *roff*s

In 2007, Caldera/SCO and Sun Microsystems, having acquired rights to AT&T Unix System V *troff*, which was based on Documenter's Workbench ("DWB") 2.0, itself a descendant of Bell Labs device-independent *troff*, released it under a free but GPL-incompatible license. Gunnar Ritter and later Carsten Kunze then enhanced it to produce Heirloom Doctools *troff*. Sometime later, AT&T Software Technology made available a subsequent revision of Documenter's Workbench *troff*, DWB 3.3, under yet another free but GPL-incompatible license. Kunze again ported the software to modern POSIX systems.

In July 2013, Ali Gholami Rudi announced *neatroff*, a permissively licensed new implementation.

Another descendant of DWB *troff* is part of Plan 9 from User Space. Since 2021, this *troff* has been available under permissive terms.

Using *roff*

When you read a man page, often a *roff* is the program rendering it. Some *roff* implementations provide wrapper programs that make it easy to use the *roff* system from the shell's command line. These can be specific to a macro package, like *mmroff*(1), or more general. *groff*(1) provides command-line options sparing the user from constructing the long, order-dependent pipelines familiar to AT&T *troff* users. Further, a heuristic program, *grog*(1), is available to infer from a document's contents which *groff* arguments should be used to process it.

The *roff* pipeline

A typical *roff* document is prepared by running one or more preprocessors in series, followed by a a formatter program and then an output driver (or "device postprocessor"). Commonly, these programs are structured into a pipeline; that is, each is run in sequence such that the output of one is taken as the input to the next, without passing through secondary storage. (Non-Unix systems may simulate pipelines with temporary files.)

```
$ preproc1 < input-file | preproc2 | ... | troff [option] ... \
    | output-driver
```

Once all preprocessors have run, they deliver pure *roff* language input to the formatter, which in turn generates a document in a page description language that is then interpreted by a postprocessor for viewing, printing, or further handling.

Each program interprets input in a language that is independent of the others; some are purely descriptive, as with *tbl*(1) and *roff* output, and some permit the definition of macros, as with *eqn*(1) and *roff* input. Most *roff* input employs the macros of a document formatting package, intermixed with instructions for one or more preprocessors, and is seasoned with escape sequences and requests from the *roff* language. Some documents are simpler still, since their formatting packages discourage direct use of *roff* requests; man pages are a prominent example. Many features of the *roff* language are seldom needed by users; only authors of macro packages require a substantial command of them.

Preprocessors

A *roff* preprocessor is a program that, directly or ultimately, generates output in the *roff* language. Typically, each preprocessor defines a language of its own that transforms its input into that for *roff* or another preprocessor. As an example of the latter, *chem* produces *pic* input. Preprocessors must consequently be run in an appropriate order; *groff*(1) handles this automatically for all preprocessors supplied by the GNU *roff* system.

Portions of the document written in preprocessor languages are usually bracketed by tokens that look like *roff* macro calls. *roff* preprocessor programs transform only the regions of the document intended for them. When a preprocessor language is used by a document, its corresponding program must process it before the input is seen by the formatter, or incorrect rendering is almost guaranteed.

GNU *roff* provides several preprocessors, including *eqn*, *grn*, *pic*, *tbl*, *refer*, and *soelim*. See *groff*(1) for a complete list. Other preprocessors for *roff* systems are known.

dformat	depicts data structures;
grap	constructs statistical charts; and
ideal	draws diagrams using a constraint-based language.

Formatter programs

A *roff* formatter transforms *roff* language input into a single file in a page description language, described in *groff_out*(5), intended for processing by a selected device. This page description language is specialized in its parameters, but not its syntax, for the selected device; the format is device-*independent*, but not device-*agnostic*. The parameters the formatter uses to arrange the document are stored in *device* and *font description files*; see *groff_font*(5).

AT&T Unix had two formatters—*nroff* for terminals, and *troff* for typesetters. Often, the name *troff* is used loosely to refer to both. When generalizing thus, *groff* documentation prefers the term "*roff*". In GNU *roff*, the formatter program is always *troff*(1).

Devices and output drivers

To a *roff* system, a *device* is a hardware interface like a printer, a text or graphical terminal, or a standardized file format that unrelated software can interpret. An *output driver* is a program that parses the output of *troff* and produces instructions or markup specific to the device or file format it supports. An output driver might support multiple devices, particularly if they are similar.

The names of the devices and their driver programs are not standardized. Technological fashions evolve; the devices popular for document preparation when AT&T *troff* was first written in the 1970s are no longer used in production environments. Device capabilities have tended to increase, improving resolution and font repertoire, and adding color output and hyperlinking. Further, to reduce file size and processing time, AT&T *troff*'s page description language placed low limits on the magnitudes of some quantities it could represent. Its PostScript output driver, *dpost*(1), had a resolution of 720 units per inch; *groff*'s *grops*(1) uses 72,000.

roff programming

Documents using *roff* are normal text files interleaved with *roff* formatting elements. The *roff* language is powerful enough to support arbitrary computation and it supplies facilities that encourage extension. The primary such facility is macro definition; with this feature, macro packages have been developed that are tailored for particular applications.

Macro packages

Macro packages can have a much smaller vocabulary than *roff* itself; this trait combined with their domain-specific nature can make them easy to acquire and master. The implementation of a package *name* is typically kept in a file called "name.*tmac*" (historically, " *tmac*.name"). Find details on the naming and placement of macro packages in *groff_tmac*(5).

A macro package anticipated for use in a document can be declared to the formatter by the command-line option **−m**; see *troff*(1). It can alternatively be specified within a document using the **mso** request of the *groff* language; see *groff*(7).

Well-known packages include *man* for traditional man pages and *mdoc* for BSD-style manual pages. Packages for typesetting books, articles, and letters include *ms* (from "manuscript macros"), *me* (named by a system administrator from the first name of its creator, Eric Allman), *mm* (from "memorandum macros"), and *mom*, a punningly named package exercising many *groff* extensions. See *groff_tmac*(5) for more.

The *roff* formatting language

The *roff* language provides requests, escape sequences, macro definition facilities, string variables, registers for storage of numbers or dimensions, and control of execution flow. The theoretically minded will observe that a *roff* is not a mere markup language, but Turing-complete. It has storage (registers), it can perform tests (as in conditional expressions like "**(\n[i] >= 1)**"), its "**if**" and related requests alter the flow of control, and macro definition permits unbounded recursion.

Requests and *escape sequences* are instructions, predefined parts of the language, that perform formatting operations, interpolate stored material, or otherwise change the state of the parser. The user can define their own request-like elements by composing together text, requests, and escape sequences *ad libitum*. A document

writer will not (usually) note any difference in usage for requests or macros; both are found on control lines. However, there is a distinction; requests take either a fixed number of arguments (sometimes zero), silently ignoring any excess, or consume the rest of the input line, whereas macros can take a variable number of arguments. Since arguments are separated by spaces, macros require a means of embedding a space in an argument; in other words, of quoting it. This then demands a mechanism of embedding the quoting character itself, in case *it* is needed literally in a macro argument. AT&T *troff* had complex rules involving the placement and repetition of the double quote to achieve both aims. *groff* cuts this knot by supporting a special character escape sequence for the neutral double quote, "**\[dq]**", which never performs quoting in the typesetting language, but is simply a glyph, '**"**'.

Escape sequences start with a backslash, "\". They can appear almost anywhere, even in the midst of text on a line, and implement various features, including the insertion of special characters with "**\(*xx***" or "**\[*xxx*]**", break suppression at input line endings with "**\c**", font changes with "**\f**", type size changes with "**\s**", in-line comments with "**\"**", and many others.

Strings store text. They are populated with the **ds** request and interpolated using the * escape sequence.

Registers store numbers and measurements. A register can be set with the request **nr** and its value can be retrieved by the escape sequence **\n**.

File naming conventions

The structure or content of a file name, beyond its location in the file system, is not significant to *roff* tools. *roff* documents employing "full-service" macro packages (see *groff_tmac*(5)) tend to be named with a suffix identifying the package; we thus see file names ending in *.man*, *.ms*, *.me*, *.mm*, and *.mom*, for instance. When installed, man pages tend to be named with the manual's section number as the suffix. For example, the file name for this document is *roff.7*. Practice for "raw" *roff* documents is less consistent; they are sometimes seen with a *.t* suffix.

Input conventions

Since a *roff* formatter fills text automatically, its experienced users tend to avoid visual composition of text in input files: the esthetic appeal of the formatted output is what matters. Therefore, *roff* input should be arranged such that it is easy for authors and maintainers to compose and develop the document, understand the syntax of *roff* requests, macro calls, and preprocessor languages used, and predict the behavior of the formatter. Several traditions have accrued in service of these goals.

- Follow sentence endings in the input with newlines to ease their recognition. It is frequently convenient to end text lines after colons and semicolons as well, as these typically precede independent clauses. Consider doing so after commas; they often occur in lists that become easy to scan when itemized by line, or constitute supplements to the sentence that are added, deleted, or updated to clarify it. Parenthetical and quoted phrases are also good candidates for placement on text lines by themselves.
- Set your text editor's line length to 72 characters or fewer; see the subsections below. This limit, combined with the previous item of advice, makes it less common that an input line will wrap in your text editor, and thus will help you perceive excessively long constructions in your text. Recall that natural languages originate in speech, not writing, and that punctuation is correlated with pauses for breathing and changes in prosody.
- Use **\&** after "**!**", "**?**", and "**.**" if they are followed by space or newline characters and don't end a sentence.
- In filled text lines, use **\&** before "**.**" and "**'**" if they are preceded by space, so that revisions to the input don't turn them into control lines.
- Do not use spaces to perform indentation or align columns of a table. Leading spaces are reliable when text is not being filled. (Exception: when laying out a table with GNU *tbl*, specifying the **nospaces** region option causes the program to ignore spaces at the boundaries of table cells.)
- Comment your document. It is never too soon to apply comments to record information of use to future document maintainers (including your future self). The **\"** escape sequence causes the formatter to ignore the remainder of the input line.

- Use the empty request—a control character followed immediately by a newline—to visually manage separation of material in the input. Many of the *groff* project's own documents use an empty request between sentences, after macro definitions, and where a break is expected, and two empty requests between paragraphs or other requests or macro calls that will introduce vertical space into the document. You can combine the empty request with the comment escape sequence to include whole-line comments in your document, and even "comment out" sections of it.

An example sufficiently long to illustrate most of the above suggestions in practice follows. An arrow → indicates a tab character.

```
.\"   nroff this_file.roff | less
.\"   groff -T ps this_file.roff > this_file.ps
→The theory of relativity is intimately connected with
the theory of space and time.
.
I shall therefore begin with a brief investigation of
the origin of our ideas of space and time,
although in doing so I know that I introduce a
controversial subject.  \" remainder of paragraph elided
.
.

→The experiences of an individual appear to us arranged
in a series of events;
in this series the single events which we remember
appear to be ordered according to the criterion of
\[lq]earlier\[rq] and \[lq]later\[rq], \" punct swapped
which cannot be analysed further.
.
There exists,
therefore,
for the individual,
an I-time,
or subjective time.
.
This itself is not measurable.
.
I can,
indeed,
associate numbers with the events,
in such a way that the greater number is associated with
the later event than with an earlier one;
but the nature of this association may be quite
arbitrary.
.
This association I can define by means of a clock by
comparing the order of events furnished by the clock
with the order of a given series of events.
.
We understand by a clock something which provides a
series of events which can be counted,
and which has other properties of which we shall speak
later.
.\" Albert Einstein, _The Meaning of Relativity_, 1922
```

Editing with Emacs

Official GNU doctrine holds that the best program for editing a *roff* document is Emacs; see *emacs*(1). It provides an *nroff* major mode that is suitable for all kinds of *roff* dialects. This mode can be activated by the following methods.

When editing a file within Emacs the mode can be changed by typing "*M-x* **nroff–mode**", where *M-x* means to hold down the meta key (often labelled "Alt") while pressing and releasing the "x" key.

It is also possible to have the mode automatically selected when a *roff* file is loaded into the editor.

- The most general approach includes file-local variables at the end of the file; we can also configure the fill column this way.

```
.\" Local Variables:
.\" fill-column: 72
.\" mode: nroff
.\" End:
```

- Certain file name extensions, like those often used by man pages, activate *nroff* mode automatically.
- Loading a file with the sequence

```
.\" -*- nroff -*-
```

  in its first line into an Emacs buffer causes the editor to enter its *nroff* major mode. Unfortunately, some implementations of the *man*(1) program are confused by this practice, so we discourage it.

Editing with Vim

Other editors provide support for *roff*-style files too, such as *vim*(1), an extension of the *vi*(1) program. Vim's highlighting can be made to recognize *roff* files by setting the **filetype** option in a Vim *modeline*. For this feature to work, your copy of *vim* must be built with support for, and configured to enable, several features; consult the editor's online help topics "auto–setting", "filetype", and "syntax". Then put the following at the end of your *roff* files, after any Emacs configuration.

```
.\" vim: set filetype=groff textwidth=72:
```

Replace "groff" in the above with "nroff" if you want highlighting that does *not* recognize many of the GNU extensions to *roff*, such as request, register, and string names longer than two characters.

Authors

This document was written by Bernd Warken and G. Branden Robinson.

See also

Much *roff* documentation is available. The Bell Labs papers describing AT&T *troff* remain available, and *groff* is documented comprehensively.

Internet sites

Unix Text Processing, by Dale Dougherty and Tim O'Reilly, 1987, Hayden Books. This well-regarded text brings the reader from a state of no knowledge of Unix or text editing (if necessary) to sophisticated computer-aided typesetting. It has been placed under a free software license by its authors and updated by a team of *groff* contributors and enthusiasts.

"History of Unix Manpages", an online article maintained by the mdocml project, provides an overview of *roff* development from Saltzer's *RUNOFF* to 2008, with links to original documentation and recollections of the authors and their contemporaries.

troff.org, Ralph Corderoy's *troff* site, provides an overview and pointers to much historical *roff* information.

Multicians, a site by Multics enthusiasts, contains a lot of information on the MIT projects CTSS and Multics, including *RUNOFF*; it is especially useful for its glossary and the many links to historical documents.

The Unix Archive, curated by the Unix Heritage Society, provides the source code and some binaries of historical Unices (including the source code of some versions of *troff* and its documentation) contributed by their copyright holders.

Jerry Saltzer's home page stores some documents using the original *RUNOFF* formatting language.

groff, GNU *roff*'s web site, provides convenient access to *groff*'s source code repository, bug tracker, and mailing lists (including archives and the subscription interface).

Historical *roff* documentation

Many AT&T *troff* documents are available online, and can be found at Ralph Corderoy's site (see above) or via Internet search. Of foremost significance are those describing the language and its device-independent implementation.

"Troff User's Manual" by Joseph F. Ossanna, 1976 (revised by Brian W. Kernighan, 1992), AT&T Bell Laboratories Computing Science Technical Report No. 54.

"A Typesetter-independent TROFF" by Brian W. Kernighan, 1982, AT&T Bell Laboratories Computing Science Technical Report No. 97.

You can obtain many relevant Bell Labs papers in PDF from Bernd Warken's "roff classical" GitHub repository.

Manual pages

A componentized system like *roff* potentially has many man pages, each describing an aspect of it. Unfortunately, there is no consistent naming scheme for these pages among the various implementations.

In GNU *roff*, the *groff* (1) man page enumerates all man pages distributed with the system, and individual pages frequently refer to external resources as well as manuals on a variety of topics imbricated with *groff*.

In other *roff*s, you are on your own, but *troff* (1) might be a good starting point.